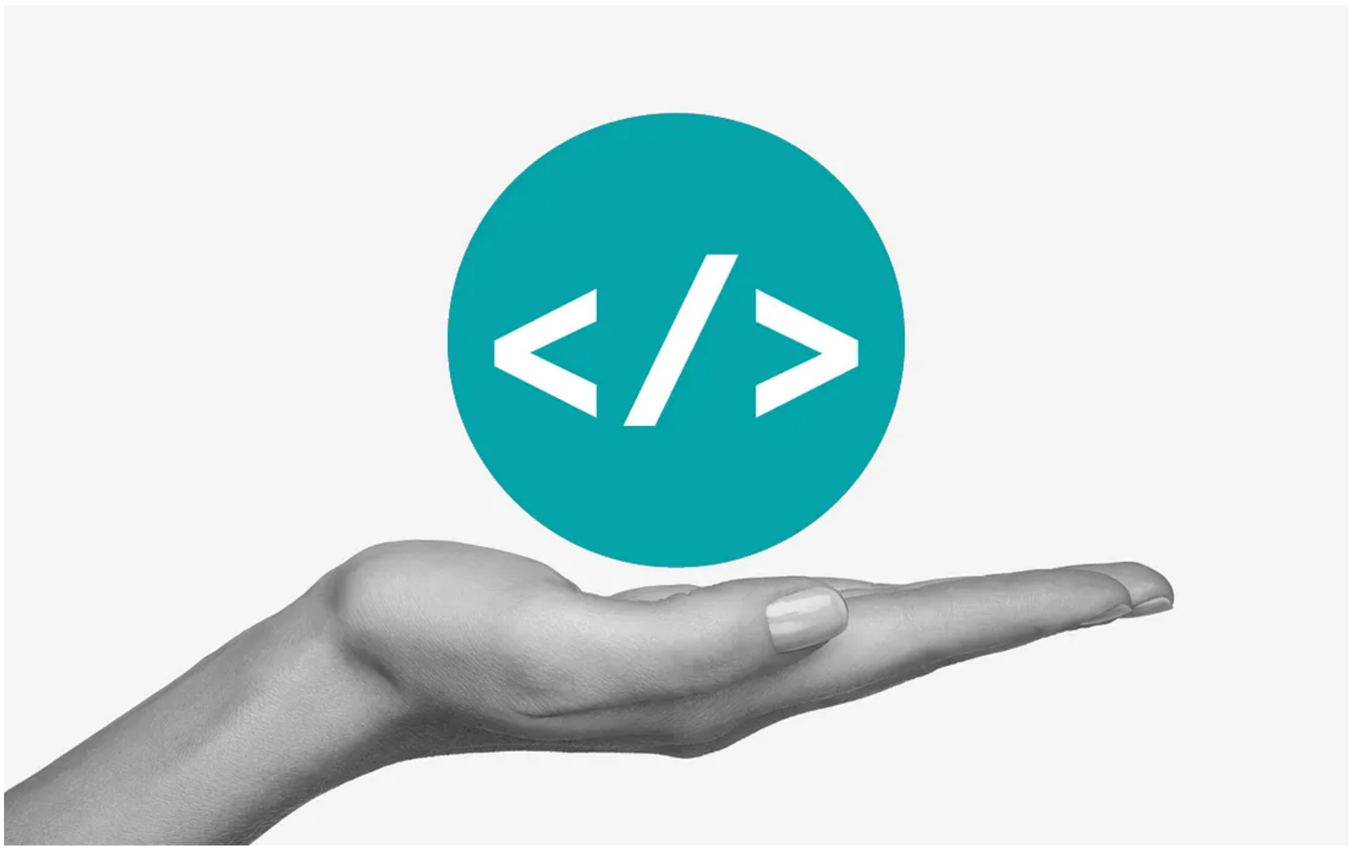
Published in Towards Data Science

Miguel Fernández Zafra Follow
Jul 9, 2019 · 8 min read · ✦ · Listen

AN END TO END MACHINE LEARNING PROJECT

Web Scraping news articles in Python

Building a web scraping application in Python made simple



Source

This article is the second of a series in which I will cover the **whole process** of developing a machine learning project. If you have not read the first one, I strongly encourage you to do it here.

The project involves the creation of a **real-time web application** that gathers data from several newspapers and shows a summary of the different topics that are being discussed in the news articles.

This is achieved with a supervised machine learning **classification model** that is able to predict the category of a given news article, a **web scraping method** that gets the latest news from the newspapers, and an interactive **web application** that shows the obtained results to the user.

As I explained in the first post of this series, the motivation behind writing these articles is that a lot of the articles or content published on the internet, books or literature regarding data science and machine learning models focus on the modelling part with the training data. However, a machine learning project is **much more** than that: once you have a trained model, you need to feed new data to it and what is more important, you need to provide **useful insights** to the final user.

The whole process is divided in three different posts:

- Classification model training (link)
- News articles web scraping (this post)
- App creation and deployment (link)

The github repo can be found here. It includes all the code and a complete report.

In the first article, we developed the **text classification model** in Python, which allowed us to get a certain news article text and predict its category with an overall good accuracy.

This post covers the second part: News articles web scraping. We'll create a script that **scrapes the latest news articles** from different newspapers and stores the text, which will be fed into the model afterwards to get a prediction of its category. We'll cover it in the following steps:

1. A brief introduction to webpages and HTML
2. Web scraping with BeautifulSoup in Python

1. A brief introduction to webpage design and HTML

If we want to be able to extract news articles (or, in fact, any other kind of text) from a website, the first step is to know how a website *works*. We will follow an example with the Towards Data Science webpage.

When we insert an *url* into the web browser (i.e. Google Chrome, Firefox, etc…) and access to it, what we see is the combination of **three technologies**:

1. **HTML** (HyperText Markup Language): it is the standard language for adding content to a website. It allows us to insert text, images and other things to our site. In one word, HTML determines the **content** of a webpage.
2. **CSS** (Cascading Style Sheets): this language allows us to set the visual design of a

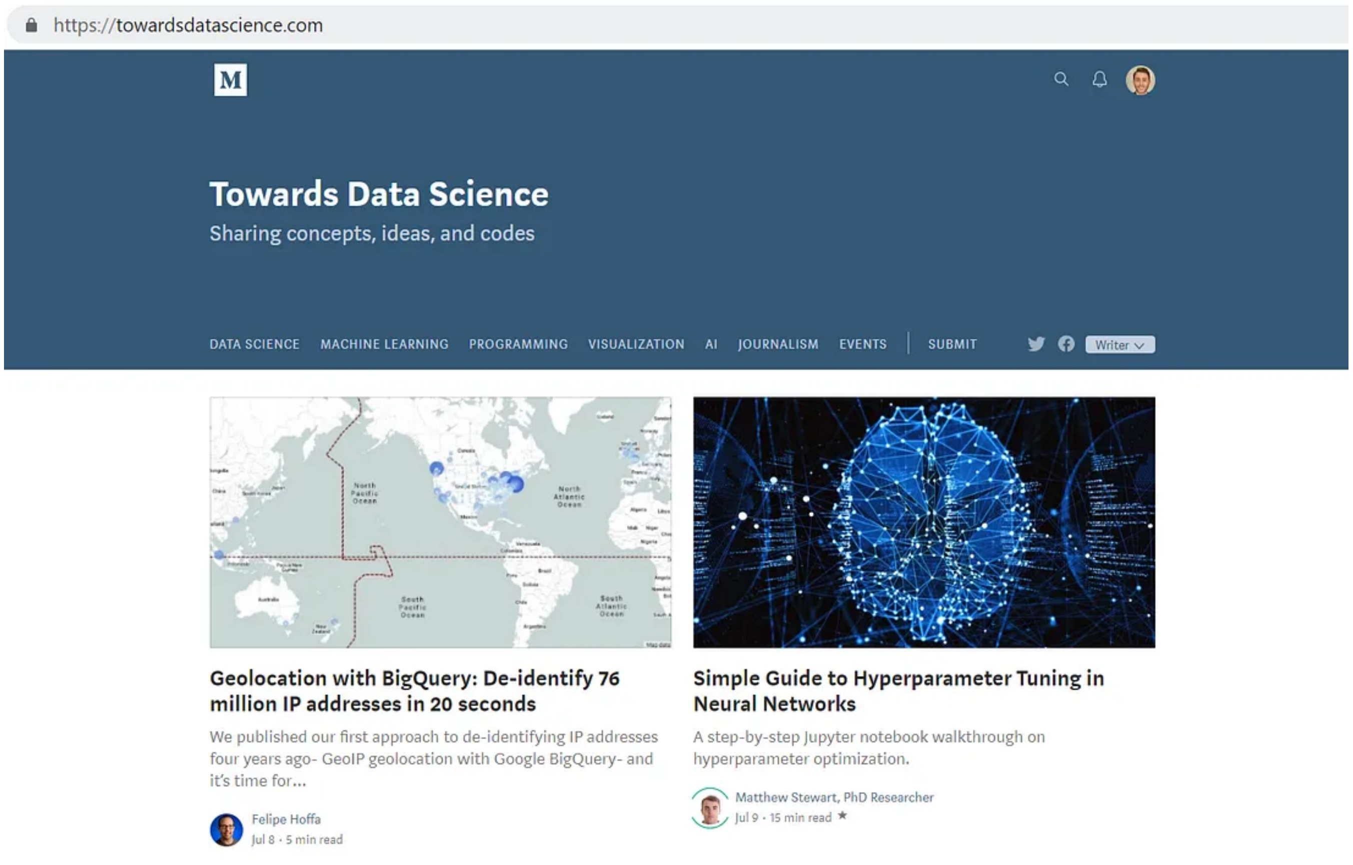
website. This means, it determines the **style** of a webpage.

3. **JavaScript**: JavaScript allows us to make the content and the style **interactive.**

Note that these three are programming languages. They will allow us to create and manipulate **every aspect** of the design of a webpage.

However, if we want a website to be accessible to every one in a browser, we need to know about additional things: standing up a web server, using a certain domain, etc… But since we are only interested in extracting content from a webpage, this will be enough for today.

Let’s illustrate these concepts with an example. When we visit the Towards Data Science homepage, we see the following:



Geolocation with BigQuery: De-identify 76 million IP addresses in 20 seconds

We published our first approach to de-identifying IP addresses four years ago- GeoIP geolocation with Google BigQuery- and it's time for...

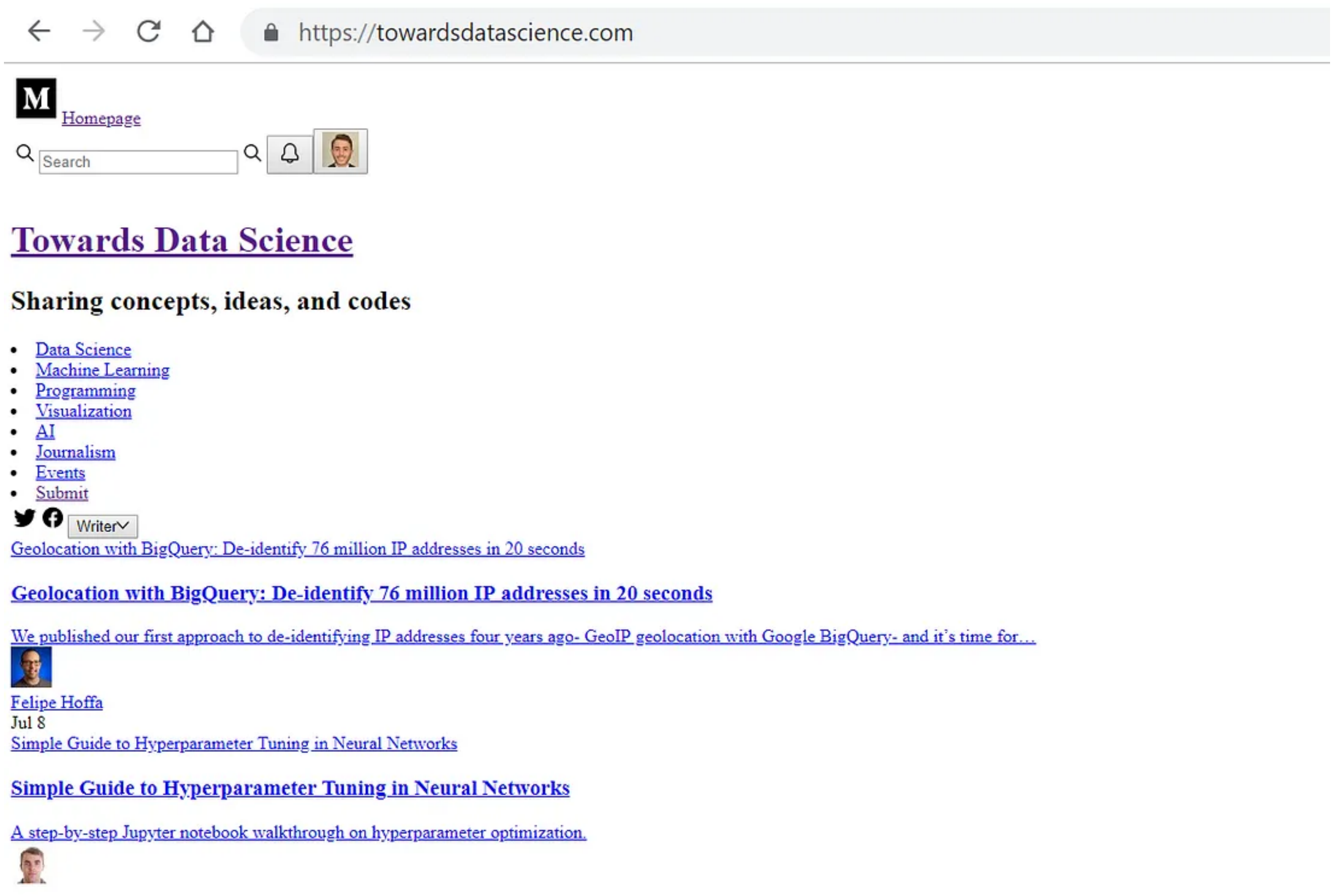
Felipe Hoffa
Jul 8 · 5 min read

Simple Guide to Hyperparameter Tuning in Neural Networks

A step-by-step Jupyter notebook walkthrough on hyperparameter optimization.

Matthew Stewart, PhD Researcher
Jul 9 · 15 min read

If we deleted the **CSS** content from the webpage, we would see something like this:



https://towardsdatascience.com

Homepage

Search

Towards Data Science

Sharing concepts, ideas, and codes

- Data Science
- Machine Learning
- Programming
- Visualization
- AI
- Journalism
- Events
- Submit

Writer

Geolocation with BigQuery: De-identify 76 million IP addresses in 20 seconds

Geolocation with BigQuery: De-identify 76 million IP addresses in 20 seconds

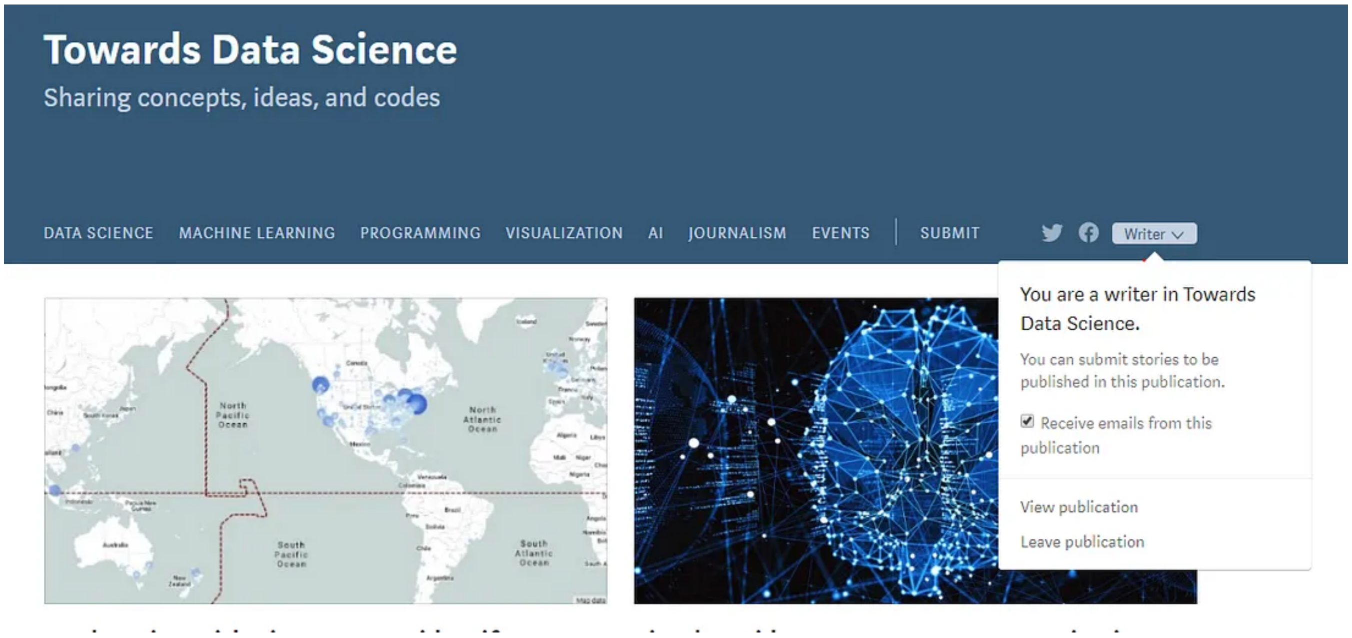
We published our first approach to de-identifying IP addresses four years ago- GeoIP geolocation with Google BigQuery- and it's time for…

Felipe Hoffa
Jul 8
Simple Guide to Hyperparameter Tuning in Neural Networks

Simple Guide to Hyperparameter Tuning in Neural Networks

A step-by-step Jupyter notebook walkthrough on hyperparameter optimization.

And if we disabled **JavaScript**, we would not be able to use this pop-up no more:



Towards Data Science

Sharing concepts, ideas, and codes

DATA SCIENCE MACHINE LEARNING PROGRAMMING VISUALIZATION AI JOURNALISM EVENTS SUBMIT Writer

You are a writer in Towards Data Science.

You can submit stories to be published in this publication.

☑ Receive emails from this publication

View publication

Leave publication

At this point, I'll ask the following question:

> *"If I want to extract the content of a webpage via web scraping, where do I need to look*

> *up?”*

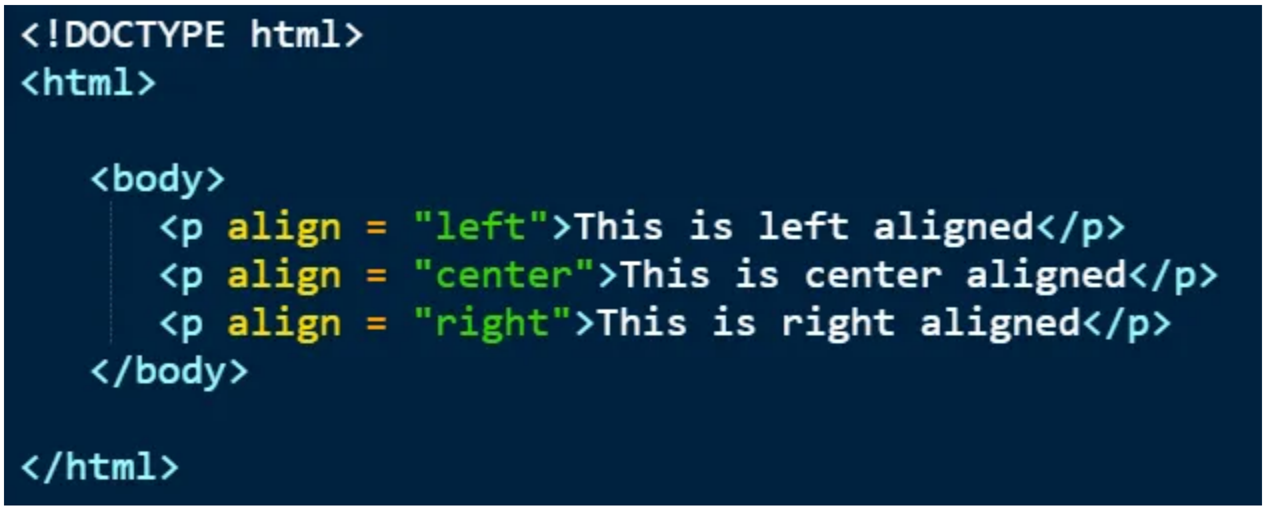
If your answer was the **HTML** code, then you’re absolutely getting it. In the above example we can see that after disabling CSS, the content (text, images, etc...) is still there.

So, the last step before performing web scraping methods is to understand a bit of the **HTML language**.

HTML is, from a really basic point of view, composed of **elements** that have **attributes.** An element could be a paragraph, and an attribute could be that the paragraph is in bold letter.

There are a lot of different types of elements, each one with its own attributes. To identify an element (this means, as an example, to set if some text is a heading or a paragraph) we use **tags**. These tags are represented with the <> symbols (for example, a *<p>* tag means a certain text is acting as a paragraph).

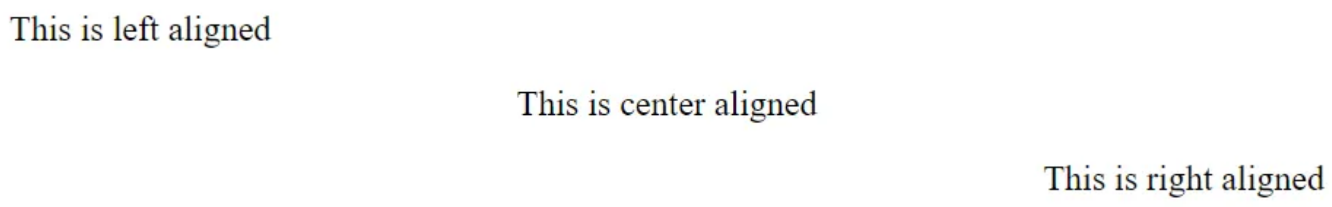
For example, this HTML code below allows us to change the alignment of the paragraphs:



```
<!DOCTYPE html>
<html>

   <body>
      <p align = "left">This is left aligned</p>
      <p align = "center">This is center aligned</p>
      <p align = "right">This is right aligned</p>
   </body>

</html>
```



```
This is left aligned

                    This is center aligned

                                          This is right aligned
```

Consequently, when we visit a website, we will be able to find **the content** and **its properties** in the HTML code.

Once we have presented these concepts, we are ready for some web scraping!

2. Web scraping with BeautifulSoup in Python

There are several packages in Python that allow us to scrape information from webpages. One of the most common ones is **BeautifulSoup**. The official package information can be found here.

BeautifulSoup allows us to parse the HTML content of a given URL and access its elements by identifying them with their tags and attributes. For this reason, we will use it to extract certain pieces of text from the websites.

It is an extremely easy-to-use yet powerful package. With almost 3–5 lines of code we will be able to extract any text we want from the internet.

To install it, please type the following code into your Python distribution:

```
! pip install beautifulsoup4
```

So as to provide BeautifulSoup with the HTML code of any page, we will also need to import the `requests` module. In order to install it if it's not already included in your python distribution, please type:

```
! pip install requests
```

We will use the `requests` module to get the HTML code from the page and then navigate through it with the BeautifulSoup package. We will learn to use two commands that will be enough for our task:

- `find_all(element tag, attribute)` : it allows us to locate any HTML element from a webpage introducing its tag and attributes. This command will locate all the elements of the same type. In order to get only the first one, we can use `find()` instead.

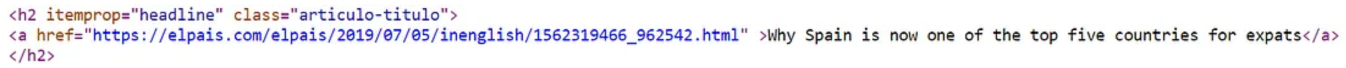
- `get_text()` : once we have located a given element, this command will allow us to extract the text inside.

So, at this point, what we need to do is to **navigate through the HTML code of our webpage** (for example, in Google Chrome we need to enter the webpage, press right click button and go to *See source code*) and **locate the elements we want to scrape.** We can simply do this searching with Ctrl+F or Cmd+F once we are seeing the source code.

Once we have identified the elements of interest, we will **get the HTML code** with the `requests` module and **extract those elements** with BeautifulSoup.

We will carry out an example with the *El Pais English* newspaper. We will first try to web scrape the news articles titles from the frontpage and then extract the text out of them.

Once we enter the website, we need to inspect the HTML code to locate the news articles. After a fast look we can see that each article in the frontpage is an element like this:



```
<h2 itemprop="headline" class="articulo-titulo">
<a href="https://elpais.com/elpais/2019/07/05/inenglish/1562319466_962542.html" >Why Spain is now one of the top five countries for expats</a>
</h2>
```

The title is an `<h2>` (heading-2) element with `itemprop=”headline"` and `class=”articulo-titulo"` atributes. It has an `<a>` element with an `href` attribute which contains the text. So, in order to extract the text, we need to code the following commands:

```
# importing the necessary packages
import requests
from bs4 import BeautifulSoup
```

With the `requests` module we can get the HTML content and save into the `coverpage` variable:

```
r1 = requests.get(url)
coverpage = r1.content
```

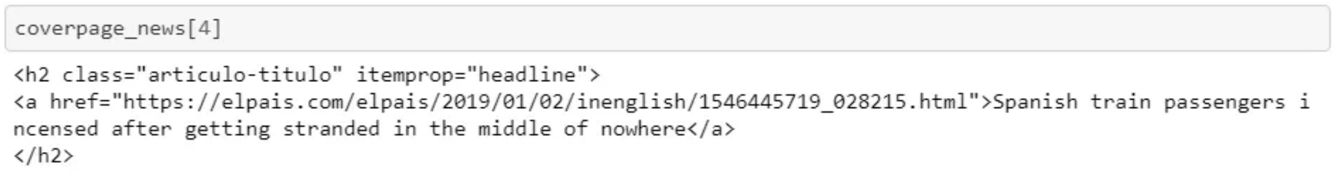
Next, we need to create a *soup* in order to allow BeautifulSoup to work:

```
soup1 = BeautifulSoup(coverpage, 'html5lib')
```

And finally, we can locate the elements we are looking for:

```
coverpage_news = soup1.find_all('h2', class_='articulo-titulo')
```

This will return a list in which each element is a news article (because with `find_all` we are getting all ocurrences):



```
coverpage_news[4]
```

```
<h2 class="articulo-titulo" itemprop="headline">
<a href="https://elpais.com/elpais/2019/01/02/inenglish/1546445719_028215.html">Spanish train passengers i
ncensed after getting stranded in the middle of nowhere</a>
</h2>
```

If we code the following command, we will be able to extract the text:

```
coverpage_news[4].get_text()
```

If we want to access the value of an attribute (in this case, the link), we can type the following:

```
coverpage_news[4]['href']
```

And we’ll get the link in plain text.

If you have understood until this point, you are ready to web scrape **any content** you want.

The next step would be to access each of the news articles content with the `href` attribute, get the source code again and find the paragraphs in the HTML code to finally get them with BeautifulSoup. It’s the same idea as before, but we need to locate the tags and attributes that identify the news article content.

The code of the full process is the following. I will show the code but won’t enter in the same detail as before since it’s exactly the same idea.

```
# Scraping the first 5 articles
number_of_articles = 5

# Empty lists for content, links and titles
news_contents = []
list_links = []
  i

            range(0, number_of_articles):

    # only news articles (there are also albums and other things)
    if "inenglish" not in coverpage_news[n].find('a')['href']:
        continue

    # Getting the link of the article
    link = coverpage_news                 f']
    list_links.append(lin

    # Getting the title
    title = coverpage_news[n].find('a').get_text()
    list_titles.append(title)

    # Reading the content (it is divided in paragraphs)
    article = requests.get(link)
    article_content = article.content
    soup_article = BeautifulSoup(article_content, 'html5lib')
    body = soup_article.find_all('div', class_='articulo-cuerpo')
    x = body[0].find_all('p')

    # Unifying the paragraphs
    list_paragraphs = []
    for p in np.arange(0, len(x)):
        paragraph = x[p].get_text()
        list_paragraphs.append(paragraph)
        final_article = " ".join(list_paragraphs)

    news_contents.append(final_article)
```

All the details can be found in my github repo.

m [illegible] code is only **useful for this webpage in particular**. If we want to scrape another one, we **should expect** that elements are identified with **different tags and attributes**. But once we know how to identify them, the process is exactly the same.

At this point, we are able to extract the content of different news articles. The final step is to apply the machine learning model we trained in the first post to predict its categories and show a summary to the user. This will be covered in the final post of this series.