Hyperiondev

TASK

Sources of Data

Visit our website

Copyright © 2021 HyperionDev. All rights reserved.

Introduction

WELCOME TO THE SOURCES OF DATA TASK!

Congratulations on making it to Level 2 of the Data Science Bootcamp! As a Data Scientist, before you start analysing data, you will first need to obtain data from different sources. Once you have raw data, you may then proceed to clean, transform, visualise, and communicate. This task will introduce you to several data sources that are used with Python as well as basic ways to manipulate data.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

Copyright © 2021 HyperionDev. All rights reserved.

DATA SOURCES

Data scientists need data! The data comes from various sources. Sometimes data is downloaded from the internet or obtained using a web API, or it may be obtained from a group of researchers directly. Wherever data comes from, it generally ends up as a file on a computer.
Before we begin working with files, let us understand File Systems. Your computer drive is organised in a hierarchical structure of files and directories (washington.edu, n.d.):

- **Files:** A file is a resource on your computer that contains and stores information. There are different types of files that serve different purposes - storing audio, video, text, settings, program files, and so on. For the purpose of data extraction, some common file types you will encounter are CSV, XML, JSON and Python files. You will learn about working with some of these files in this task.

- **Directories:** A directory is a cataloguing structure which contains references to other computer files and possibly other directories. Simply put, a directory is a location for storing files on your computer.

Your file system starts with a **root directory**, typically notated by a forward slash ('/') on Unix and by a drive letter (e.g. 'C:/') on Windows (washington.edu, n.d.). Now that we are familiar with file systems, we can dive into the different types of files that can be used as sources of data.

JSON

JSON stands for JavaScript Object Notation. It is a syntax used for storing and exchanging unstructured data. As the name suggests, JSON is derived from the JavaScript programming language but is used widely in other programming languages, such as Java and Ruby. (Tagliaferri, 2016).

The syntax for a JSON object looks like a Python dictionary. It is a key-value structure enclosed with curly braces. Remember that a key-value structure is a data structure which uses an associative array of keys, where each key is associated with only one value in a collection. JSON objects can be found in a file with a .json extension. In the example below, notice the syntax used for a JSON file that contains an array of book objects:

Copyright © 2021 HyperionDev. All rights reserved.

```
{
	"books": [{
		"title": "Homegoing",
		"author": "Yaa Gyasi",
		"year": "2002",
		"edition": 2
	}, {
		"title": "The river between",
		"author": "Ngugi wa Thiong'o",
		"year": "2004"
	}]
}
```

JSON keys are on the left side of the colon. Within each object, keys need to be unique. JSON objects can be different (for instance, in the code snippet above, we can see that the second object does not have the 'edition' key). This is referred to as unstructured data.

JSON files are commonly used since this format is easy to transmit between a web server and a client or browser. The JSON values can be any of the following data types: strings, numbers, objects, arrays, booleans, or null.

A note from our coding mentor
Valerie

When you are new to working with JSON, it may not be easy to spot syntax errors in a JSON file. If you'd like to check if a JSON file is formatted correctly, use a JSON linter ***like this one****. For more information about correctly formatting a JSON file, see* ***here****.*

See another example of how JSON objects can be nested within other JSON objects below:

```
{
	"categories": [{
		"Comedy": [{
				"title": "The Big Bang Theory"
			},
```


Copyright © 2021 HyperionDev. All rights reserved.

```
                {
                        "title": "Melissa and Joey"
                },
                {
                        "title": " Scorpion"
                }
        ],
        "Thrillers": [{
                        "title": "The Dark Knight"
                },
                {
                        "title": "Inception"
                }
        ]
    }]
}
```

Converting Python Objects to JSON

There are a couple of packages that support JSON in Python, such as **simplejson** and **json**. In this task, we'll use the json package, which is natively supported by Python.

JSON is commonly used in Python when dealing with application programming interfaces (APIs). An API is an interface by which software applications communicate with each other. Tech companies, like Amazon or Google, release their APIs to the public so that other software developers can design software that is powered by API services.

You can often use APIs to retrieve data from a third party. APIs frequently return JSON files.

In the example below, we use the json package and extract data from an actual API. We use a URL to access an API. In this case, we use a URL that allows us to access data from the **API provided by indeed.com**. The data that is returned by the API is in JSON format.

```
# import the modules that we need

import json
import requests # Used to open a URL link in Python

# Open the URL
```


Copyright © 2021 HyperionDev. All rights reserved.

```
url='http://api.indeed.com/ads/apisearch?publisher=4407976915591140&q=java&l
=austin&v=2&format=json'

response = requests.get(url)

# save the data in a json file
with open('data.json', 'w', encoding='utf-8') as outfile:
    outfile = json.dump(response.json(), outfile, sort_keys=True, indent=4)
```

The data from the URL is now saved into a JSON file in your current active directory.

Reading JSON Files

Now let's see how to read the JSON file that we've just created. We use `json.load` to load the files:

```
with open('data.json', 'r') as j:
    json_data = json.load(j)
    print(json_data)
```

(Read more about the JSON module on Python's official **webpage**.)

XML

XML files are also commonly used to store data used by data scientists. XML (eXtensible Markup Language) files are also used to transmit data over the internet. As a data scientist, you'll need to understand how to work with XML so that you can parse the data contained in XML files (Tagliaferri, 2016).

Whereas XML must be parsed with an XML parser, JSON can be parsed with a standard Python function. Since JSON is closer in format to Python data structures, like maps and dictionaries, it is easier to parse than XML files.

XML Tags, elements, and attributes

Let us look at a sample XML document:

Copyright © 2021 HyperionDev. All rights reserved.

```
<menu>
	<meal type="Breakfast">
		<name>Belgian Waffles</name>
		<total>$5.95</total>
		<description>
Two of our famous Belgian Waffles with plenty of real maple syrup
</description>
	</meal>
	<meal type="Lunch">
		<name>Strawberry Belgian Waffles</name>
		<total>$7.95</total>
		<description>
Light Belgian waffles covered with strawberries and whipped cream
</description>
	</meal>
</menu>
```

An XML document can consist of tags, elements, and attributes. A **tag** is the text between the left angle bracket `<` and the right angle bracket `>`. Every tag has a starting tag and a closing tag. For example, `<menu>` and `</menu>` are the starting and closing tags for the document.

An **element** can be described as the starting tag, the ending tag and everything in between. In the sample above, the `<meal>` element is the root element, which contains three child elements: `<name>`, `<total>` and `<description>`. Note that these child elements can also act as parents and contain their own child elements, which are then called "sub-child elements".

An **attribute** is found in the starting tag of an element. It is a key-value pair that gives more information about the element. In the above example, 'type' is an attribute of the <meal> element. You can use attributes to filter which element you would like to work with. You can check the syntax of your XML file by using an XML validator such as **this one**.

What is an Element Tree?

It is easy to navigate an XML tree structure programmatically. We are going to use a library named **ElementTree**. It is a simple library that allows us to read XML files while returning a Python object that reflects the nodes and attributes.

Take the sample XML above and save it in a file called **movies.xml**. Now using the code below, we will be able to load the file:

Copyright © 2021 HyperionDev. All rights reserved.

```
import xml.etree.ElementTree as ET

tree = ET.parse('movies.xml')
root = tree.getroot()
```

Once we have initialised the tree, we can print out the values to understand how the tree is formatted. You will notice that every part of the tree has a tag associated with it. We'll see below how to use the tags to traverse the tree.

The code below will print the root element. You will see that the root tag is `<menu>`.

```
print(root.tag)
```

According to the line below, the root does not have any attributes.

```
print(root.attrib)
```

For loops

You can iterate over the child nodes of an XML element as shown below:

```
for child in root:
    print(child.tag, child.attrib)
```

CSV

CSV stands for Comma Separated Values. CSV files are supported by many tools like spreadsheets (like Excel, OpenOffice and Google Docs), complex databases and almost all programming languages. As such, it is probably the most widely supported structured data format. CSV data is easily readable to a person and to a machine.

A CSV is the simplest possible structured format for data. A CSV file contains a list of data separated by commas. CSV files are used for exchanging large chunks of data between systems.

A CSV is a two-dimensional structure consisting of rows of data, with each row containing multiple cells. Rows are usually separated by line terminators, so each row corresponds to one line (Data Hub, 2018). Cells within a row are separated by commas (hence the Commas part in "CSV").

Copyright © 2021 HyperionDev. All rights reserved.

The structure of the CSV file is shown in the example from Data Hub below:

```
A, B, C, D
1, 2, 3, 4
4, "5,3", 6, 7.3
```

In the example above, there are 3 rows and 4 columns. For this example, let us assume that the first row is the header row which sets out the column names and that the data is contained in the two rows below this header row (Data Hub 2018).

Also note in the second row of data (third row in the example above, since we're considering the first row to be the header), there is a value which has double quotation marks. This is because it contains two digits with a comma in between, but we want the computer to disregard the comma inside the quotation marks and essentially read in "5,3" as one value.

A note from our coding mentor
Sarah

Additional reading: *Jeroen Janssens has published a book entitled "**Data Science at the Command Line**". **Chapter 3** of the book deals with "Obtaining data" and contains excellent information about converting spreadsheets, calling APIs and querying databases. For more information about getting and formatting data, please refer to this excellent resource for optional additional reading.*

Instructions

Before you get started, we strongly suggest you start using Notepad++ or IDLE to open all text files (.txt) and Python files (.py). Do not use the normal Windows Notepad as it will be much harder to read.

Copyright © 2021 HyperionDev. All rights reserved.

Compulsory Task 1

Follow these instructions:

- Create a file named **books.json**.
- You are going to populate the file with data about books. Have at least 6 books and choose four attributes, such as the author and year of publication. Create JSON objects for the books. Make sure your JSON string is valid by Parsing it through this **site**.

Compulsory Task 2

Follow these instructions:

- Create a python program called **taskXML.py**. Write the code to:
    - Read in the **movie.xml** file.
    - Read about the `iter()` and `itertext()` function **here**. Use the `iter()` function to list all the child tags of the movie element.
    - Use the `itertext()` function to print out the movie descriptions.
    - Find the number of movies that are favourites and the number of movies that are not.

Completed the task(s)?

Ask your mentor to review your work!

Review work

Copyright © 2021 HyperionDev. All rights reserved.

Rate us
Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved? Do you think we've done a good job?

Click here to share your thoughts anonymously.

References:

Data Hub. (2018). CSV - Comma Separated Values. Retrieved from datahub.io: https://datahub.io/docs/data-packages/csv

Tagliaferri, L. (2016, December 8). An Introduction to JSON. Retrieved from digitalocean.com: https://www.digitalocean.com/community/tutorials/an-introduction-to-json

washington.edu. (n.d.). How do I interact with files in Python? Retrieved May 14, 2019, from courses.cs.washington.edu: https://courses.cs.washington.edu/courses/cse140/13wi/file-interaction.html

Copyright © 2021 HyperionDev. All rights reserved.