**Scraping and Analytics**

**Workshop**

**Background**

Educational workshop ON SCRAPING AND ANALYTICS APPLIED TO SMARTCITIES. This work involves a multi-university team of students, Partnering and Faculty.

**Software tools**

* Scrapy, a free and open-source Python library (module) created for web scraping. <https://scrapy.org>
* Anaconda, a free and open-source distribution of Python (and R) which simplify package management. <https://anaconda.org>
* Google Chrome, web browser developed by Google, based on the open source web browser called Chromium. <https://www.google.com/chrome>

**Accessible Tutorial sources**

The below sites have many hands-on examples of web scraping using Python library Scrapy, for your awareness.

* <https://doc.scrapy.org/en/latest/intro/tutorial.html>
* <https://www.tutorialspoint.com/scrapy/>
* <https://www.digitalocean.com/community/tutorials/how-to-crawl-a-web-page-with-scrapy-and-python-3>
* <https://www.analyticsvidhya.com/blog/2017/07/web-scraping-in-python-using-scrapy/>

**Tool introduction**

Scrapy is a Python 3 library (module) created as framework to extract web page content in large scale. In order to simplify its Python environment and library implementation, it is used in this workshop the Anaconda Python platform which provides an installation tool to enable easy installation and clean installation.

Anaconda orientation resource:

* Anaconda used in this workshop has the Python version 3.7. Any earlier version should be suitable for this workshop. Additional information about Anaconda can be found in the following link:
  + <http://bit.ly/AnacondaTut2>
* It is important the basic knowledge of Python 3, as Scrapy is a library (module) of this language. In case you have no relevant knowledge of Python is highly recommended to study its foundation. Several basic Python tutorials are available in the internet, but the following link you can access to a recommended video training.
  + <http://bit.ly/Python4B>
* The Python version 2 is not considered this workshop as the end of life (EOL) is announced for 2020. In case you have knowledge of the old version it is recommended to watch an overview of the main difference between Python version 2 and 3.
  + <http://bit.ly/Python2and3>

**Content**

**Part 1: Workshop Preparation …………………………………… 5**

**Part 2: Scraping Introduction ……………………………………. 19**

**Part 3: Scraping …………………………………………………… 23**

**Part 4: Data Analysis ……………………………………………… 32**

**Part 5: Assignment .……….……………………………………….. 36**

**Part 1: Workshop Preparation**

**1. Definitions**

Follow the definition of the main tools necessary for web site scrape and analytics.

**Web scraping**

Web scraping, aka web harvesting, is the technique used for extracting data from websites. Web scraping software can access web sites using the http protocol, or a web browser. Typically is used to capture data from web sites and stored into a central local database for later analysis.[[1]](#footnote-1)

**Scrapy**

Scrapy is a free and open-source framework based in Python to scraping web pages. This frame work is also capable to extract data using APIs. It is maintained by Scrapinghub Ltd., a web-scraping development and services company.[[2]](#footnote-2)

**Python Definition**

Python is high-level intepretable programming language. First released in 1991 by its creator Guido van Rossum, Python focused on code readability enabling the programmers to build a clean source code.[[3]](#footnote-3)

**Anaconda Definition**

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing including Data Science activities, Data Mining and Text Mining with a easy to use control of library and isolated environments.[[4]](#footnote-4)

**Source Code Editor Definition**

A source code editor is a text editor software used for editing source code of computer programs. Can be supplied as a single software or in an IDE. Source code editors is the mandatory for programming.[[5]](#footnote-5)

**Jupyter Notebook Definition[[6]](#footnote-6)**

Jupyter Notebook is a web-based interactive computational environment for creating Jupyter notebooks documents based on R or Python programming languages. The Jupyter Notebook document is strucutered as a JSON document. The document is formed by individual cells which can execute its programming content independently of the other cells.

**2. Installations.**

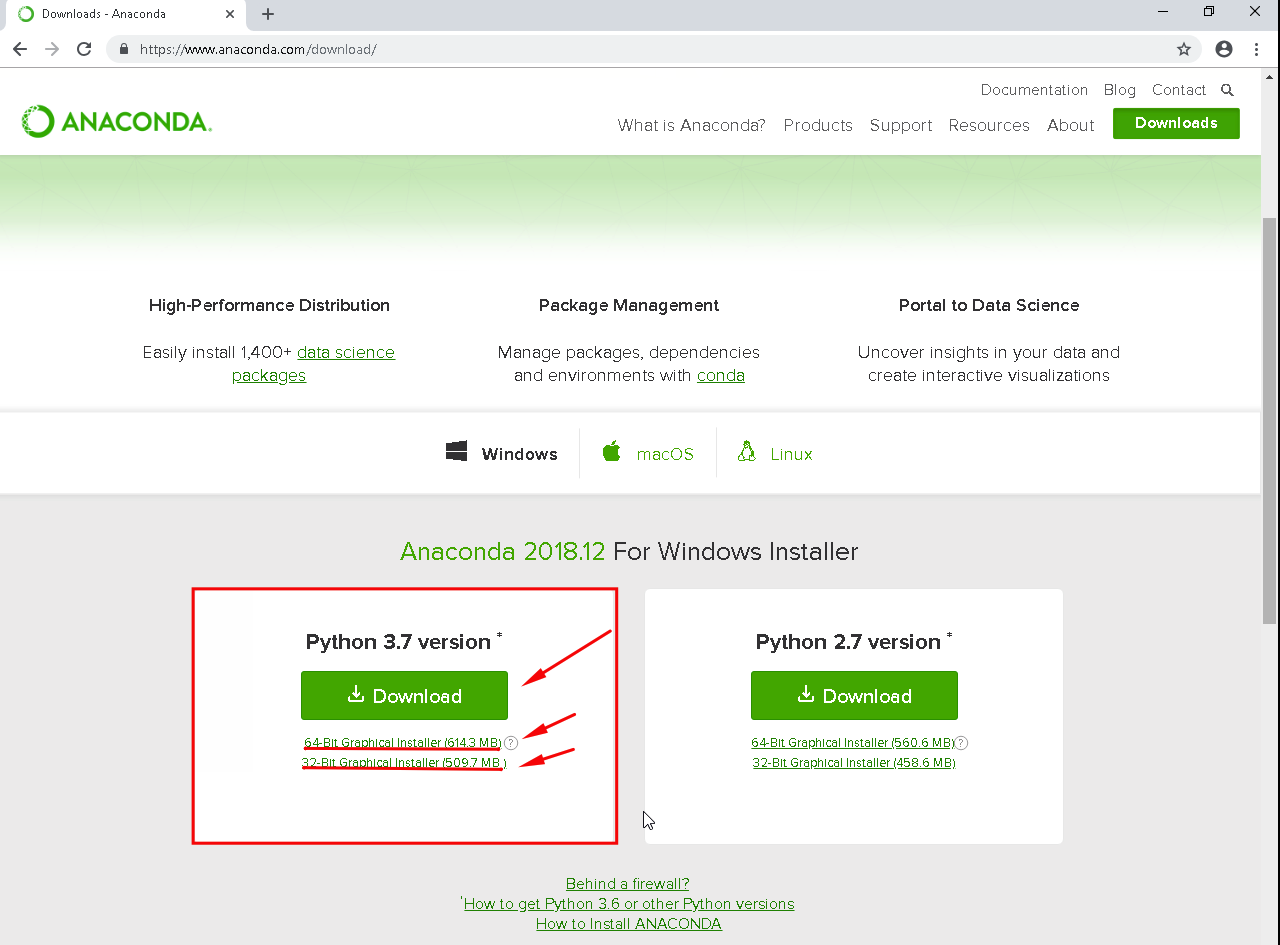
**Python environment installation**

As mentioned before the Python environment will be deployed with the support of Anaconda software. Its installation is described in the following procedure for Windows 10 OS which is the majority of cases. If necessary, the installation on Linux or macOS, please refer to the Anaconda web page supplied in this document.

**Anaconda Installation in Windows 10**

The installation package can be obtained in the following link:

<https://www.anaconda.com/download/>



The main green button is related to Python 3.7, 64 bits version. Below the green button you can find two options available: Windows 64bits and 32 bits. Select the suitable link according to your operation system.

|  |  |
| --- | --- |
| **Step 1** | **Step 2** |
| **Step 3** | **Step 4** |
| Before click “next” make a note of the Destination Folder indicated in the pop-up windows. This will be used to configure your Windows after finish the installation. In the case above the Destination folder was:  C:\Users\”My user name”\Anaconda3 | |

|  |  |
| --- | --- |
| **Step 5** | **Step 6** |
| **Step 7** | **Step 8**    In this step you have the option to install the source code editor VSCode. This is an open-source, free software from Microsoft, which can be used for the purpose of this workshop. In case you don’t have any IDE installed it is recommended to install this application for simplicity. |
|  |  |

|  |  |
| --- | --- |
| **Step 9** |  |

**Windows PATH**

It is necessary to configure the Windows Variable Environment with the path of Anaconda. In the Cortana window left below in the desktop write “environment variables”. The Windows will give a series of suggestions, please select the first one.

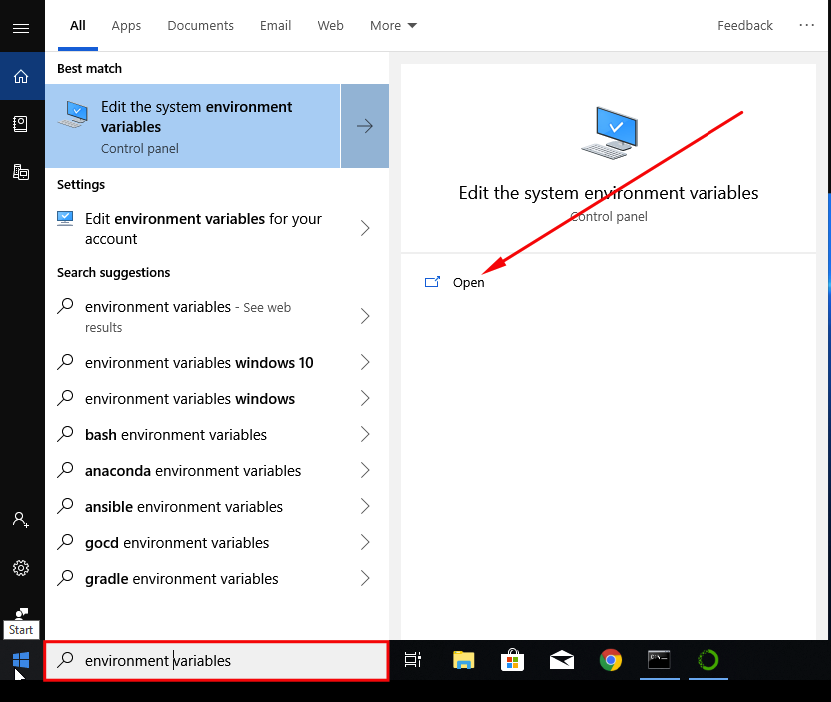


Figure 1-Windows Cortana Search.

|  |
| --- |
| Figure 2-System property. |

|  |  |
| --- | --- |
| **Setting Anaconda application path**  In the next window will be select first “New” button, then write the “Folder Destination” information noted during the installation process of Anaconda, adding the folder name “Scripts”. This will be enough to add Anaconda path for commands in the Terminal.  After this close all the windows by clicking the option “OK”. | Figure 3-Edit Environment Variable. |
| In the new pop-up window select first the “Path” indicated and then click the button “Edit”. | Figure 4-Environment variable. |

**Installation Verification**

After the installation of Anaconda, it is necessary to test if the installation was well succeeded[[7]](#footnote-7) by listing the packages installed in the OS. Using the Terminal run the command:

conda list

If the result is a list of libraries it is sufficient sign to indicate a successful installation.

**Preparing an Environment for Scrapy**

It is considered part of good practices the creation of a specific Anaconda environment to execute Scrapy. One of advantages is the prevention of Python and library version conflict in your machine. In order to create the environment, it is necessary to execute the Anaconda Navigador and select the option “environment” on the left side of interface as depicted in the figure 5.

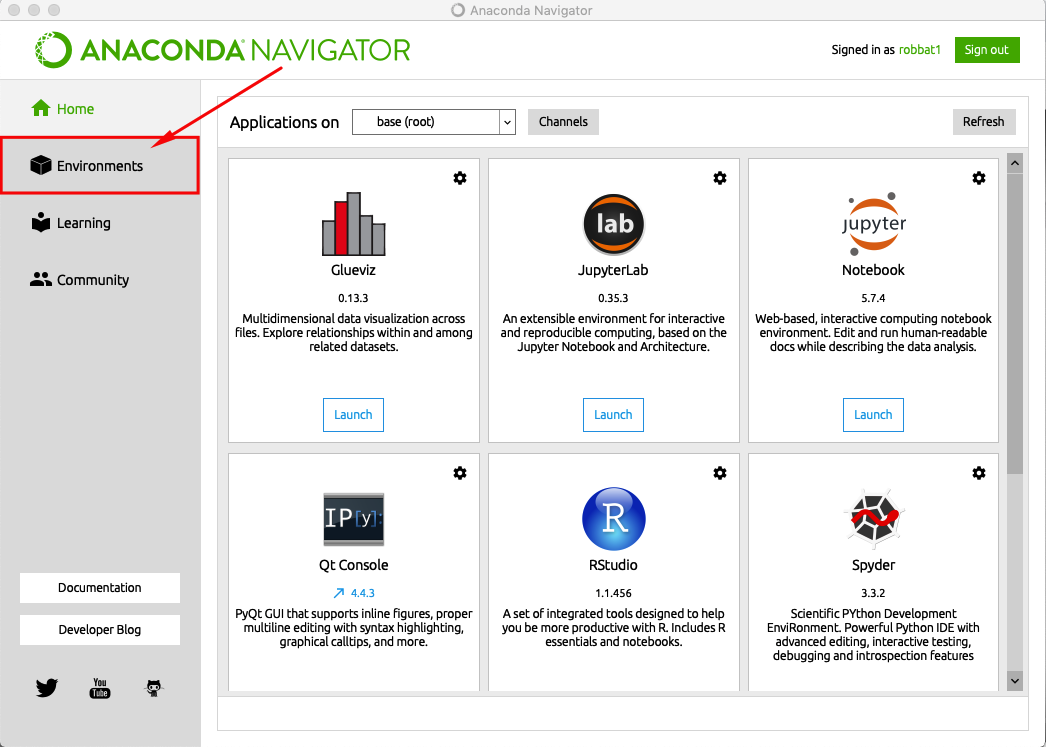
****

Figure 5-Anaconda navigator.

After select the button “environments” select the button “create“, as indicated in the point “1” of the figure 6.



A pop-up box will present the items to be filed:

Point 2: Add a name for the new environment. In this example was chosen “Scrapy”.

Point 3: Select the Python version for the environment. For this workshop we are using Python 3.7.

Point 4: Click on “Create”.

After follow the four points, Anaconda will start to create the environment and in the end will list it below the base (root) environment.

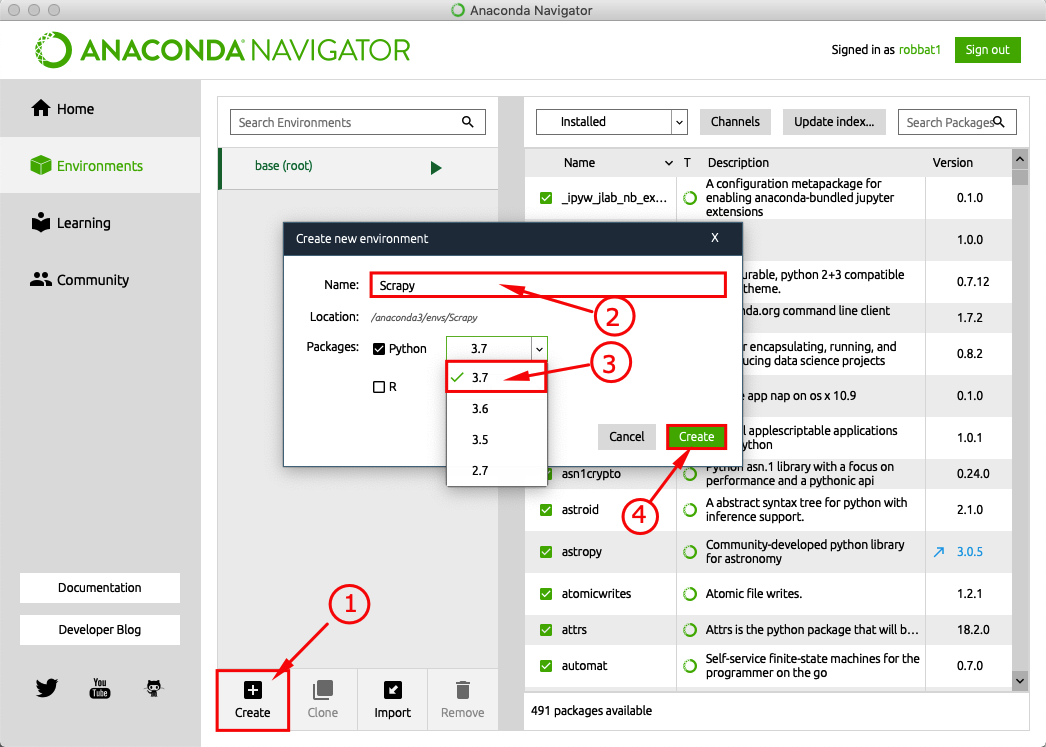
****

Figure 6-Environment creation.

**Scrapy Installation**

To install Scrapy using Anaconda it is easy. Following the points in the figure 7 you should select “All” and write the package name to be installed, in this case “scrapy”. Immediately the Scrapy package will be listed in the window (figure 8).

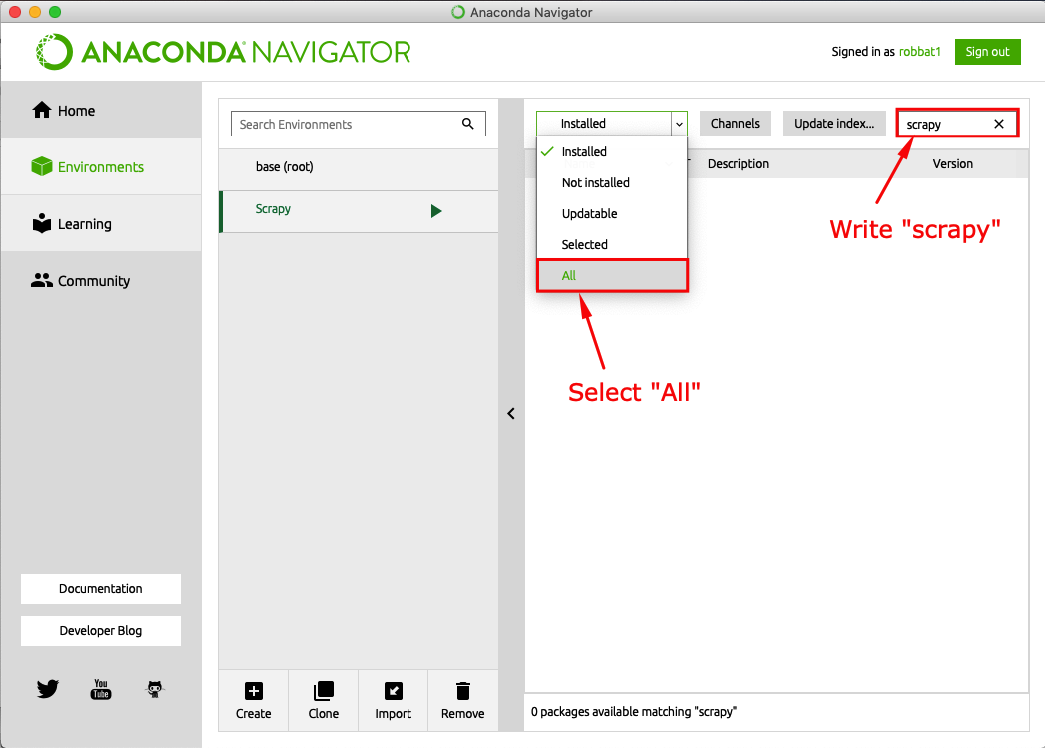


Figure 7-Scrapy installation using Anaconda.

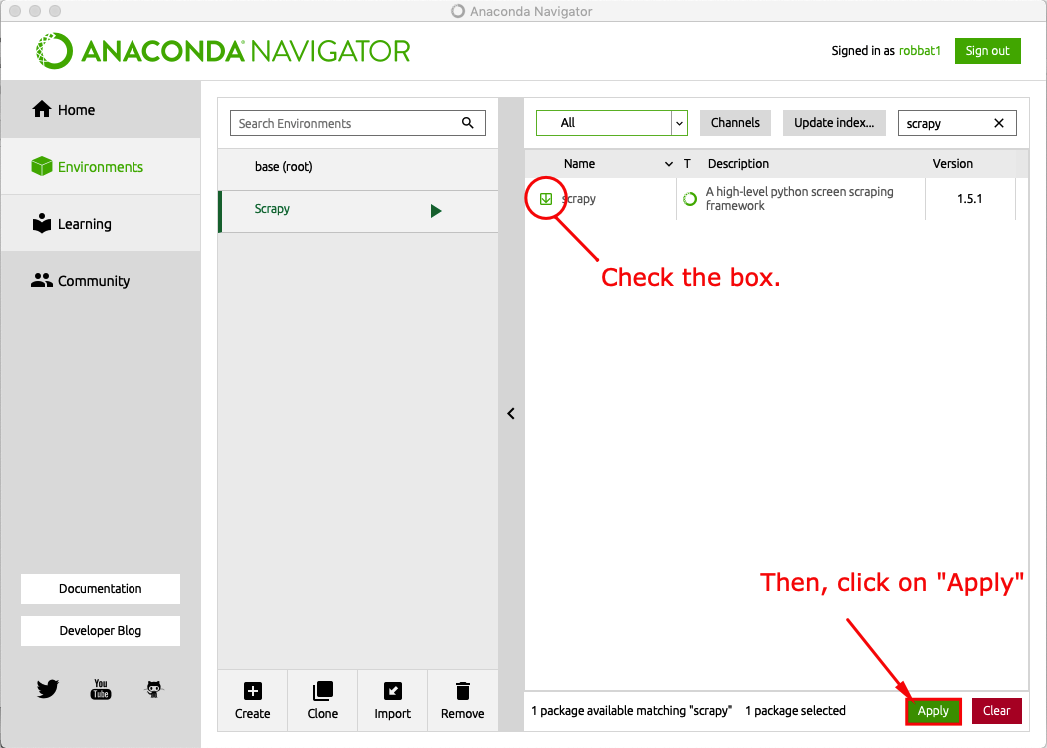


Figure 8-Installing Scrapy.

|  |  |
| --- | --- |
| Figure 9-Solving package specifications.  Anaconda will search for all the packages which needs to be installed to support Scrapy module. | Figure 10-Package dependences list.  After list all the dependences just click on “Apply”. |

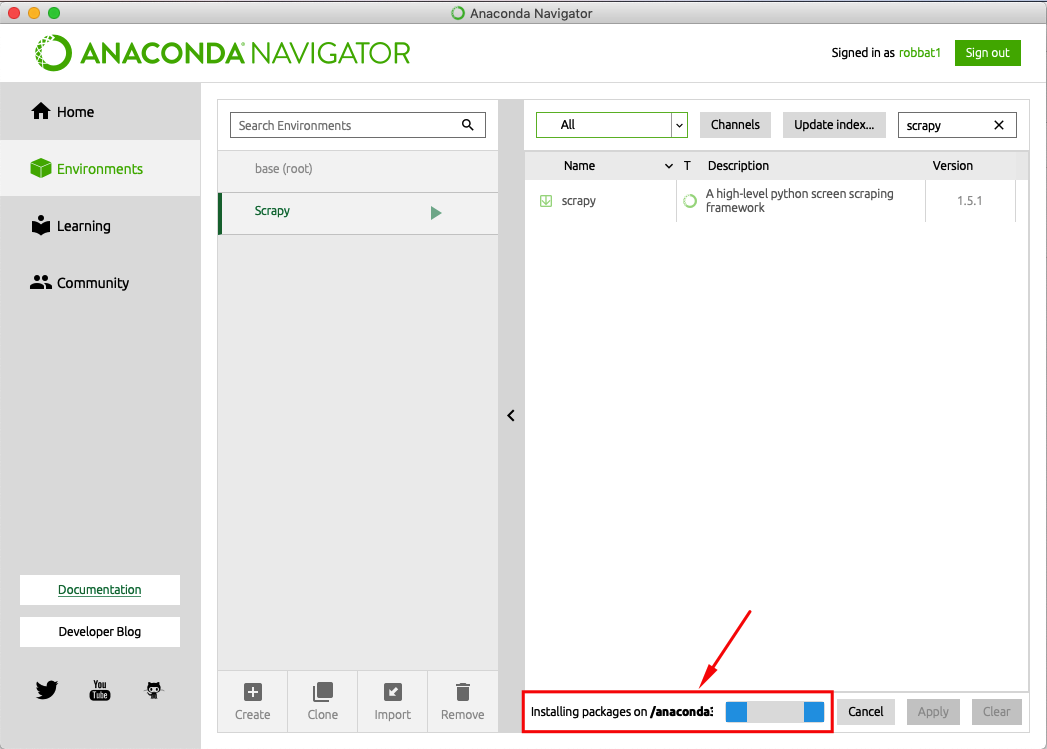


Figure 11-Installing packages from dependences list.

Anaconda will be installing all the packages which can take few minutes to accomplish this task (figure 11).

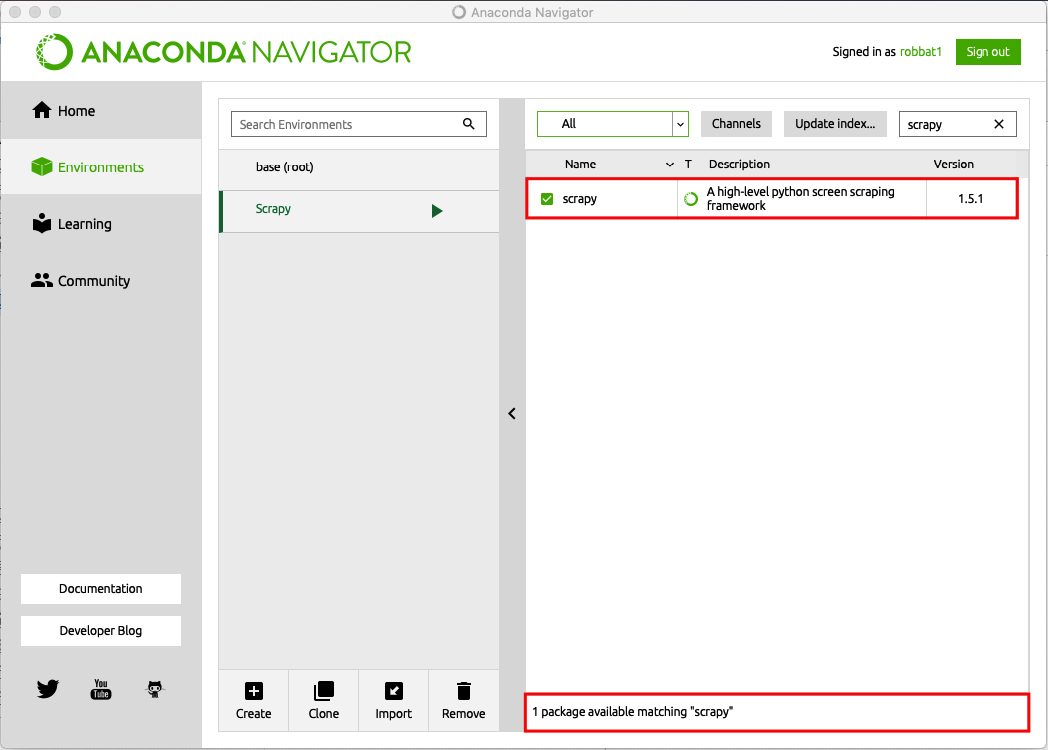


Figure 12-Scrapy successful installed.

Final Anaconda will indicate that Scrapy was successful installed (figure 12).

**Anaconda installation using command-line interface**

It is possible to install Scrapy using Anaconda command-line Interface. Using the Windows Terminal, you are able to install Scrapy package executing the following commands:

conda install -c conda-forge scrapy

**Jupiter Notebook installation**

The Jupyter notebook software will be used for Data Analytics after the scrape process delivers the csv file. This software is part of Anaconda Navigator and can be easily installed with the following steps.

To initiate the installation, it is important to certify which the environment selected in the Anaconda Navigator is “Scrapy”, go to the Anaconda Navigator and select the option “install”.

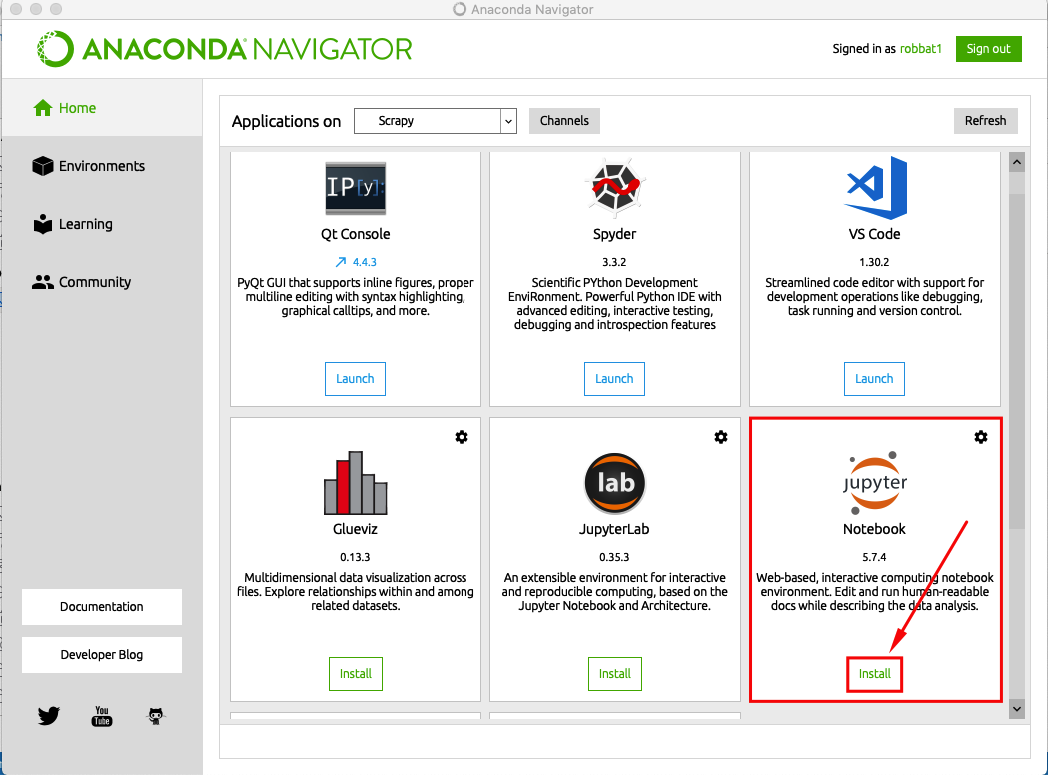


Figure 13-Anaconda Navigator and Jupyter Notebook Installation.

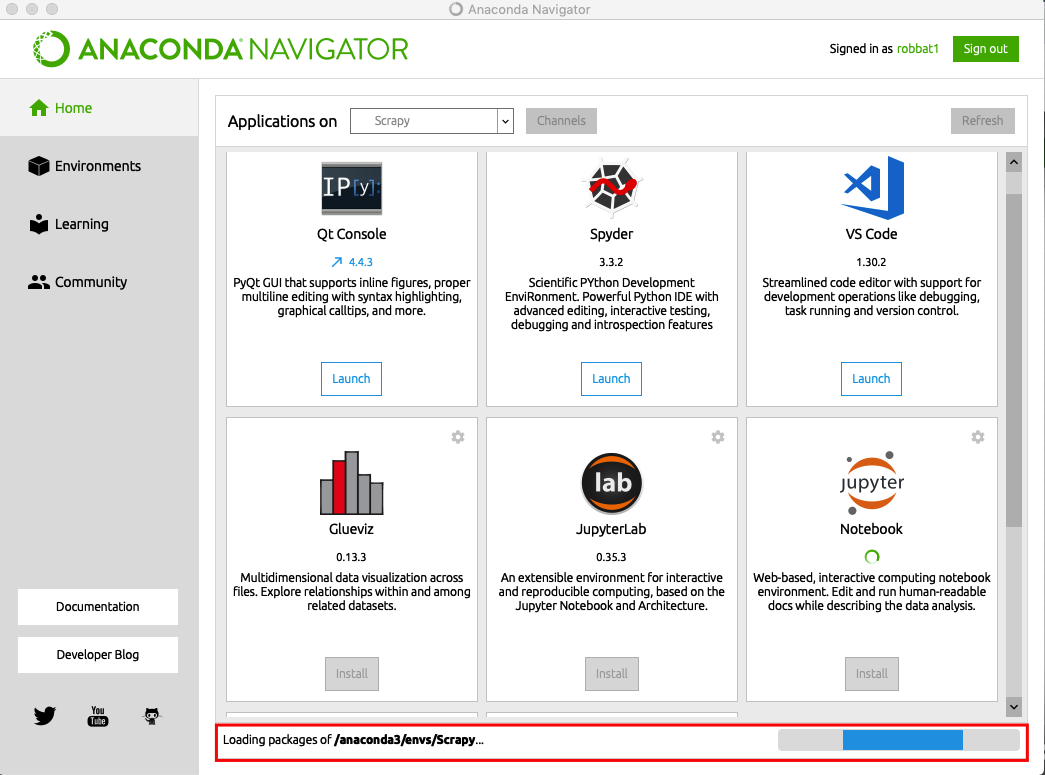


Figure 14-Anaconda installs Jupyter Notebook.

**Part 2: Scraping Introduction**

**1. Initial Considerations**

Before we start to use Scrapy it is necessary to know certain fundamental concepts of web scraping and of Scrapy itself, which will help you to succeed in this important tool for data analysis.

**Ethics**

The objective of this workshop is educational, but it is necessary to point that each web page has a different crawling policy which is necessary to understand and respect. This information is contained in the “robots.txt” file available in the web site root directory, and in some cases in the legal material such like “terms and conditions”. The irresponsible scraping of page can violate copyrights and perform as like a Denial-of-Service (DoS) hacker attack. Always verify the legal and technical web site details of the web site you intend to scrape.

Some web pages made available an API to access their content, this method is preferable instead of scraping. For educational purposes we will use [www.ebay.com](http://www.ebay.com) web page with a specific search configuration and specific group of words related to equipment used for health purposes - “heart rate monitor.” This equipment is subject to smart cities wellness studies.

**Data Target Element Location**

The web scraping has the purpose to extract specific data content from an Internet web page which can be located in different areas of the page. In order to point this specific data for extraction Scrapy uses XPath and CSS expressions. A web page can have its source code with hundreds of thousands of HTML and CSS nodes, demanding a tool to identify the location of a specific element of interest in the web page. The tool chosen to be used in this workshop is the Inspector from Chrome Web Browser. Below has an example of use.

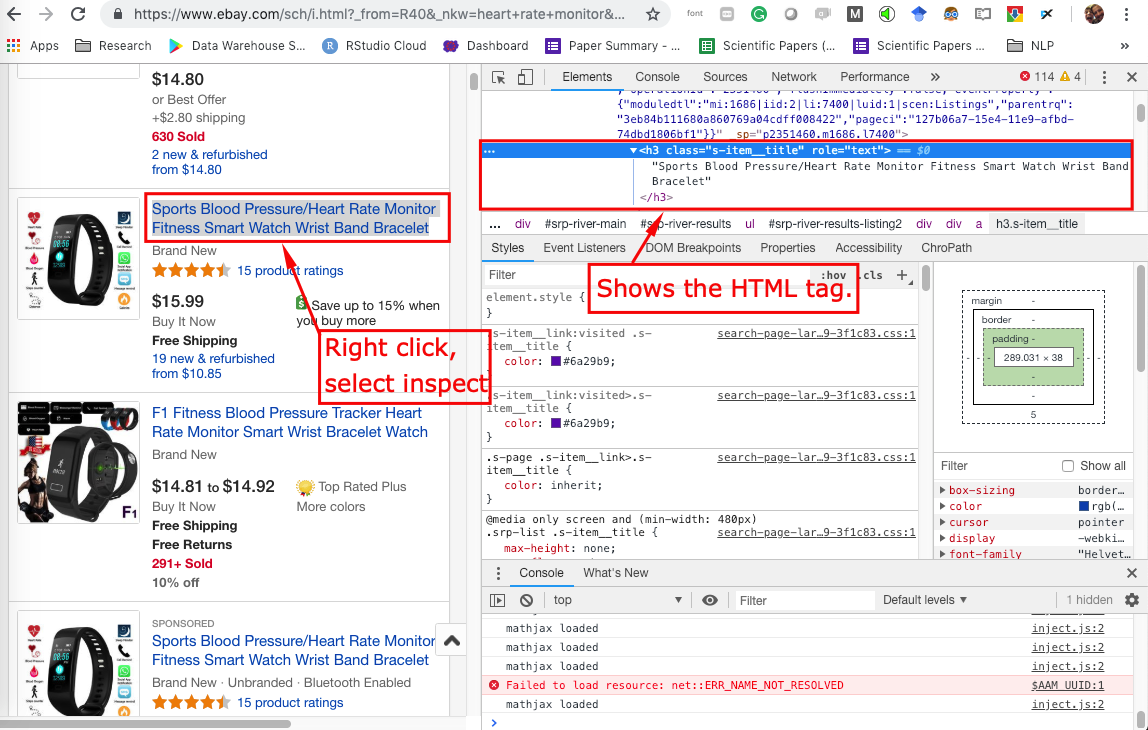


Figure 15-Chrome web browser Inspector.

The inspector can give us the Xpath and CSS expressions to reach specific data in the HTML page. This functionality can be used by selecting the element in the web page, right-click, and select in the list the option “Inspect”. A window will be opened with a series of tabs. The figure 17 shows an example of use. The information needed is the XPath or CSS element to be used in the Spider.

An important consideration is that some pages - specially pages with search option - uses dynamic techniques which make the web page HTML and CSS nodes changes depending on the option of search you use. For instance, in eBay if you select specific price range, features, brands, among other options, the resulted web page will provide can contain different HTML and CSS nodes if compared with search with same terms.

**XPath and CSS**

XPath is a query language used to select nodes inside of XML document and HTML, and it is used to navigate inside of the web page contents. CSS is a syntax to for applying styles to HTML documents. Scrapy uses XPath expressions to make selectors for the parts of interest in the scraping process.

To know more please visit the web site below indicated:

https://www.w3schools.com/xml/xml\_xpath.asp

**2. Scrapy Core Components**

The figure below depicted the Scrapy elements architecture and the flow of request and response. The basic elements are:

Engine: The engine is responsible for controlling the data flow, and triggering events based on actions.

Spider: Spiders is the element responsible to crawl web sites and extract data according to the selectors built-in.

Downloader: Downloaders is a middleware that perform HTML requests and responses.

Scheduler: Scheduler is a schedule mechanism responsible to process requests enqueued.

Items Pipeline: Item pipeline is a Python class that implements a simple method which is executed sequentially to process the URL response.

Items: Item objects are simple containers used to collect the scraped data.

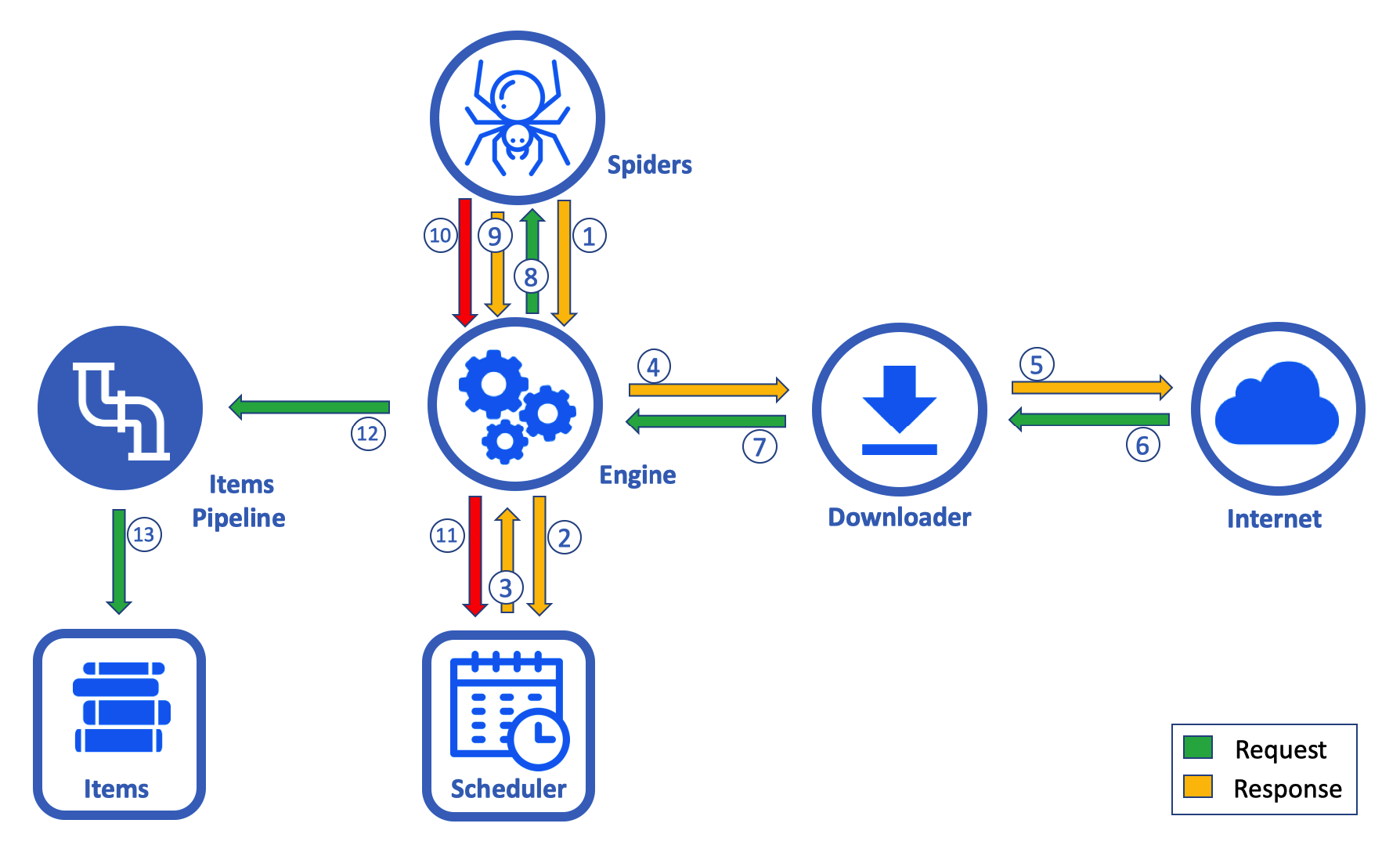


Figure 16-Scrapy architecture.[[8]](#footnote-8)

**Data Flow**

(1) The Spider sends initial crawl request to the Engine.

(2) (3) The Engine sends the request to the Scheduler which queue it and return it to Engine following the schedule.

(4) Engine forward the Request to the Downloader.

(5) The download makes a HTTP request to the web site server.

(6) The Response goes through Downloader, to the Engine (7), then pass to the Spider (8) which scrap the response.

The Spider (9) returns the scraped content, and a new request is submitted to the Engine (10), which forward to the Scheduler to queue it (11).

(12) (13) The Spider Response Scraped is forward to Items Pipeline, which process the item.

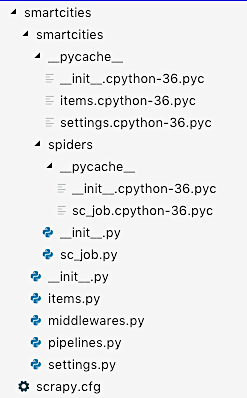


Figure 17-Scrapy Project default directory structure.

Scrapy is framework software controlled using a command-line tool called scrapy, and its subcommands. Below you can see syntax of scrapy commands supplied by typing the scrapy command without arguments in the terminal:

*Usage:*

*scrapy <command> (options) (args)*

*Available commands:*

*bench Run quick benchmark test*

*check Check spider contracts*

*crawl Run a spider*

*edit Edit spider*

*fetch Fetch a URL using the Scrapy downloader*

*genspider Generate new spider using pre-defined templates*

*list List available spiders*

*parse Parse URL (using its spider) and print the results*

*runspider Run a self-contained spider (without creating a project)*

*settings Get settings values*

*shell Interactive scraping console*

*startproject Create new project*

*version Print Scrapy version*

*view Open URL in browser, as seen by Scrapy*

*Use "scrapy <command> -h" to see more info about a command*

In this workshop we will use some of these commands.

**Part 3: Scraping**

**1. Investigation Steps**

It is necessary to investigate the target page in order to familiarize with it and make initial tests related to the download, data selection and, extraction. The tests proposed will be done using the interactive environment called Scrapy Shell which is specifically design for this purpose.

The Scrapy Shell can try XPath or CSS expressions presenting the results with no need to configure a spider for it. We will initiate using the command below:

**$ scrapy shell**

After execute the command above the final data showed in the terminal is the available Scrapy objects in the shell environment:

*[s] Available Scrapy objects:*

*[s] scrapy scrapy module (contains scrapy.Request, scrapy.Selector, etc)*

*[s] crawler <scrapy.crawler.Crawler object at 0x1092933c8>*

*[s] item {}*

*[s] settings <scrapy.settings.Settings object at 0x10b825898>*

*[s] Useful shortcuts:*

*[s] fetch(url[, redirect=True]) Fetch URL and update local objects (by default, redirects are followed)*

*[s] fetch(req) Fetch a scrapy.Request and update local objects*

*[s] shelp() Shell help (print this help)*

*[s] view(response) View response in a browser*

**Step One - Fetching the page**

Once on the Scrapy Shell “>>>” we will use the ‘fetch’ command (short cut) to fetch the target web site. Which in this case will be the following search from [www.ebay.com](http://www.ebay.com).

**'https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=heart+rate+monitor&\_sacat=0&LH\_ItemCondition=3&rt=nc&LH\_BIN=1&\_ipg=200'**

**>>>** **fetch("https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=heart+rate+monitor&\_sacat=0&LH\_ItemCondition=3&rt=nc&LH\_BIN=1&\_ipg=200")**

*2019-01-14 07:56:13 [scrapy.core.engine] INFO: Spider opened*

*2019-01-14 07:56:15 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=heart+rate+monitor&\_sacat=0&LH\_ItemCondition=3&rt=nc&LH\_BIN=1&\_ipg=200> (referer: None)*

The response from Scrapy indicates in the DEBUG the information “Crawled (200)”, which means that the web page crawled was performed successfully. After the page be crawled, we are ready to start our initial tests.

The first test to be done is the confirmation that the ‘fetch’ was done in the correct URL:

**>>> response.url**

*'https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=heart+rate+monitor&\_sacat=0&LH\_ItemCondition=3&rt=nc&LH\_BIN=1&\_ipg=200'*

We could identify that the URL used by fetch was correct. The next verification is to retrieve the web page content.

**>>> view(response)**

*True*

After the command above be executed your web browser should open automatically and show the page fetched. The local URL displayed should be something like this:

<file:///private/var/folders/y0/nz25304x2kxflcmmlc63dhtc0000gn/T/tmp3cgexmxr.html>

We accomplished the task of fetch the page and verify its consistency with the original page.

**Step Two – Defining the Expressions for Extraction**

The next step is to verify in the page fetched the content to be scraped and its XPath. It is important to highlight that the content of some web sites, like [www.ebay.com](http://www.ebay.com), change the response according to the User Agent used. For this reason, the XPath investigated need to be done in the page fetched.

For this training we will extract the title of items and its price. To create the XPath expression, we will click with the right button of mouse over the title and select “Inspect” which is the mentioned tool that will help us to get the XPath. As indicated in the figure right-click on the HTML portion, select options “Copy” and then “Copy Selector”.

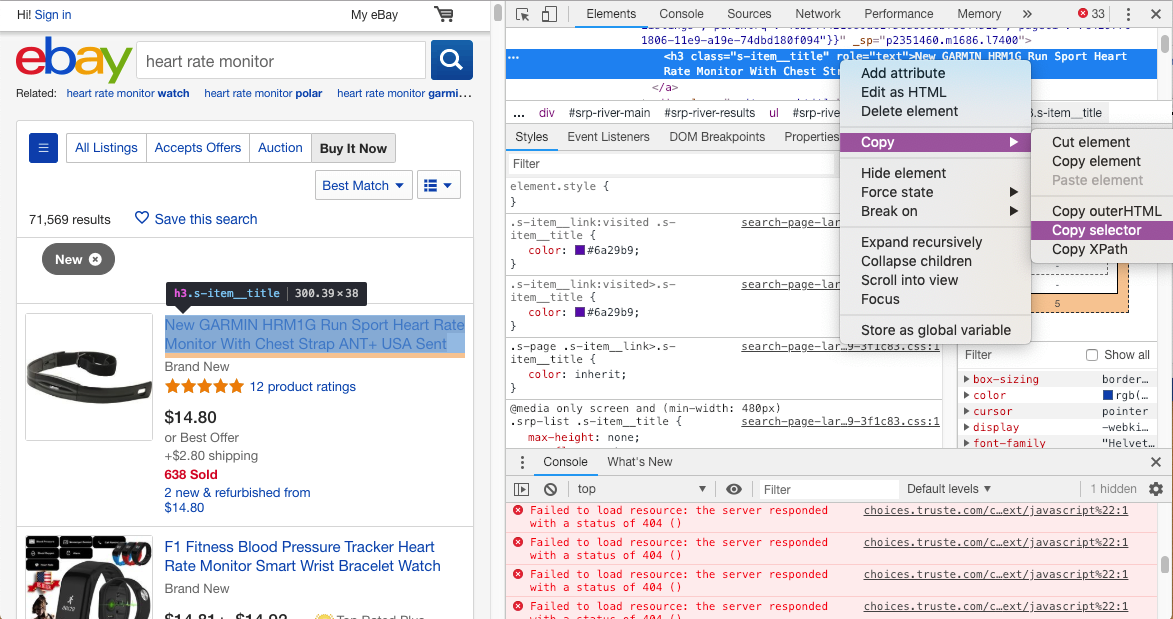


Figure 18-Copying the selector.

The expression copied will be used in the Scrapy Shell to test if we in fact can extract the content of interest. Expression:

**#srp-river-results-listing1 > div > div.s-item\_\_info.clearfix > a > h3**

Using the command response, we will test the expression above:

**>>> response.css('#srp-river-results-listing1 > div > div.s-item\_\_info.clearfix > a > h3')**

*[<Selector xpath="descendant-or-self::\*[@id = 'srp-river-results-listing1']/div/div[@class and contains(concat(' ', normalize-space(@class), ' '), ' s-item\_\_info ') and (@class and contains(concat(' ', normalize-space(@class), ' '), ' clearfix '))]/a/h3" data='<h3 class="s-item\_\_title" role="text">Ne'>]*

The result given is not generalized to all the titles, but to the specific title we copy the selector. In order to make it generalized it is necessary to remove the parent node which is “#srp-river-results-listing1 >”. Let’s execute without this part.

**>>> response.css('div > div.s-item\_\_info.clearfix > a > h3')**

*[<Selector xpath="descendant-or-self::div/div[@class and contains(concat(' ', normalize-space(@class), ' '), ' s-item\_\_info ') and (@class and contains(concat(' ', normalize-space(@class), ' '), ' clearfix '))]/a/h3" data='<h3 class="s-item\_\_title" role="text">Ne'>,*

***[…]***

*' s-item\_\_info ') and (@class and contains(concat(' ', normalize-space(@class), ' '), ' clearfix '))]/a/h3" data='<h3 class="s-item\_\_title s-item\_\_title--'>]*

The result above just show the first and last line of response. We can identify which the response potentially covers all the titles present in the document. The next step is to add the option ‘::text’ in the expression and a function called ‘extract()’ in order to clean-up the data retrieved.

**>>> response.css('div > div.s-item\_\_info.clearfix > a > h3::text').extract()**

*['New GARMIN HRM1G Run Sport Heart Rate Monitor With Chest Strap ANT+ USA Sent', 'Sports Blood Pressure/Heart Rate Monitor Fitness Smart Watch Wrist Band Bracelet', 'F1 Fitness Blood Pressure Tracker Heart Rate Monitor Smart Wrist Bracelet Watch', 'Sports Blood Pressure/Heart Rate Monitor Fitness Smart Watch Wrist Band Bracelet',*

The response above which shows few lines that the text titles extracted from the page. The next step is to collect the price from each item in the web page. Similar to the process of get the selector of the title we will proceed with the price.

**>>> response.css('#srp-river-results-listing1 > div > div.s-item\_\_info.clearfix > div.s-item\_\_details.clearfix > div:nth-child(1) > span')**

*[<Selector xpath="descendant-or-self::\*[@id = 'srp-river-results-listing1']/div/div[@class and contains(concat(' ', normalize-space(@class), ' '), ' s-item\_\_info ') and (@class and contains(concat(' ', normalize-space(@class), ' '), ' clearfix '))]/div[@class and contains(concat(' ', normalize-space(@class), ' '), ' s-item\_\_details ') and (@class and contains(concat(' ', normalize-space(@class), ' '), ' clearfix '))]/div[count(preceding-sibling::\*) = 0]/span" data='<span class="s-item\_\_price">$14.80</span'>]*

Once again the response was specific to the selected price, now we will generalize the expression removing the parent node, and add the entire command with ‘::text’ and the function ‘extract()’.

**>>> response.css('div > div.s-item\_\_info.clearfix > div.s-item\_\_details.clearfix > div:nth-child(1) > span::text').extract()**

*['$14.80', '$15.99', '$14.81', '$14.92', '$15.99', '$13.45', '$22.95', '$13.45', '$13.50', '$49.99', '$22.78', '$56.49', '$8.90', '$10.99', '$13.49', '$16.18', '$13.58', '$14.24',*

The response above shows the first lines and give us the initial evidence that the select got all the prices. The validation will be done in the next steps.

**2. Scraping Steps**

**Step 1 - Create a Project Directory**

After we test our selector, we will create our Scrapy project in a new folder that we will create, for this example was created the folder ‘SCRAPY\_PROJECT’ in ‘/Users/username/’.

**Step 2 - Create a Project**

To create a new Scrapy project we will use the command scrapy startproject <project\_name> [project\_dir], in this case we will use ‘ebay’ as the ‘project\_name’, and the default directory by omitting the argument ‘project\_dir’.

**$ scrapy startproject ebay**

*New Scrapy project 'ebay', using template directory '/anaconda3/lib/python3.6/site-packages/scrapy/templates/project', created in:*

*/Users/user\_name/SCRAPY\_PROJECTS/ebay*

*You can start your first spider with:*

*cd ebay*

*scrapy genspider example example.com*

**Step 3 – Create a Spider**

After create the project we will create our first spider inside of the directory defined (/Users/ user\_name/SCRAPY\_PROJECTS/ebay ) using the command scrapy genspider [options] <name> <domain>

**$ scrapy genspider ebay\_job ebay.com**

*Created spider 'ebay\_job' using template 'basic' in module:*

*ebay.spiders.ebay\_job*

The command results in a creation of a directory structure pre-defined as depicted in the figure 19.

**Step 4 – Editing the job file**

After create the spider we need to configure it. We open the spider file ‘ebay\_job.py’ using an IDE. In this case was used Microsoft VSCode.

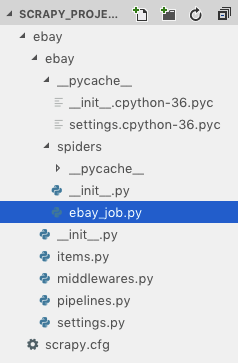


Figure 19-Spider Directory Structure

Opening the file ‘ebay\_job.py’ we can see the default content created:

# -\*- coding: utf-8 -\*-

import scrapy

class EbayJobSpider(scrapy.Spider):

name = 'ebay\_job'

allowed\_domains = ['ebay.com']

start\_urls = ['http://ebay.com/']

def parse(self, response):

pass

We will edit few components in this file. First it is necessary to substitute the ‘start\_url’ by the link we investigated previously:

'https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=heart+rate+monitor&\_sacat=0&LH\_ItemCondition=3&rt=nc&LH\_BIN=1&\_ipg=200'

# -\*- coding: utf-8 -\*-

import scrapy

class EbayJobSpider(scrapy.Spider):

name = 'ebay\_job'

allowed\_domains = ['ebay.com']

start\_urls = [“https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=heart+rate+monitor&\_sacat=0&LH\_ItemCondition=3&rt=nc&LH\_BIN=1&\_ipg=200”]

def parse(self, response):

pass

It is also necessary to edit the function ‘parse()’. Basically, it is added two variables: ‘titles’ and ‘prices’, containing the previous command used to test the CSS expression (in blue).

In addition (in purple) was created a ‘loop for’ using the python function ‘zip()’ to aggregate the iterables and return tuples into the function ‘EbayItem()’ which is defined in the file ‘items.py’. The main goal is to structure the outcome data of Scrapy.

# -\*- coding: utf-8 -\*-

import scrapy

from ebay.items import EbayItem

class EbayJobSpider(scrapy.Spider):

name = 'ebay\_job'

allowed\_domains = ['ebay.com']

start\_urls = ["https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=heart+rate+monitor&\_sacat=0&LH\_ItemCondition=3&rt=nc&LH\_BIN=1&\_ipg=200"]

def parse(self, response):

**titles = response.css("div > div.s-item\_\_info.clearfix > a > h3::text").extract()**

**prices = response.css("div > div.s-item\_\_info.clearfix > div.s-item\_\_details.clearfix > div:nth-child(1) > span").extract()**

**for item in zip(titles, prices):**

**new\_item = EbayItem()**

**new\_item['titles'] = item[0]**

**new\_item['prices'] = item[1]**

**yield new\_item**

**Step 5 – Edit Item.py**

It is necessary to edit ‘item.py’ in order to create the Let’s see in detail the ‘items.py’ file with its default content:

# -\*- coding: utf-8 -\*-

# Define here the models for your scraped items

#

# See documentation in:

# https://doc.scrapy.org/en/latest/topics/items.html

import scrapy

class EbayItem(scrapy.Item):

# define the fields for your item here like:

# name = scrapy.Field()

pass

We will add the fields ‘titles’ and ‘prices’ which will be the attributes for the resulting dataset (in purple).

# -\*- coding: utf-8 -\*-

# Define here the models for your scraped items

#

# See documentation in:

# https://doc.scrapy.org/en/latest/topics/items.html

import scrapy

class EbayItem(scrapy.Item):

# define the fields for your item here like:

# name = scrapy.Field()

**titles = scrapy.Field()**

**prices = scrapy.Field()**

Now that you created the fields in the class EbayItem, import it in the ebay\_job.py as indicated below in purple:

# -\*- coding: utf-8 -\*-

import scrapy

**from ebay.items import EbayItem**

class EbayJobSpider(scrapy.Spider):

name = 'ebay\_job'

allowed\_domains = ['ebay.com']

start\_urls = ['https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=heart+rate+monitor&\_sacat=0&LH\_ItemCondition=3&rt=nc&LH\_BIN=1&\_ipg=200']

def parse(self, response):

titles = response.css('div > div.s-item\_\_info.clearfix > a > h3::text').extract()

prices = response.css('div > div.s-item\_\_info.clearfix > div.s-item\_\_details.clearfix > div:nth-child(1) > span').extract()

for item in zip(titles, prices):

new\_item = EbayItem()

new\_item['titles'] = item[0]

new\_item['prices'] = item[1]

yield new\_item

**Step 6 – Crawling**

After we set the scrapy framework structure it is time to run the Crawling function. To do this we will execute the following command:

$ scrapy crawl ebay\_job

The result - if everything is right – will be a long log with DEBUG and INFO levels in the beginning and showing the extracted data, and finishing with the Scrapy statistics like the sample below:

[...]

2019-01-20 20:12:32 [scrapy.core.scraper] DEBUG: Scraped from <200 https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=heart+rate+monitor&\_sacat=0&LH\_ItemCondition=3&rt=nc&LH\_BIN=1&\_ipg=200>

{'prices': '<span class="s-item\_\_price">$13.39<span class="DEFAULT"> to '

'</span>$13.79</span>',

'titles': 'Original Xiaomi Mi Band 2 Smart Wristband Bracelet Heart Rate '

'Monitor Smartwatch'}

2019-01-20 20:12:32 [scrapy.core.engine] INFO: Closing spider (finished)

2019-01-20 20:12:32 [scrapy.statscollectors] INFO: Dumping Scrapy stats:

{'downloader/request\_bytes': 759,

'downloader/request\_count': 2,

'downloader/request\_method\_count/GET': 2,

'downloader/response\_bytes': 103540,

'downloader/response\_count': 2,

'downloader/response\_status\_count/200': 2,

'finish\_reason': 'finished',

'finish\_time': datetime.datetime(2019, 1, 21, 1, 12, 32, 735435),

'item\_scraped\_count': 206,

'log\_count/DEBUG': 209,

'log\_count/INFO': 7,

'memusage/max': 51384320,

'memusage/startup': 51376128,

'response\_received\_count': 2,

'scheduler/dequeued': 1,

'scheduler/dequeued/memory': 1,

'scheduler/enqueued': 1,

'scheduler/enqueued/memory': 1,

'start\_time': datetime.datetime(2019, 1, 21, 1, 12, 29, 486444)}

**Step 7 – Exporting the extracted data for Data Analytics**

It is not enough the extraction of data, it is necessary to export it in a tabular dataset for further analyses. In order to do it is necessary to add arguments in the crawled command used:

$ scrapy crawl ebay\_job -o ebay\_extraction.csv -t csv

The resulting file will be located in the same directory that you execute the crawling command, and a test should be done to check how is the dataset.

**Part 4: Data Analysis**

After the collection of data analysis, it is necessary to evaluate the result and in some cases to clean-up or transform specificities of data collected. To process it we will use the Jupyter notebook.

**1. Starting the Jupyter Notebook**

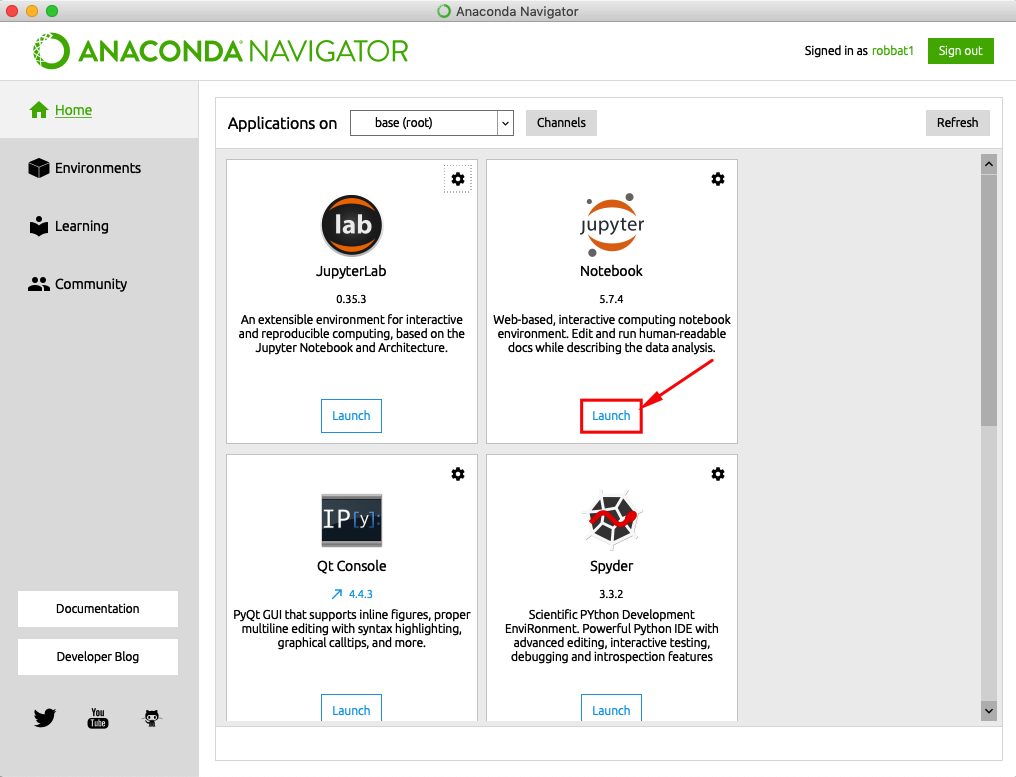
****

Figure 20-Initiating Jupyter Notebook.

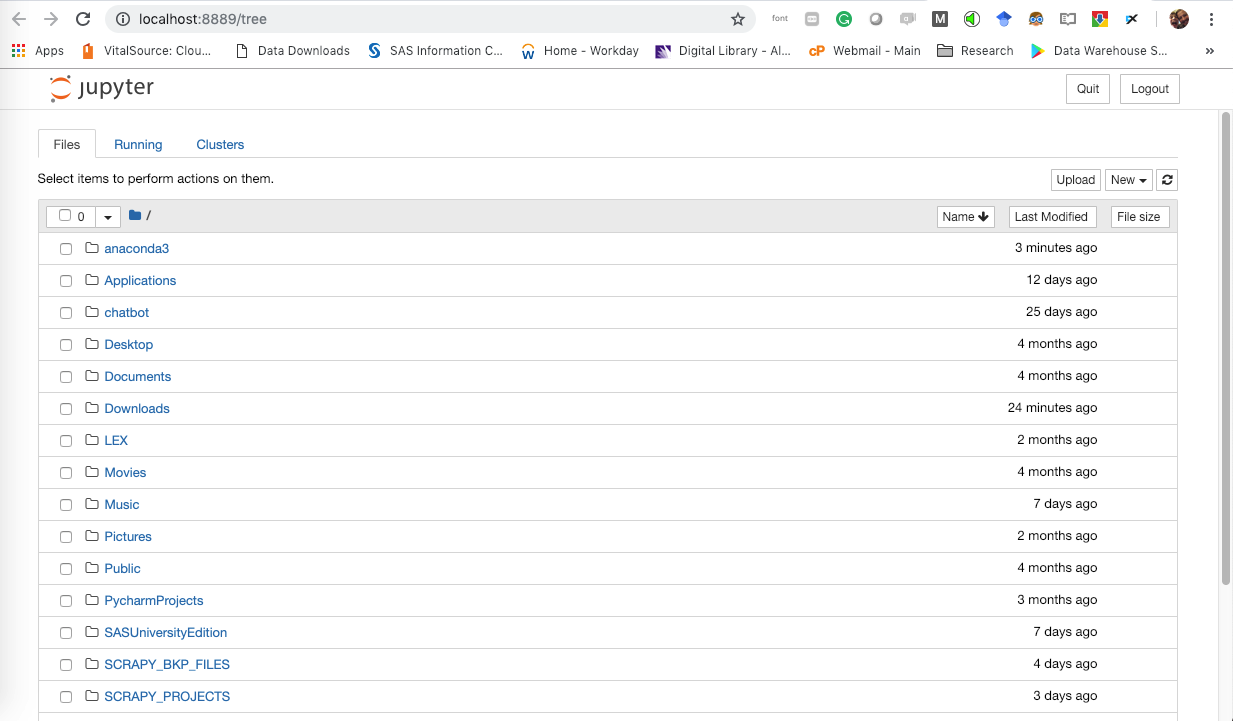


Figure 21-A web page is opened with Jupyter Notebook.

After open the Jupyter Notebook you will need to navigate in the directories to reach the Scrapy directory created.

**2 – Creating a new notebook.**

Once in the directory you can create your new note book clicking on File > New Notebook > Python 3.

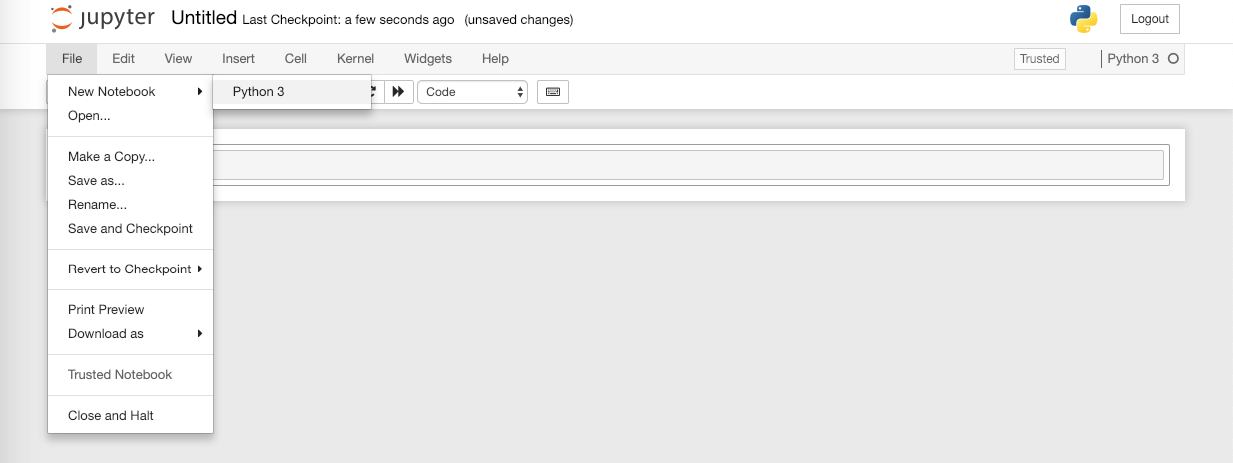


Figure 22-Creating a new Jupyter Notebook.

**3 – Initiating to fill the notebook.**

It is important that you have a minimum knowledge of Jupyter Notebook and Python to go further. In case you fell that you need help a good introductory course in the internet that you can attend.

The initial part of notebook is Data Wrangling. You will be refining the dataset generated by Scrapy before any analysis.

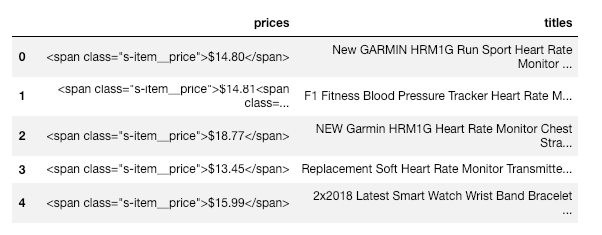


In this case it is necessary to import three libraries:

* re: Regular Expression. Allow the use of regular expression to select certain patterns of data.
* os: Operationa System, allow to navigate in the different directories of the computer and to do several operations like listing files, directories, make directories, renaming files, changing the default directory.
* Pandas: a powerful library that enable data manipulation in a simple way.



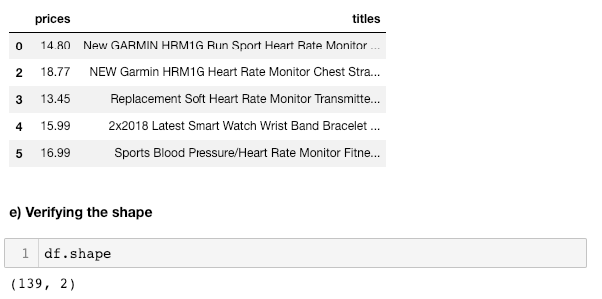
After move to the directory that the cvs file is, it is necessary to import the CSV file into a variable. Note that was used the function ‘head’ to print the first five lines of the dataset stored in the variable ‘df’.



Next step we will remove the “$” sign present in the content and the HTML tags. It is important to have a cleaned data in order to do analysis. After this change in the data it is necessary to remove the lines with ranges of price, which is not interest.



The result is a cleaned data prepared for the data analysis based on descriptive analysis.

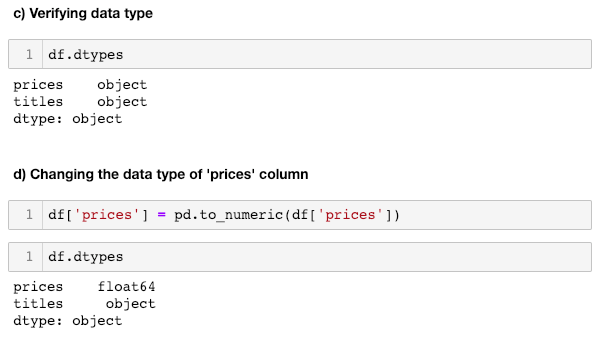


**4 – Data Analytics**

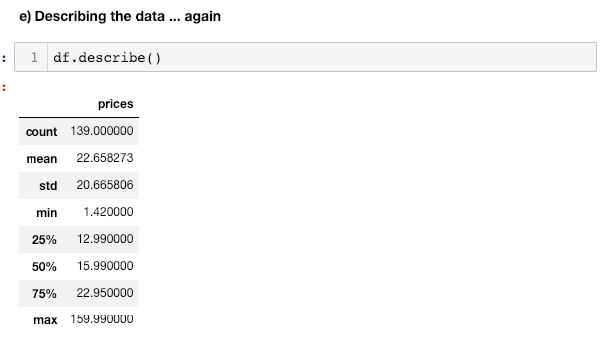
In order to initiate the analysis, it is necessary to check the data types for the attributes.

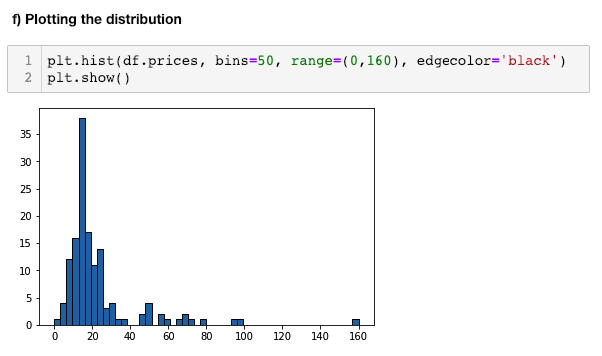


We can identify above that the function describe that should give descriptive statistics like max, min, mean, standard deviation, etc. It is showing information typical related to string (object). To fix this, it is necessary to apply the function to\_numeric() in the ‘prices’ column.



Once the datatype is correct we apply the function describe(). The result show us clearly the main characteristics of the data. In order to improve this understanding, we will print the histogram to depict the distribution of data.



****

The histogram revels the concentration of data on the right of the histogram, some outliers and the right skew nature of the data.

**Part 5: Assignment**

**Assignment 1:**

**Execute the following instructions making notes of all your steps taken.**

1. Visit a page https://www.instructables.com, search for “monitor heart rate” (using quotes).
2. Go to Scrapy Shell and:
   1. Use ‘**fetch’** to get the page resulted from your search.
   2. Use ‘**view(response)’** to visualize the page captured.
   3. Use the techniques described in this workshop to find the class used in the HTML file and to define the CSS expression for the following items:
      1. Title
      2. Channel
      3. Favorites
      4. Views
3. Create a directory in your computer with the name “**instructables”.**
4. Using terminal go to the directory created. You can use the command reference for Linux or DOS according to your operating system in the end of this document.
5. Create a Scrapy project using ‘**scrapy startproject instructables’** command.
6. Create a Spider using the command **‘scrapy genspider instructables\_job instructables.com’.**
7. Edit the Scrapy files according to the instruction of this document to configure the Scrapy to scrap the **instructables** page resulted from the search.
8. Export the scraping into a CSV file.
9. Report the difficult found in the process.

**Part 6: DOS Commands**

Prepared by www.opensource.com

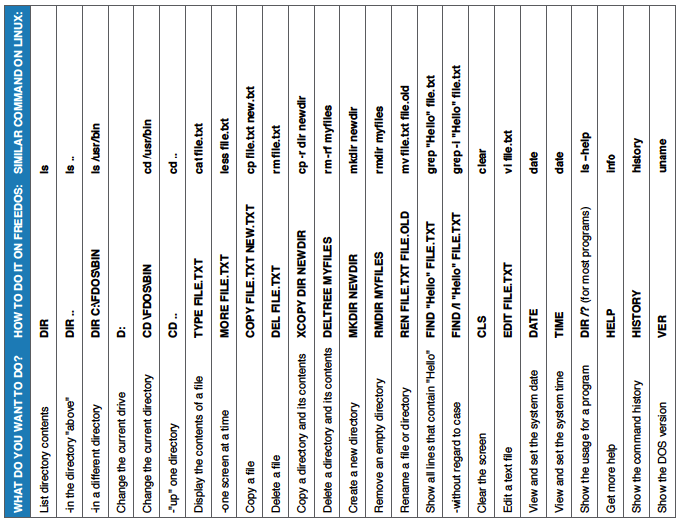
****

Figure 23-DOS Commands 1

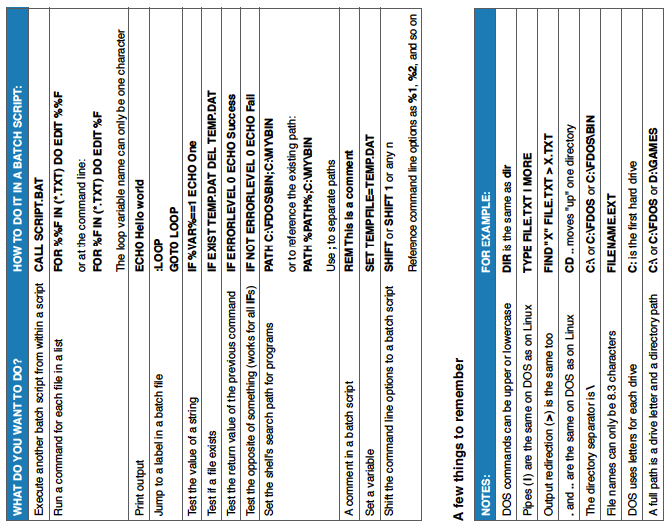
****

Figure 24-DOS Commands 2

**Part 7: Linux Commands**

Prepared by FossWire.com

|  |  |
| --- | --- |
|  |  |

1. Vargiu & Urru (2013). "Exploiting web scraping in a collaborative filtering- based approach to web advertising". Artificial Intelligence Research. 2 (1). doi:10.5430/air.v2n1p44 [↑](#footnote-ref-1)
2. "Release notes — Scrapy documentation". doc.scrapy.org. Retrieved 2019-01-24. [↑](#footnote-ref-2)
3. Guttag, John V. (2016-08-12). Introduction to Computation and Programming Using Python: With Application to Understanding Data. MIT Press. ISBN 978-0-262-52962-4. [↑](#footnote-ref-3)
4. "What is Anaconda, Inc.?". docs.anaconda.com. [↑](#footnote-ref-4)
5. L.Rosen (2013). Open Source Licensing: Software Freedom and Intellectual Property Law. ISBN 978-0131487871 [↑](#footnote-ref-5)
6. "Project Jupyter - About Us". 2019-01-27. Retrieved 2018-05-03 [↑](#footnote-ref-6)
7. https://conda.io/docs/user-guide/install/test-installation.html [↑](#footnote-ref-7)
8. Icons made by Freepik, except “download” icon made by Google, and “cloud” icon made by SimpleIcon, from www.flaticon.com [↑](#footnote-ref-8)