## Scrapy For Data Analyst Jobs

This project scrap data from the website "<https://www.reed.co.uk/jobs/data-analyst-jobs>" where we get 2966 and more job data available, and stored the scraped data in JSON and CSV file having fields Detail URL, Title, Salary, Contract Type, Job Type and Location.

## Getting Started

If you already have python, skip to step 2

1. Install Python on your operating system. You can either download python from the official site and install on your own or you can follow these python installation tutorials below:

* For Windows -> <https://kinsta.com/knowledgebase/install-python/>
* For Mac -> <https://kinsta.com/knowledgebase/install-python/>
* For Linux -> <https://kinsta.com/knowledgebase/install-python/>

1. Now, create a new folder for your project in your laptop then. Install virtaul environment. Virtual environment is a tool to create isolated Python projects. You can also install on your terminal (mac/linux) or command line (windows).

*pip install virtualenv*

*Inside your project folder create a new virtaulenv*

*python -m venv new\_venv*

*Then activate it*

*new\_venv\Scripts\activate*

For linux -> <https://www.geeksforgeeks.org/creating-python-virtual-environment-windows-linux/>

For mac -> <https://programwithus.com/learn/python/pip-virtualenv-mac>

1. Install Scrapy (anaconda comes with it, but just in case). You can also install on your terminal (mac/linux) or command line (windows).

*pip install Scrapy*

## Creating a new Scrapy Project

Open a terminal (max/linux) or command line (windows).

*scrapy startproject new\_project*

This makes a new\_project directory with the following contents:

|  |
| --- |
|  |

* **settings.py** is where all your project settings are contained, like activating pipelines, middlewares etc. Here you can change the delays, concurrency, and lots more things.
* **items.py** is a model for the extracted data. You can define a custom model (like a NewProjectItem) that will inherit the Scrapy Item class and contain your scraped data.
* **pipelines.py** is where the item yielded by the spider gets passed, it’s mostly used to clean the text and connect to file outputs or databases (CSV, JSON SQL, etc).
* **middlewares.py** is useful when you want to modify how the request is made and scrapy handles the response.
* **scrapy.cfg** is a configuration file to change some deployment settings, etc.

## Creating Spider

Now, create spider that will do the scraping. Scrapy provides a number of different sider types, here we use the common one, the generic Spider. To create a new generic spider, run the following command

#scrapy genspider name\_of\_spider website

*scrapy genspider jobspider* [*https://www.reed.co.uk/jobs/data-analyst-jobs*](https://www.reed.co.uk/jobs/data-analyst-jobs)

A new spider has been added to spiders folder and it look like



* start\_urls should contain the url which we want to scrape. Here, we can change the url as we want but in this project, we are going to scrape data from same url so, we don’t have to change.
* Inside parse function, we are going to write code for scraping

### **Finding XPath or CSS Selector**

To extact data from a HTMl Page, use XPath or CSS selectors to tell Scrapy where in the page is the data. XPath and CSS selectors are like little maps for Scrapy to navigate the DM tree and find the location of the data we need. Here, we are going to use XPath to parse the data from the page “<https://www.reed.co.uk/jobs/data-analyst-jobs>”. We will use Scapy Shell to create these XPath and it comes with a built-in shell that allows quickly test and debug XPath and CSS selectors. Without having to run full scraper to see if your Xpath or CSS selector are correct, you can enter them directly into terminal and see the result.

Install IPython as you scrapy shell which provides smart auto-completion and colorized output.

*pip install ipython*

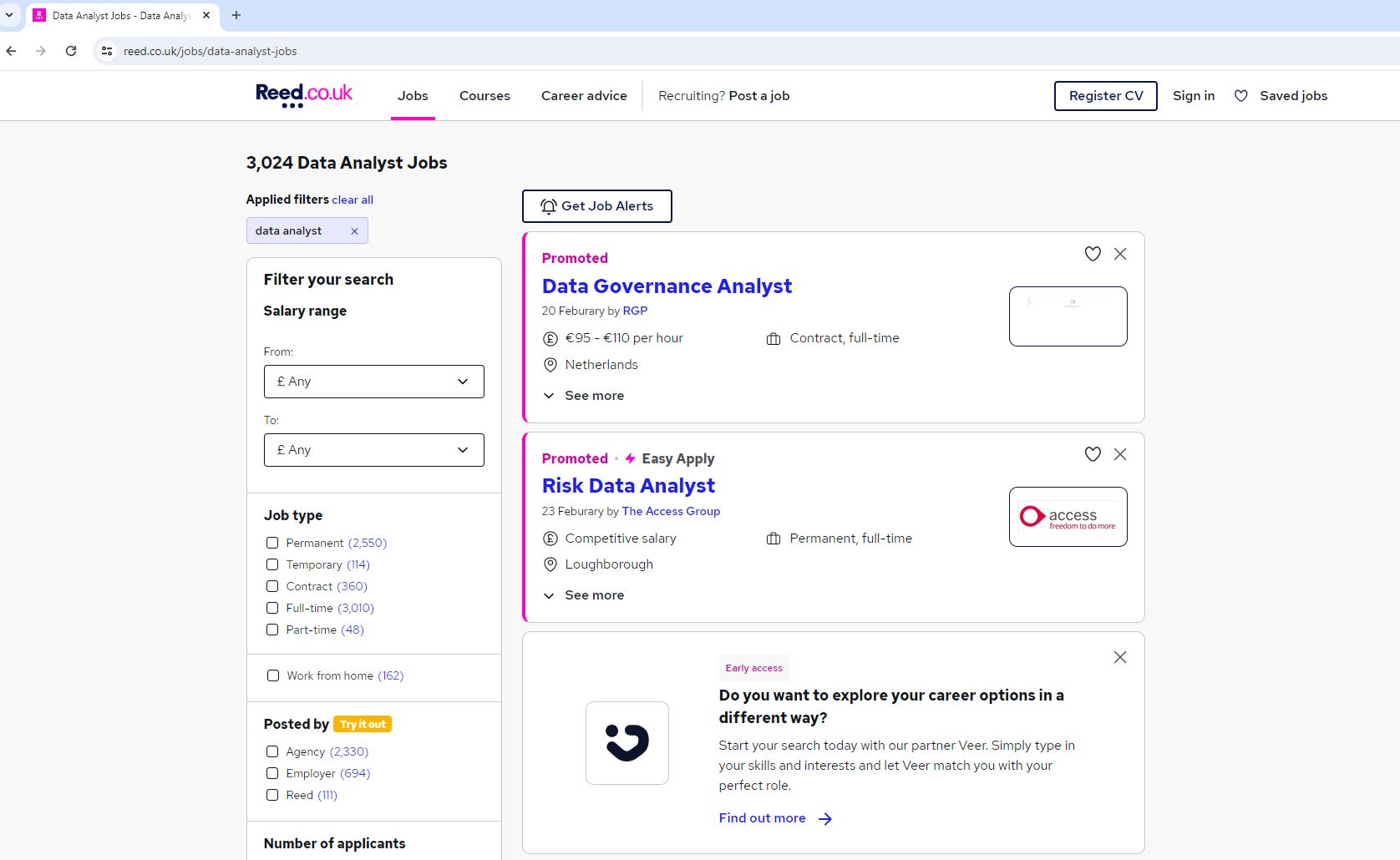
Then after, open scrapy shell.

*scrapy shell*

**Fetch the Page**

To create our XPath we will be testing them on the given web page.

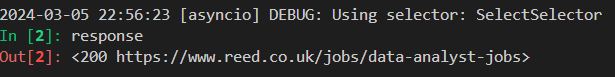
“<https://www.reed.co.uk/jobs/data-analyst-jobs>”.



**Fetch the page by putting this url in scrapy shell.**

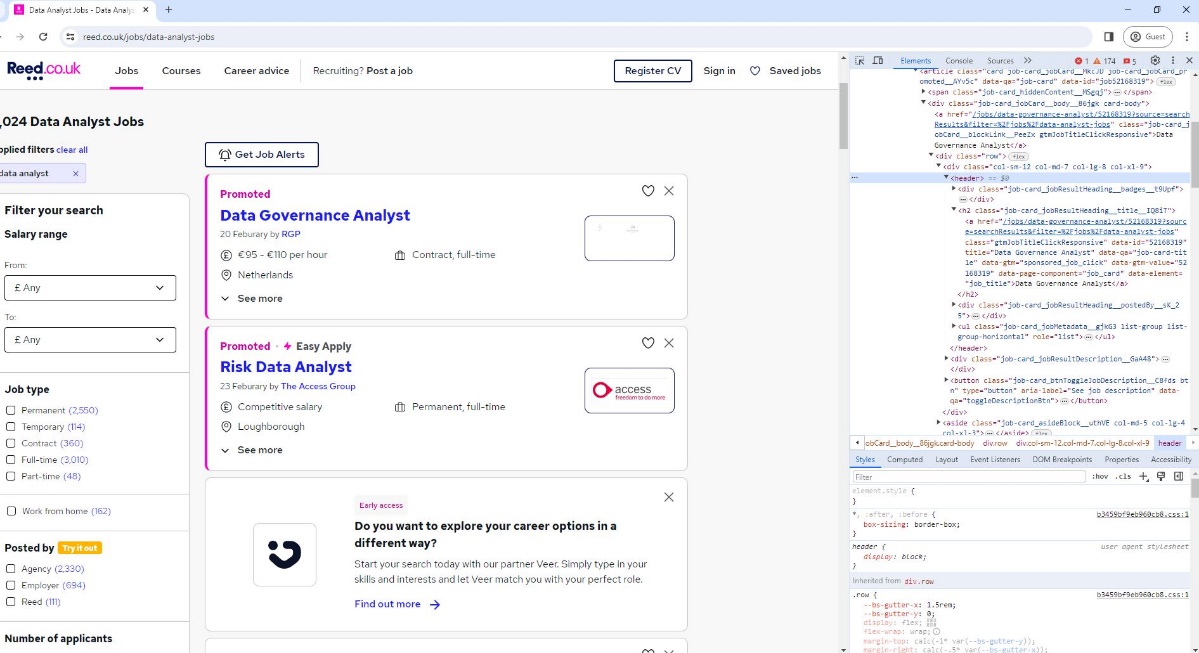
*fetch('*[*https://www.reed.co.uk/jobs/data-analyst-jobs*](https://www.reed.co.uk/jobs/data-analyst-jobs)*’)*

After successful retrieve the page, Scrapy shell has automatically saved the HTMl response in the response variable.



**Find Job XPath**

To find the correct XPath to parse the job details, open the website and then right click on the page and click inspect.

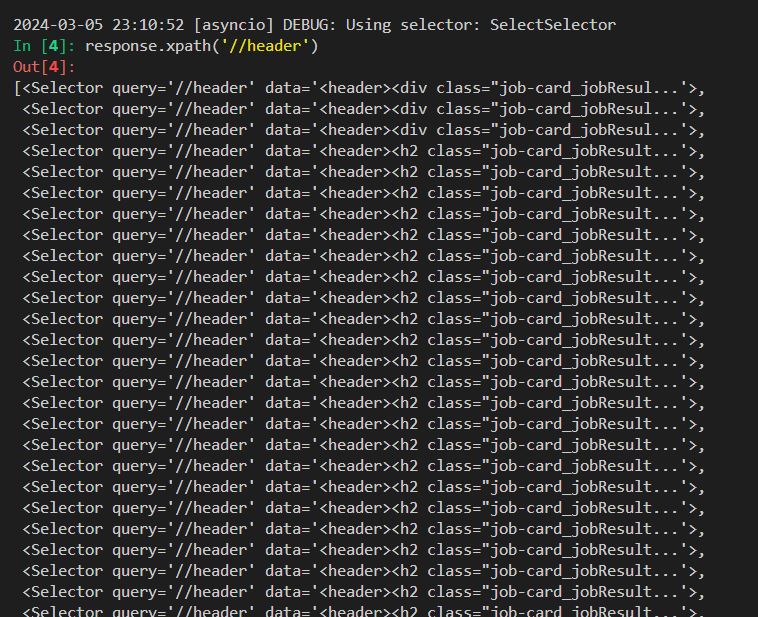


Using the inspect element, hover over the item and look at the tags, id’s and classes on the individual job lists.

In this case, we can see that each job detail has its own tag called header.

Now, using our Scrapy shell we can see if we can extract the product information using this tag.

*response.xpath(‘//header’)*

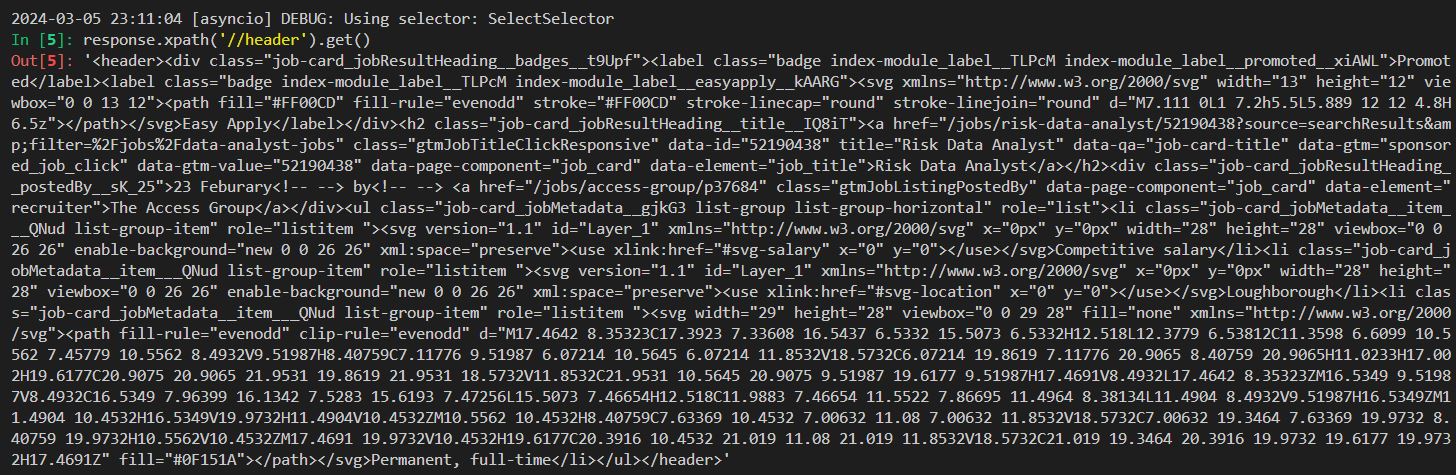


It has found all the elements that match this xpath.

**Get First Job detail**

To get the first job detail use .get()

*response.xpath(‘//header’).get()*



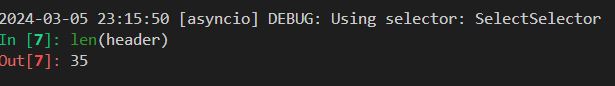
**Get All Job Detail**

Save the data into a response variable and loop through the items and extract the date that need.

*header = response.xpath(‘//header’)*

To check the length of header variable to know how many jobs are there.

*len(header)*



**Extract Job Detail**

Now, extract the field url\_detail, title, salary, contract\_type, job\_type and location of ech job from list of jobs.

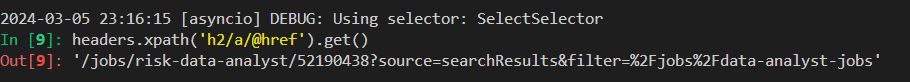
Test xpath on the first element of the list

Get single product

*headers = header[0]*

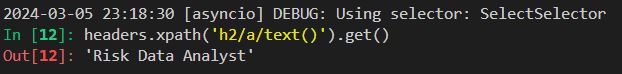
url\_detail – the job url can be found with:

*headers.xpath(‘h2/a/@href’).get()*



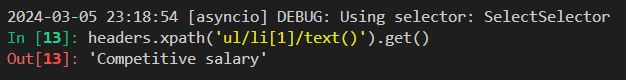
**Title** – the job title can be found with:

*headers.xpath(‘h2/a/text()’).get()*



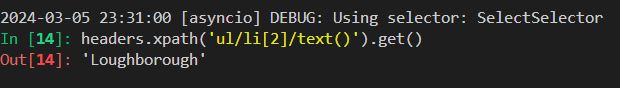
**Salary** – the job salary can be found with:

*headers.xpath(‘ul/li[1]/text()’).get()*



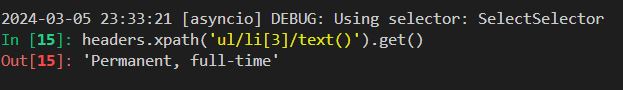
**Location** – the job location can be found with:

*headers.xpath(‘ul/li[2]/text()’).get()*



**Contract and Job type** – the job contract and job type can be found with:

*headers.xpath(‘ul/li[3]/text()’).get()*



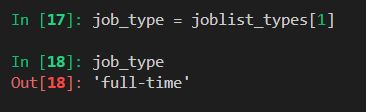
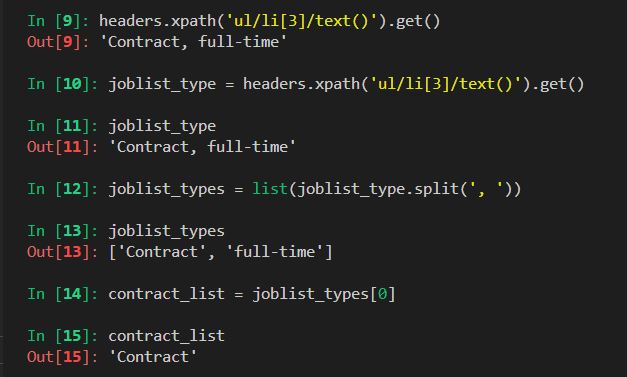
Contract and Job type are in same string but we need it separately. Here we have separate this value. To do so, first make it list by using list() function and split it by using split function then, we get a list having separate value.

*joblist\_type = headers.xpath('ul/li[3]/text()').get()*

*joblist\_types = list(joblist\_type.split(', '))*

*contract\_list = joblist\_types[0]*

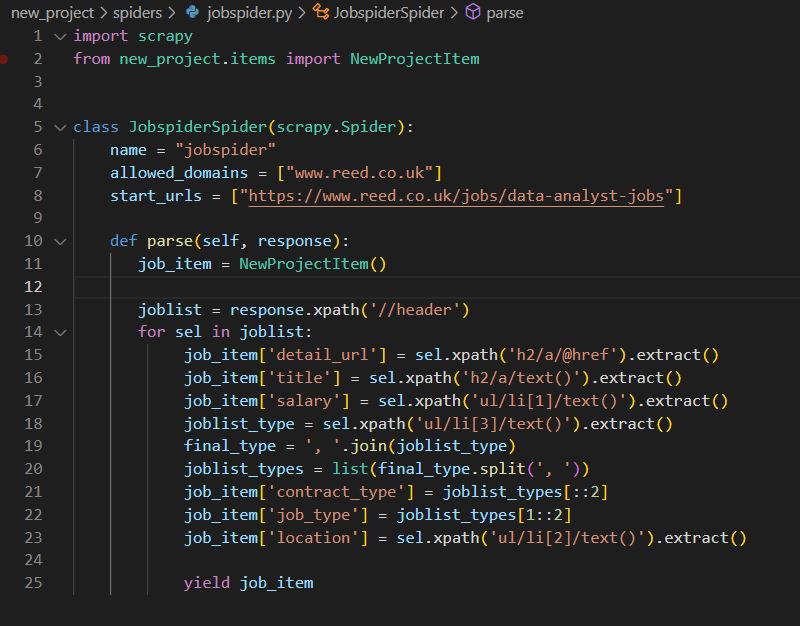
*job\_type = joblist\_types[1]*



**Update Spider**

Now, exit scrapy shell with the exit() command. Then, update the spider with xpath

Update spider code



Here, spider does the following steps:

1. Makes a request to “<https://www.reed.co.uk/jobs/data-analyst-jobs>”.
2. When it gets a response, it extracts all the products from the page using

*joblist = response.xpath(‘//header’)*

1. Loops through each sel and extracts the detail\_url, title, salary, contract\_type, job\_type and location using xpath we created.
2. Here, for contract\_type and job\_type. First, we join a list using join() function which concatenation a list of string with ‘, ‘ to form a string. After that , convert that string into list using list() function, split using split function with ‘, ’ and assign into variable called joblist\_types.
3. After getting joblist\_types list, separate the value inside the list to get contract\_type and job\_type by using the logic of odd even in index of list. The odd index of list will be contract\_type and even index of list will be job\_type

*job\_item[‘contract\_type’] = joblist\_types[::2]*

*job\_item[‘job\_type’] = joblist\_types[1::2]*

1. Yields these items so they can be stored in a CVS, JSON etc.

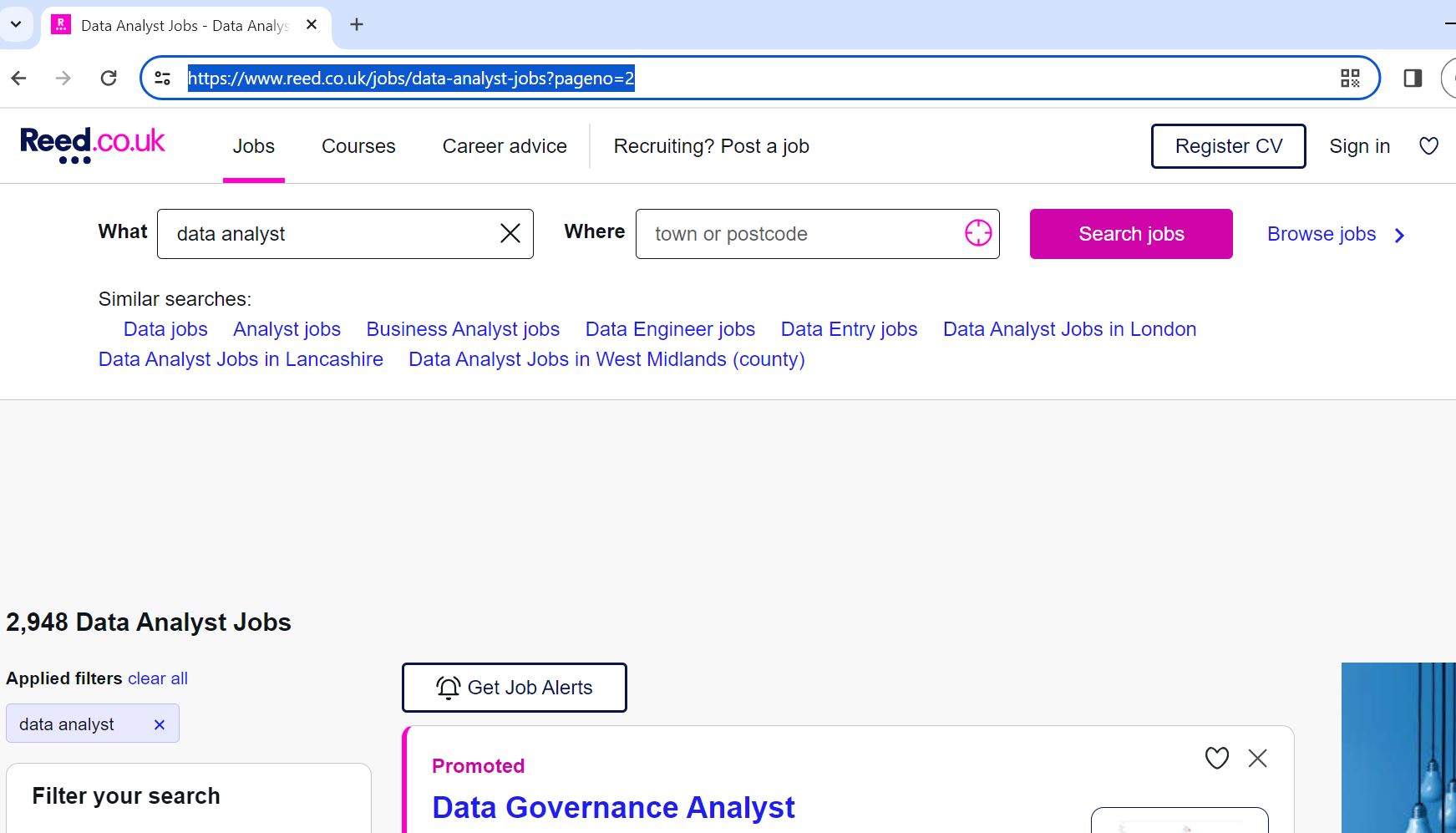
**Extracting data for next page**

To extract the same format of data from next page, we have to update our spider with some new code.

Define a variable page\_number assigning a value 2 inside spider class

*page\_number = 2*

Then, go inside the parse function, and define next\_page variable having a value website url of next page and page\_number.

To get the url of next page. Go to page number 2 and copy the url :

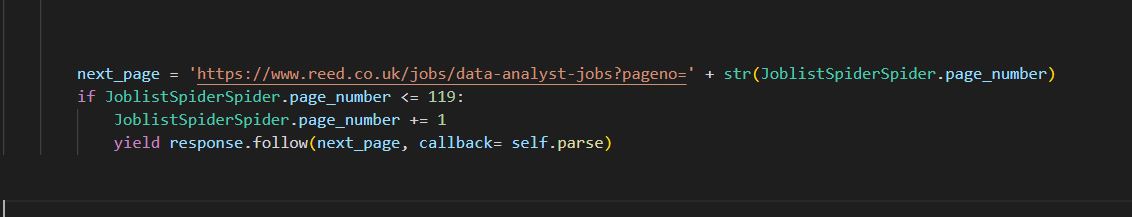
And remove the integer value from url and your final url be like:

[‘https://www.reed.co.uk/jobs/data-analyst-jobs](https://www.reed.co.uk/jobs/data-analyst-jobs)?pageno=’

Just, concate the url and page\_number. We cannot concate string and integer so, convert the integer value into string by using str() function. Then, concate the url and page number by using operator ‘+’.

*Next\_page = ‘*[*https://www.reed.co.uk/jobs/data-analyst-jobs*](https://www.reed.co.uk/jobs/data-analyst-jobs)*?pageno=’ + str(JobspiderSider.page\_number)*

Here, we have 119 pages of job list. We have to scrape all job list from this pages. To do so, we will define if condition that if page\_number is less than or equal to 119 then page\_number should increment by 1 and go to the next page and scrape the data by using parse function.



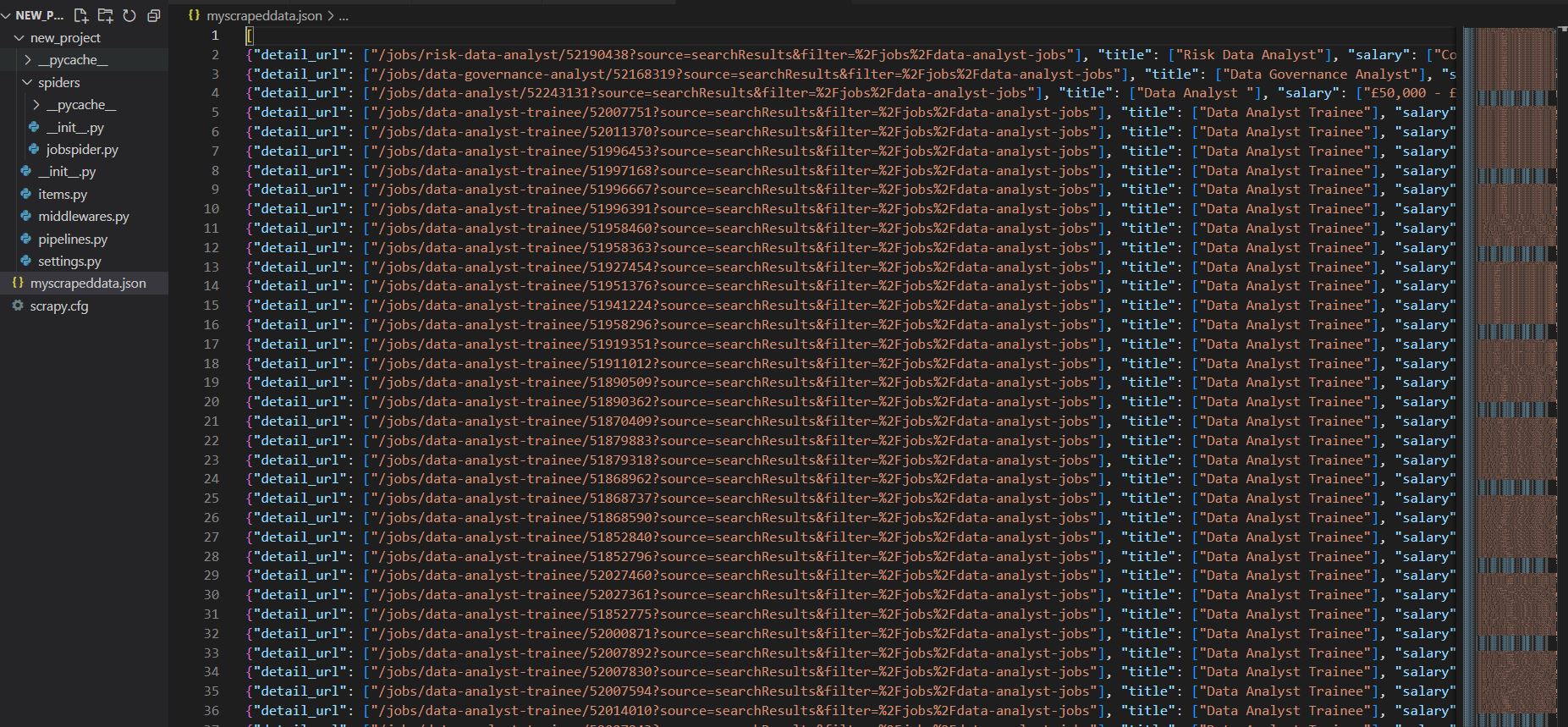
After that run the project.

# scrapy crawl spidername

*scrapy crawl jobspider*

Now, you will get job list detail of every pages until its 119. To save this data to a JSON file we can use the –O option, followed by the name of the file.

*scrapy crawl jobspider -O myscrapeddata.json*



For CSV file

scrapy crawl jobspider -O myscrapeddata.csv

