DS605 Fundamentals of Machine Learning
Academic Year 2025-26 (Autumn)

LAB-2
Data Scraping using Scrapy and Data Preprocessing

What is Scrapy?

Scrapy is a powerful and fast **Python framework** used to extract data from websites, process it, and save it (e.g., in CSV, JSON, databases).

> It's not just a library—**Scrapy is a full-blown framework for building web crawlers.**

How Scrapy Works (Big Picture)

You write a Spider → Spider sends requests → Responses are parsed → Data is extracted → Output is stored

Create Python Virtual Environment:

```
python3 -m venv venv
source venv/bin/activate
pip install scrapy
```

Objective: This lab will guide you through the process of extracting data from a website using the Scrapy framework and then cleaning and preparing that data for analysis using the pandas library.

Prerequisites:

- Python 3.x installed.
- Basic understanding of Python and HTML.
- pip (Python's package installer) available in your terminal or command prompt.

1. Introduction to Web Scraping & Scrapy

What is Web Scraping?

Web Scraping (or data scraping) is the technique of automatically extracting large amounts of data from websites. The data on websites is typically unstructured and embedded in HTML. Web scraping tools parse this HTML to extract the desired information and save it in a structured format like CSV, JSON, or a database.

Why use Scrapy?

Scrapy is a powerful, open-source, and collaborative web crawling framework written in Python. It's designed to be fast, simple, and extensible. Instead of just making requests and parsing HTML, Scrapy provides a complete architecture for building "spiders" – automated bots that can crawl websites and extract data.

Scrapy Architecture

Scrapy's process is asynchronous, meaning it doesn't wait for one request to finish before starting another, which makes it very efficient.

The main components are:

- **Scrapy Engine:** Controls the data flow between all components.
- **Scheduler:** Manages the queue of URLs to be crawled.
- **Downloader:** Fetches the web pages.
- **Spiders:** You write these! They define how to follow links and what data to extract from the pages.
- **Item Pipelines:** Processes the data extracted by the spiders. This is where data cleaning, validation, and storage happen.

---

2. Setting Up Your Environment

First, you need to install the necessary Python libraries. Open your terminal or command prompt and run the following command:

Bash

```
pip install scrapy pandas
```

- scrapy: The web scraping framework.
- pandas: The library we'll use for data preprocessing.

We will scrape quotes, authors, and tags from the website http://quotes.toscrape.com, a site designed specifically for scraping practice.

Step 1: Create a Scrapy Project

Navigate to the directory where you want to store your project and run:

```
scrapy startproject quotes_scraper
```

This command creates a new directory named quotes_scraper with the following structure:

```
quotes_scraper/
    scrapy.cfg            # deploy configuration file
    quotes_scraper/       # project's Python module
        __init__.py
        items.py          # define the data containers
        middlewares.py
        pipelines.py      # process items from spiders
        settings.py       # project settings
        spiders/          # directory for your spiders
```

```
            __init__.py
```

Step 2: Define the Item

An Item in Scrapy is like a container or a dictionary that will hold the scraped data. Open the quotes_scraper/items.py file and define the fields for the data you want to scrape.

```
# quotes_scraper/items.py

import scrapy

class QuoteItem(scrapy.Item):
    # define the fields for your item here like:
    text = scrapy.Field()
    author = scrapy.Field()
    tags = scrapy.Field()
```

Step 3: Write the Spider

A **Spider** is a class where you define the initial requests, how to follow links on pages, and how to parse the page content to extract data into Items.

Inside the spiders directory, create a new file named quotes_spider.py.

```
Bash
# Navigate into the main project directory first
cd quotes_scraper
# Now, you are in the directory with the second quotes_scraper folder
```

Now, add the following code to **quotes_scraper/spiders/quotes_spider.py:**

```
import scrapy
from ..items import QuoteItem
```

```
class QuotesSpider(scrapy.Spider):
    name = "quotes"  # Unique name for the spider
    start_urls = [
        'http://quotes.toscrape.com/page/1/',
    ]

    def parse(self, response):
        # This method is called to handle the response downloaded for each request
made.

        # We use CSS selectors to find the HTML elements containing the data.
        all_div_quotes = response.css('div.quote')

        # Create an instance of our item
        items = QuoteItem()

        for quote_div in all_div_quotes:
            # Extract data using CSS selectors
            text = quote_div.css('span.text::text').extract_first()
            author = quote_div.css('.author::text').extract_first()
            tags = quote_div.css('.tag::text').extract()

            # Populate the item fields
            items['text'] = text
            items['author'] = author
            items['tags'] = tags

            # Yield the populated item to the Scrapy engine
            yield items

        # Find the 'Next' button link and follow it if it exists
        next_page = response.css('li.next a::attr(href)').get()

        if next_page is not None:
            # The 'response.follow' method handles relative URLs automatically
        yield response.follow(next_page, callback=self.parse)
```

- name: Identifies the Spider. It must be unique within a project.
- start_urls: A list of URLs where the Spider will begin to crawl from.
- parse(): A method that will be called to handle the response downloaded for each of the requests made. The response parameter is an instance of TextResponse that holds the page content. We use it to extract the data and find new URLs to follow.
- yield: Scrapy uses yield to return the extracted data as an Item and to schedule new requests to be followed.

Step 4: Run the Spider

Now it's time to run your spider and save the output. Navigate to the project's top-level directory (the first quotes_scraper folder) in your terminal and run:

scrapy crawl quotes -o quotes.csv

- scrapy crawl quotes: This command tells Scrapy to run the spider named "quotes".
- -o quotes.csv: This is an output flag. -o stands for output, and it tells Scrapy to save the scraped data into a file named quotes.csv. Scrapy can also export to other formats like JSON (-o quotes.json) or XML.

After the command finishes, you will find a quotes.csv file in your project directory containing all the scraped quotes!

Resources:

https://docs.scrapy.org/en/latest/intro/tutorial.html

Assignment 1 :- https://books.toscrape.com/