Scrapy Beginners Series Part 3: Storing Data With Scrapy

In **Part 1** and **Part 2** of this **Python Scrapy 5-Part Beginner Series** we learned how to build a basic scrapy spider and get it to scrape some data from a website as well as how to clean up data as it was being scraped.

In **Part 3** we will be exploring how to save the data into files/formats which would work for most common use cases. We'll be looking at how to save the data to a CSV or JSON file as well as how to save the data to a database or S3 bucket.

Python Scrapy 5-Part Beginner Series

- **Part 1: Basic Scrapy Spider -** We will go over the basics of Scrapy, and build our first Scrapy spider. (Part 1)
- **Part 2: Cleaning Dirty Data & Dealing With Edge Cases -** Web data can be messy, unstructured, and have lots of edge cases. In this tutorial we will make our spider robust to these edge cases, using Items, Itemloaders and Item Pipelines. (Part 2)
- **Part 3: Storing Our Data -** There are many different ways we can store the data that we scrape from databases, CSV files to JSON format, and to S3 buckets. We will explore several different ways we can store the data and talk about their Pro's, Con's and in which situations you would use them. (This Tutorial)
- **Part 4: User Agents & Proxies -** Make our spider production ready by managing our user agents & IPs so we don't get blocked. (Part 4)
- **Part 5: Deployment, Scheduling & Running Jobs -** Deploying our spider on a server, and monitoring and scheduling jobs via ScrapeOps. (Part 5)

The code for this project is available on Github here!

In this tutorial, **Part 3: Storing Data With Scrapy** we're going to cover:

- Using Feed Exporters
- Saving Data to a JSON or CSV file
- Saving Data to Amazon S3 Storage
- Saving Data to a Database

With the intro out of the way let's get down to business.

Using Feed Exporters

Scrapy already has a way to save the data to several different formats. Scrapy call's these ready to go export methods **Feed Exporters**.

Out of the box scrapy provides the following formats to save/export the scraped data:

- JSON file format
- CVS file format
- XML file format
- Pythons pickle format

The files which are generated can then be saved to the following places using a Feed Exporter:

- The machine Scrapy is running on (obviously)
- To a remote machine using FTP (file transfer protocall)
- To Amazon S3 Storage
- To Google Cloud Storage
- Standard output

In this guide we're going to give examples on how your can use Feed Exporters to store your data in different file formats and locations. However, there are many more ways you can store data with Scrapy.

Saving Data to a JSON or CSV File

We've already quickly looked at how to export the data to JSON and CSV in part one of this series but we'll quickly go over how to store the data to a **JSON** file and a **CSV** file one more time. Feel free to skip ahead if you know how to do this already!

To get the data to be saved in the most simple way for a once off job we can use the following commands:

Saving in JSON format

To save to a JSON file simply add the flag `-o` to the `scrapy crawl` command along with the file path you want to save the file to:

```
scrapy crawl chocolatespider -o my_scraped_chocolate_data.json
```

You can also define an absolute path like this:

```
scrapy crawl chocolatespider -O file:///path/to/my/project/my_scraped_chocolate_data.json:json
```

Saving in CSV format

To save to a CSV file add the flag `-o` to the `scrapy crawl` command along with the file path you want to save the file to:

```
scrapy crawl chocolatespider -o my_scraped_chocolate_data.csv
```

You can also define an absolute path like this:

```
scrapy crawl chocolatespider -O file:///path/to/my/project/my_scraped_chocolate_data.csv:csv
```

You can also decide whether to overwrite or append the data to the output file.

For example, when using the crawl or runspider commands, you can use the `-O` option instead of `-o` to overwrite the output file. (Be sure to remember the difference as this might be confusing!)

Saving Data to Amazon S3 Storage

Now that we have saved the data to a CSV file, lets save the created CSV files straight to an Amazon S3 bucket (You need to already have one setup).

You can check out how to set up an S3 bucket with amazon here: https://docs.aws.amazon.com/AmazonS3/latest/userguide/setting-up-s3.html

OK- First we need to install Botocore which is an external Python library created by Amazon to help with connecting to S3.

```
pip3 install botocore
```

Now that we have that installed we can save the file to S3 by specifying the URI to your Amazon S3 bucket:

```
scrapy crawl chocolatespider -O s3://aws_key:aws_secret@mybucket/path/to/myscrapeddata.csv:csv
```

Obviously you will need to replace the `aws_key` & `aws_secret` with your own **Amazon Key & Secret**. As well as putting in your **bucket name** and **file path**. We need the `:csv` at the end to specify the format but this could be `:json` or `:xml`.

You can also save the `aws_key` & `aws_secret` in your project settings file:

```
AWS_ACCESS_KEY_ID = 'myaccesskeyhere'
AWS_SECRET_ACCESS_KEY = 'mysecretkeyhere'
```

Note: When saving data with this method the AWS S3 Feed Exporter uses delayed file delivery. This means that the file is first temporarily saved locally to the machine the scraper is running on and then it's uploaded to AWS once the spider has completed the job.

Saving Data to MySQL and PostgreSQL Databases

Here well show you how to save the data to MySQL and PostgreSQL databases. To do this we'll be using **Item Pipelines** again.

For this we are presuming that you already have a database setup called `chocolate_scraping`.

For more information on setting up a MySQL or Postgres database check out the following resources:

Windows: MySQL - Postgres

Mac: MySQL - Postgres

Ubuntu: MySQL - Postgres

Saving data to a MySQL database

We are assuming you have already have a database setup and a table called `chocolate_products` in your DB. If not you can login to your database and run the following command to create the table:

```
CREATE TABLE IF NOT EXISTS chocolate_products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    price VARCHAR(255),
    url TEXT
);
```

To save the data to the databases we're again going to be using the Item Pipelines. If you don't know what they are please check out part 2 of this series where we go through how to use Scrapy Item Pipelines!

The first step in our new Item Pipeline class, as you may expect is to connect to our MySQL database and the table in which we will be storing our scraped data.

We are going to need to install the `mysql` package for Python.

```
pip install mysql
```

If you already have mysql installed on your computer - you might only need the connection package.

```
pip install mysql-connector-python
```

Then create a Item pipeline in our `pipelines.py` file that will connect with the database.

```
import mysql.connector

class SavingToMySQLPipeline(object):

    def __init__(self):
        self.create_connection()

    def create_connection(self):
        self.conn = mysql.connector.connect(
            host = 'localhost',
            user = 'root',
            password = '123456',
            database = 'chocolate_scraping'
        )
        self.curr = self.conn.cursor()
```

Now that we are connecting to the database, for the next part we need to save each chocolate product we scrape into our database item by item as they are processed by Scrapy.

To do that we will use the scrapy `process_item()` function (which runs after each item is scraped) and then create a new function called `store_in_db` in which we will run the MySQL command to store the Item data into our `chocolate_products` table.

```
import mysql.connector

class SavingToMySQLPipeline(object):

    def __init__(self):
        self.create_connection()

    def create_connection(self):
        self.connection = mysql.connector.connect(
            host = 'localhost',
            user = 'root',
            password = '123456',
            database = 'chocolate_scraping'
        )
        self.curr = self.connection.cursor()

    def process_item(self, item, spider):
        self.store_db(item)
        #we need to return the item below as Scrapy expects us to!
        return item

    def store_db(self, item):
        self.curr.execute(""" insert into chocolate_products ( name, price, url)  values (%s,%s,%s)""", (
            item["name"],
            item["price"],
            item["url"]
        ))
        self.connection.commit()
```

Before trying to run our pipeline we mustn't forget to add the pipeline to our `ITEM_PIPELINES` in our project `settings.py` file.

```
ITEM_PIPELINES = {
    'chocolatescraper.pipelines.PriceToUSDPipeline': 100,
    'chocolatescraper.pipelines.DuplicatesPipeline': 200,
    'chocolatescraper.pipelines.SavingToMySQLPipeline': 300,
}
```

Saving data to a PostgreSQL database

As in the above section - we are assuming you have already have a postgres database setup and you have created a table called `chocolate_products` in your DB. If not you can login to your postgres database and run the following command to create the table:

```
CREATE TABLE IF NOT EXISTS chocolate_products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    price VARCHAR(255),
    url TEXT
        );
```

To save the data to a PostgreSQL database the main thing we need to do is to update how the connection is created. To do so we will will install the Python package `psycopg2`.

```
pip install psycopg2
```

And update the connection library in our function.

```
import psycopg2

class SavingToPostgresPipeline(object):

    def __init__(self):
        self.create_connection()


    def create_connection(self):
        self.connection = psycopg2.connect(
            host="localhost",
            database="chocolate_scraping",
            user="root",
            password="123456")

        self.curr = self.connection.cursor()


    def process_item(self, item, spider):
        self.store_db(item)
        #we need to return the item below as scrapy expects us to!
        return item

    def store_db(self, item):
        try:
            self.curr.execute(""" insert into chocolate_products (name, price, url) values (%s, %s, %s)""", (
                item["name"],
                item["price"],
                item["url"]
            ))

        except BaseException as e:
            print(e)
            self.connection.commit()
```

Again before trying to run our pipeline we mustn't forget to add the pipeline to our `ITEM_PIPELINES` in our project `settings.py` file.

After running our spider again we should be able to see the data in our database if we run a simple select command like the following(after logging into our database!):

```
select * from chocolate_products;
    'chocolatescraper.pipelines.PriceToUSDPipeline': 100,
    'chocolatescraper.pipelines.DuplicatesPipeline': 200,
    'chocolatescraper.pipelines.SavingToPostgresPipeline': 300,
}
```

We hope you now have a good understanding of how to save the data you've scraped into the file or database you need! If you have any questions leave them in the comments below and we'll do our best to help out!

If you would like the code from this example please check it out on Github.

The next tutorial covers how to make our spider production ready by managing our user agents & IPs so we don't get blocked. (Part 4)

Need a Free Proxy? Then check out our **Proxy Comparison Tool** that allows to compare the pricing, features and limits of every proxy provider on the market so you can find the one that best suits your needs. Including the best free plans.