Python Requests Package: How to Download Web Files?

Python Tutorials 3 min read

Why use Python 'requests' package?

Data collection is an integral step of every company's data analysis pipeline.

Lucky for data science professionals, there are many ways to obtain useful data – through a company's internal data collection mechanisms, by taking advantage of APIs (https://365datascience.com/data-connectivity/) or just by downloading a relevant file from the web.

When you browse on this site, cookies and other technologies collect data to enhance your experience and personalize the content and advertising you see. Ok, got it! Read More (https://365datascience.com/privacy-policy/)

In this tutorial, we will discuss one of these methods – we will show you how to programmatically download files from the web with the help of the Python 'requests (https://requests.readthedocs.io/en/master/)' library.

But let's begin with a couple of words about the 'requests' package.

What is Python 'requests' package?

It is a user-friendly implementation of the HTTP request abstract concept (https://www.w3schools.com/tags/ref_httpmethods.asp). That is to say that you as a user, don't need to know exactly how HTTP requests work. Instead, you can use the simple interface of the 'requests' module. It is one of the most popular Python packages for this purpose.

Now that we've briefed you about Python 'requests' package, let's go ahead and examine a rather naïve approach to downloading files.

If you think about it for a bit, you may realize that *connecting to a webpage on the web* is practically the same as *downloading its contents*.

By the same logic, if there is a file we wish to download, we can make a GET request to it, catch the response and then export that response to a local file on our machine… Which is practically downloading the file.

For the purposes of this example, we can use this (https://upload.wikimedia.org/wikipedia/commons/thumb/d/d9/Collage_of_Nine_Dogs.jpg/1024px-Collage_of_Nine_Dogs.jpg) Wikipedia image as the file we'd like to obtain.

So, here are the steps to downloading a file using Python 'requests' package

The first thing we need to do is to import 'requests'.

```
In [1]: import requests

        file_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/d/d9/Collage_of_Nine_Dogs.jpg/1024px-Collage_of_Nine_Dogs.jpg"

        response = requests.get(file_url)
```

Then, for simplicity, save the URL of the file in a variable.

The next step is to request this file from the server.

When you browse on this site, cookies and other technologies collect data to enhance your experience and personalize the content and advertising you see. Ok, got it! Read More (https://365datascience.com/privacy-policy/)

This is where the Python 'requests' package comes into play – we submit a GET request using its *.get()* method. Yes, it is that easy.

At this point, we've actually downloaded the data, so we just need to write it out to a local file.

The good news is that dealing with files in Python is not that hard.

We open or create files with the standard 'open' function. It returns the file as a Python object. This way, we store it in a variable.

```
In [2]: file = open("dog_image.jpg", "wb")
        file.write(response.content)
        file.close()
```

This function takes 2 main parameters.

The first one is the name of the file we want to open or create along with its type.

The second one is more interesting. It specifies the mode in which we open the file. There are several options in this department. For instance, the most popular ones are:

- 'r': Opens the file in read-only mode;
- 'rb': Opens the file as read-only in binary format;
- 'w': Creates a file in write-only mode. If the file already exists, it will overwrite it;
- 'wb': Write-only mode in binary format;
- 'a': Opens the file for appending new information to the end;
- 'w+': Opens the file for writing and reading;

Now you understand what the first line does – it creates a JPG file for writing in binary.

Next, to write to the file, we simply pass the response content to the write method.

Finally, we should close the file at the end.

This is important for the proper execution of programs and saving of the file.

This has now exported the downloaded contents of the Wiki image to a local file.

There's one last trick about dealing with files.

To ensure that the file is always closed, no matter what we do with it, we can use the '*with*' statement as shown below. It automatically calls the close method at the end.

When you browse on this site, cookies and other technologies collect data to enhance your experience and personalize the content and advertising you see. Ok, got it! Read More (https://365datascience.com/privacy-policy/)

```
In [3]: with open("dog_image_2.jpg", "wb") as file:
            file.write(response.content)
```

So, what's the problem with this method of downloading files?

Well, the more eagle-eyed may have noticed that we first received the whole file through the GET request and then we went through its entirety to write it on the hard disk.

The main issue with this is that the file is first stored entirely in the RAM before being transferred to the Hard Drive. The RAM is usually not designed for this purpose and this can really slow down the process for bigger files and potentially overflow and crash.

Luckily, Python 'requests' package does provide a solution to this dilemma.

To illustrate this point, we can try to download a sample video file provided by the file-examples.com (https://file-examples.com/wp-content/uploads/2017/04/file_example_MP4_480_1_5MG.mp4) website. Here is the code:

```
In [4]: url = "https://file-examples.com/wp-content/uploads/2017/04/file_example_MP4_480_1_5MG.mp4"

        r = requests.get(url, stream = True)

        with open("Sample_video_1,5_MB.mp4", "wb") as f:

            # Now we iterate over the response in chunks
            for chunk in r.iter_content(chunk_size = 16*1024):
                f.write(chunk)
```

As you see, in the GET request we should set the stream parameter to 'True'. This tells the program that the file will be downloaded in several smaller parts as opposed to in one go.

Then, we open the file using the 'with' statement we just talked about. This time, however, we can't just write the whole response to the file, we need to do it in chunks.

That's why we have a 'for loop'.

A thing to note here is the 'chunk size' parameter. This denotes how big of a chunk should be read at a time, which may increase or decrease the speed of the download. The optimal size depends on your system and internet speed, so you may need to experiment a bit with it.

This method of downloading files is much faster as it uses the RAM to store only one chunk at a time.

We and our partners on this site, cookies and other technologies collect data to enhance your experience and personalize the content and advertising you see. Ok, got it! Read More (https://365datascience.com/privacy-policy/)

One final note is that when setting the stream parameter, the connection to the web server may potentially need to be closed. Thus, we are going to use another 'with' for the request. The final code looks like this.

```
In [5]: url = "https://file-examples.com/wp-content/uploads/2017/04/file_example_MP4_1920_18MG.mp4"

with requests.get(url, stream = True) as r:
    with open("Sample_video_18_MB.mp4", "wb") as f:
        for chunk in r.iter_content(chunk_size = 16*1024):
            f.write(chunk)
```

So, this is one way to obtain useful data. After that, however, you still need to analyze it. That said, if you want to strengthen your analytical skillset, check out our complete Data Science program. It is designed to help you develop all in-demand competencies for a data scientist or a data analyst job. If you still aren't sure you want to turn your interest in data science into a full-scale career, we also offer a free preview of the Data Science Program (https://365datascience.com/courses/). You'll receive 12 hours of beginner to advanced content for free. It's a great way to see if the program is right for you.

(https://365datascience.com/courses/)

Earn your Data Science Degree

Expert instructions, unmatched support and a verified certificate upon completion!

When you browse on this site, cookies and other technologies collect data to enhance your experience and personalize the content and advertising you see. Ok, got it! Read More (https://365datascience.com/privacy-policy/)

Go To Courses (Https://365datascience.Com/Courses/) See Pricing (Https://365datascience.Com/Pr

Related Posts

(https://365datascience.com/python-else-statement-exercise/)

Python ELSE Statement Exercise (https://365datascience.com/python-else-statement-exercise/)

1 Question Exercises

(https://365datascience.com/basic-python-syntax/)

Basic Python Syntax – Introduction to Syntax and Operators (https://365datascience.com/basic-python-syntax/)

18 min read Python Tutorials

(https://365datascience.com/combining-statements-functions-exercise/)

Combining Python Statements and Functions Exercise: (https://365datascience.com/combining-statements-functions-exercise/)

1 Question Exercises

(https://365datascience.com/declare-python-variables/)

How to Declare Python Variables (https://365datascience.com/declare-python-variables/)

6 min read Python Tutorials

When you browse on this site, cookies and other technologies collect data to enhance your experience and personalize the content and advertising you see. Ok, got it! Read More (https://365datascience.com/privacy-policy/)

Leave a Reply

Your email address will not be published.

COMMENT

NAME

EMAIL

POST COMMENT

(https://www.linkedin.com/company/365datascience/)

© 2020 365 Data Science. All Rights Reserved

When you browse on this site, cookies and other technologies collect data to enhance your experience and personalize the content and advertising you see. Ok, got it! Read More (https://365datascience.com/privacy-policy/)