Web scraping - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 2

Classification video - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 10

**1. What is web scraping?**

Web scraping (or data scraping) is a technique used to collect content and data from the internet. This data is usually saved in a local file so that it can be manipulated and analyzed as needed. If you’ve ever copied and pasted content from a website into an Excel spreadsheet, this is essentially what web scraping is, but on a very small scale.

However, when people refer to ‘web scrapers,’ they’re usually talking about software applications. Web scraping applications (or ‘bots’) are programmed to visit websites, grab the relevant pages and extract useful information. By automating this process, these bots can extract huge amounts of data in a very short time. This has obvious benefits in the digital age, when big data—which is constantly updating and changing—plays such a prominent role.

**What kinds of data can you scrape from the web?**

If there’s data on a website, then in theory, it’s scrapable! Common data types organizations collect include images, videos, text, product information, customer sentiments and reviews (on sites like Twitter, Yell, or Tripadvisor), and pricing from comparison websites. There are some legal rules about what types of information you can scrape, but we’ll cover these later on.

**2. What is web scraping used for?**

Web scraping has countless applications, especially within the field of data analytics. Market research companies use scrapers to pull data from social media or online forums for things like customer sentiment analysis. Others scrape data from product sites like Amazon or eBay to support competitor analysis.

Meanwhile, Google regularly uses web scraping to analyze, rank, and index their content. Web scraping also allows them to extract information from third-party websites before redirecting it to their own (for instance, they scrape e-commerce sites to populate Google Shopping).

Many companies also carry out contact scraping, which is when they scrape the web for contact information to be used for marketing purposes. If you’ve ever granted a company access to your contacts in exchange for using their services, then you’ve given them permission to do just this.

There are few restrictions on how web scraping can be used. It’s essentially down to how creative you are and what your end goal is. From real estate listings, to weather data, to carrying out SEO audits, the list is pretty much endless!

However, it should be noted that web scraping also has a dark underbelly. Bad players often scrape data like bank details or other personal information to conduct fraud, scams, intellectual property theft, and extortion. It’s good to be aware of these dangers before starting your own web scraping journey. Make sure you keep abreast of the legal rules around web scraping. We’ll cover these a bit more in section six.

**3. How does a web scraper function?**

So, we now know what web scraping is, and why different organizations use it. **But how does a web scraper work?** While the exact method differs depending on the software or tools you’re using, all web scraping bots follow three basic principles:

* Step 1: Making an HTTP request to a server
* Step 2: Extracting and parsing (or breaking down) the website’s code
* Step 3: Saving the relevant data locally

### Step 1: Making an HTTP request to a server

As an individual, when you visit a website via your browser, you send what’s called an HTTP request. This is basically the digital equivalent of knocking on the door, asking to come in. Once your request is approved, you can then access that site and all the information on it. Just like a person, a web scraper needs permission to access a site. Therefore, the first thing a web scraper does is send an HTTP request to the site they’re targeting.

### Step 2: Extracting and parsing the website’s code

Once a website gives a scraper access, the bot can read and extract the site’s HTML or XML code. This code determines the website’s content structure. The scraper will then parse the code (which basically means breaking it down into its constituent parts) so that it can identify and extract elements or objects that have been predefined by whoever set the bot loose! These might include specific text, ratings, classes, tags, IDs, or other information.

### Step 3: Saving the relevant data locally

Once the HTML or XML has been accessed, scraped, and parsed, the web scraper will then store the relevant data locally. As mentioned, the data extracted is predefined by you (having told the bot what you want it to collect). Data is usually stored as structured data, often in an Excel file, such as a .csv or .xls format.

As we will show in the below example in wed scarping which used in our project in **Elasticsearch.py** file:

#import libraries

from bs4 import BeautifulSoup

import csv

import requests

#extract transcript from data received

def filter\_(script):

    skip = 0

    j = -1

    transcript = ''

    while( j < len(script) ):

        j += 1

        skip = 0

        if(j >= len(script)):

            break

        if script[j] == '<':

            while(1):

                if(j >= len(script)):

                    break

                if(script[j] == '>'):

                    j+=1

                    break

                j+=1

        if(j >= len(script)):

            break

        if (script[j] >= '0' and script[j] <= '9') or script[j] == ':' or script[j] == '\n' or script[j] == '\t':

            skip = 1

        if skip == 1:

            continue

        transcript += script[j]

    return transcript

TED\_TALK\_URL = 'https://www.ted.com/index.php/talks/'

#this function is for request an url

def get\_html(url):

    html = requests.get(url)

    soup = BeautifulSoup(html.content, 'html.parser')

    return soup

# for Get trascripts for ted.com videos

def get\_transcript(talk\_url):

    talk\_url = talk\_url[:(len(talk\_url)-12)]

    if not talk\_url.startswith('https://'):

        talk\_url = TED\_TALK\_URL + talk\_url

    talk\_url = talk\_url  + '/transcript?language=en'

    print(talk\_url)

    soup = get\_html(talk\_url)

    script = soup.find\_all('div', class\_='Grid Grid--with-gutter d:f@md p-b:4')

    return filter\_(str(script))

# load our data set

with open('data.csv', encoding='UTF8') as file\_obj:

    # Create reader object by passing the file

    # object to DictReader method

    reader\_obj = csv.DictReader(file\_obj)

    # Iterate over each row in the csv file

    # using reader object

    record = dict()

    i = 0

    for row in reader\_obj:

        record['id'] = i

        record['title'] = row['title']

        record['author'] =  row['speaker\_name']

        record['date'] = row['posted\_date']

        record['views'] = row['views']

        record['tags'] = row['tags']

        record['link'] = row['Link']

        # get transcript for videos by web scraping from TED.com

        transcript = get\_transcript(row["Link"])

        #print(transcript)

        record['transcript'] = transcript

        #write transcript into new csv file

        with open('d.csv', 'a', encoding='UTF8')as f:

            # create the csv writer

            writer = csv.writer(f)

            # write a row to the csv file

            writer.writerow([ record['id'], record['title'], record['author'], record['date'], record['views'], record['tags'], record['link'], record['transcript']])

        i += 1

        print(record)

**Execution:**

it is execution by type the following command in terminal(or command line):

**python .\** **webScraping.py**

The previous code is take each video url from **data.csv** file and get the transcript for it, and store the desired data in new file called **d.csv**.

Also this work done using web scraping technique.

**Classification ted.com videos depending on video transcript:**

The following images show the classification process we use machine learning, naive-bayes-classifier and Natural language processing (NLP).

# **Machine Learning**

Is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that makes it more similar to humans: **The ability to learn.** Machine learning is actively being used today, perhaps in many more places than one would expect.

**What is Naive Bayes Classifier?**

Naive Bayes is a **statistical classification technique based on Bayes Theorem**. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

**What is natural language processing?**

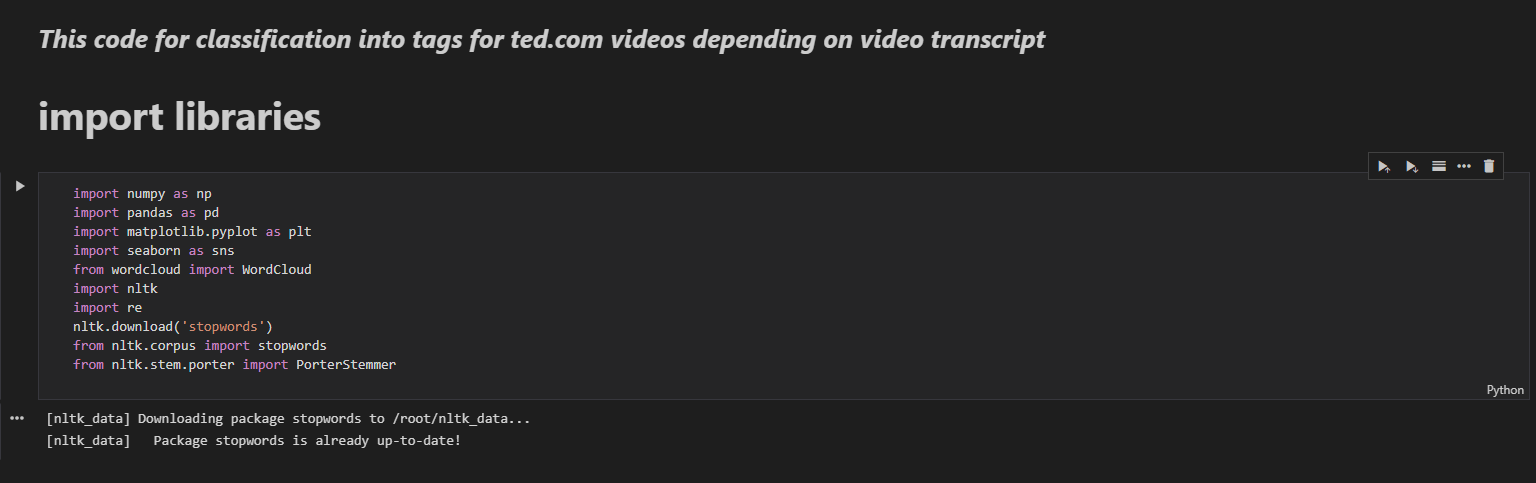
**Natural language processing (NLP)** refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

As we will show in the below example in classification process which used in our project in **text\_classification.ipynb** file:

**Note**: to run **text\_classification.ipynb** file I think you need to go <https://colab.research.google.com/> and upload **text\_classification.ipynb** file with **d.cvs** file to this site, then run it, you don’t need to install any library on colab.

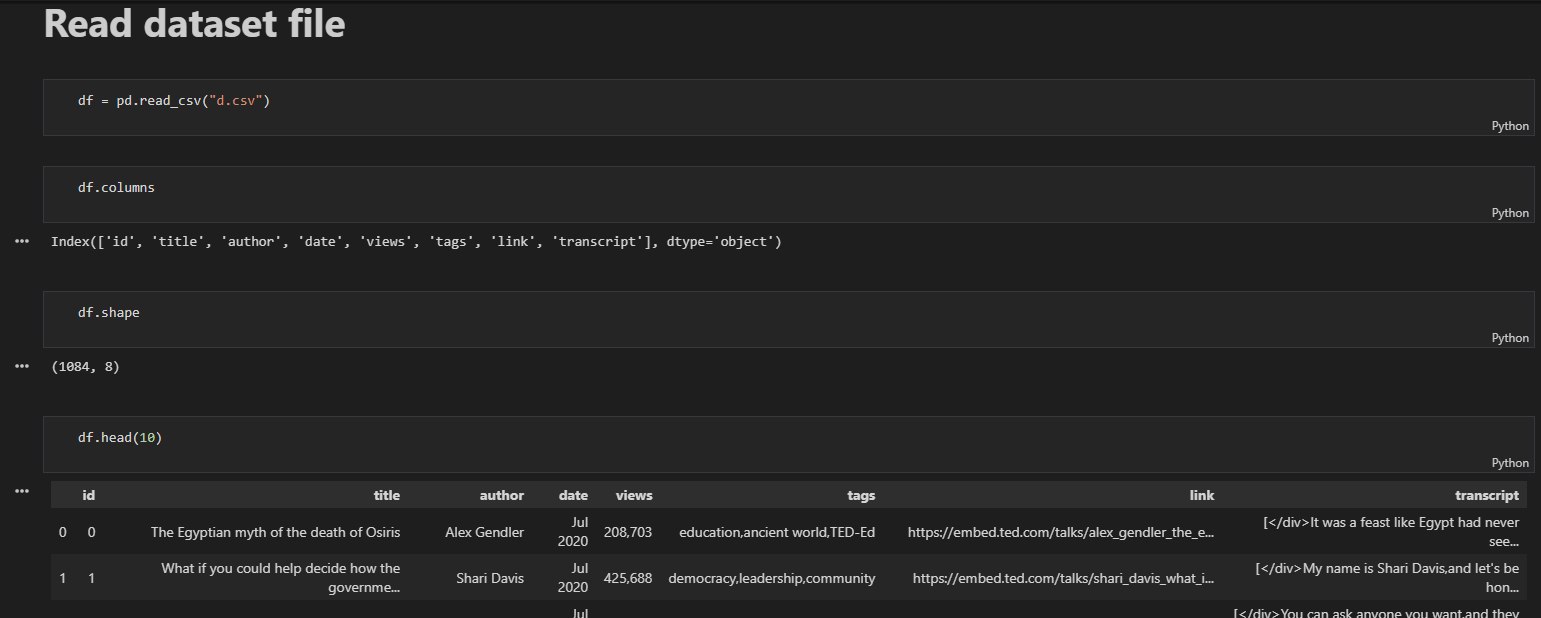
For this section open the **text\_classification.ipynb** in colab or jupyter and you will see the execution like in images bellow.

Here in the figure 2 we import the needed libraries that will use:



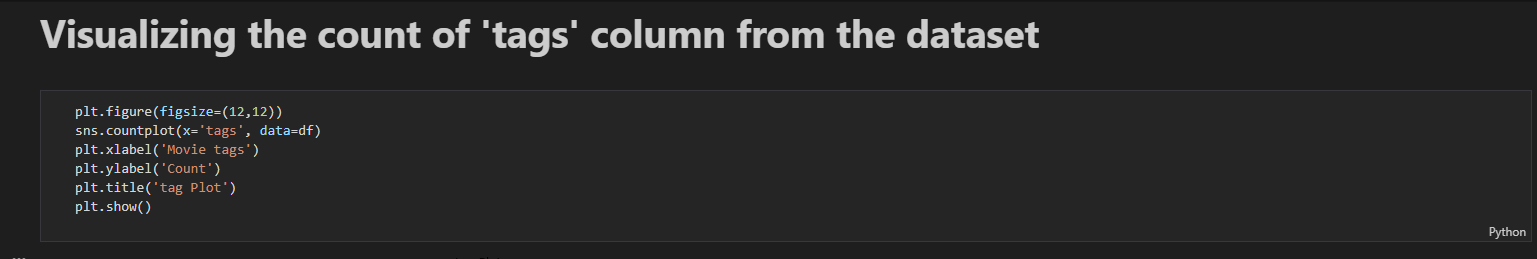
**Figure 1**

In the figure 3, we will read our dataset “d.csv” file:



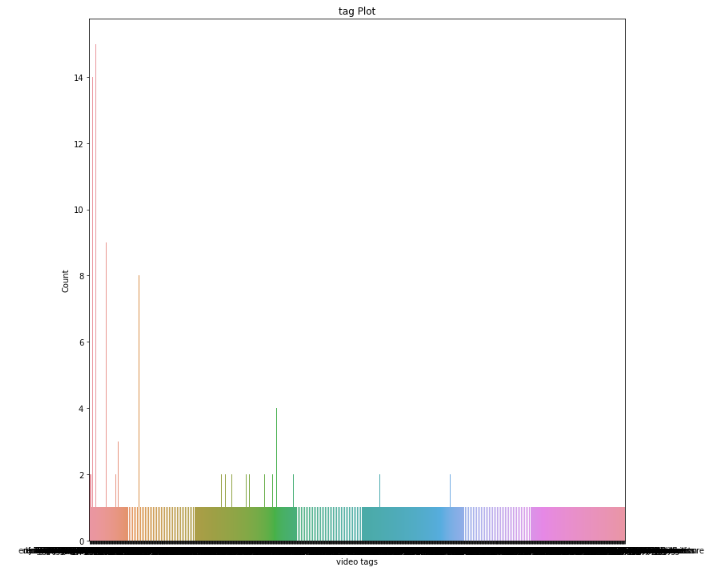
**Figure 2**

**Here, we visualizing the number of “tags” column:**



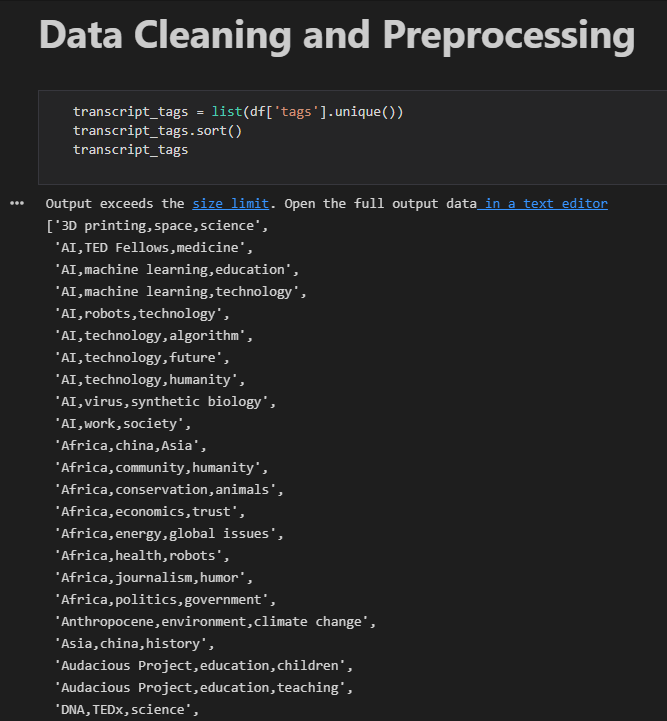
**Figure 3**

**The result of the previous code:**



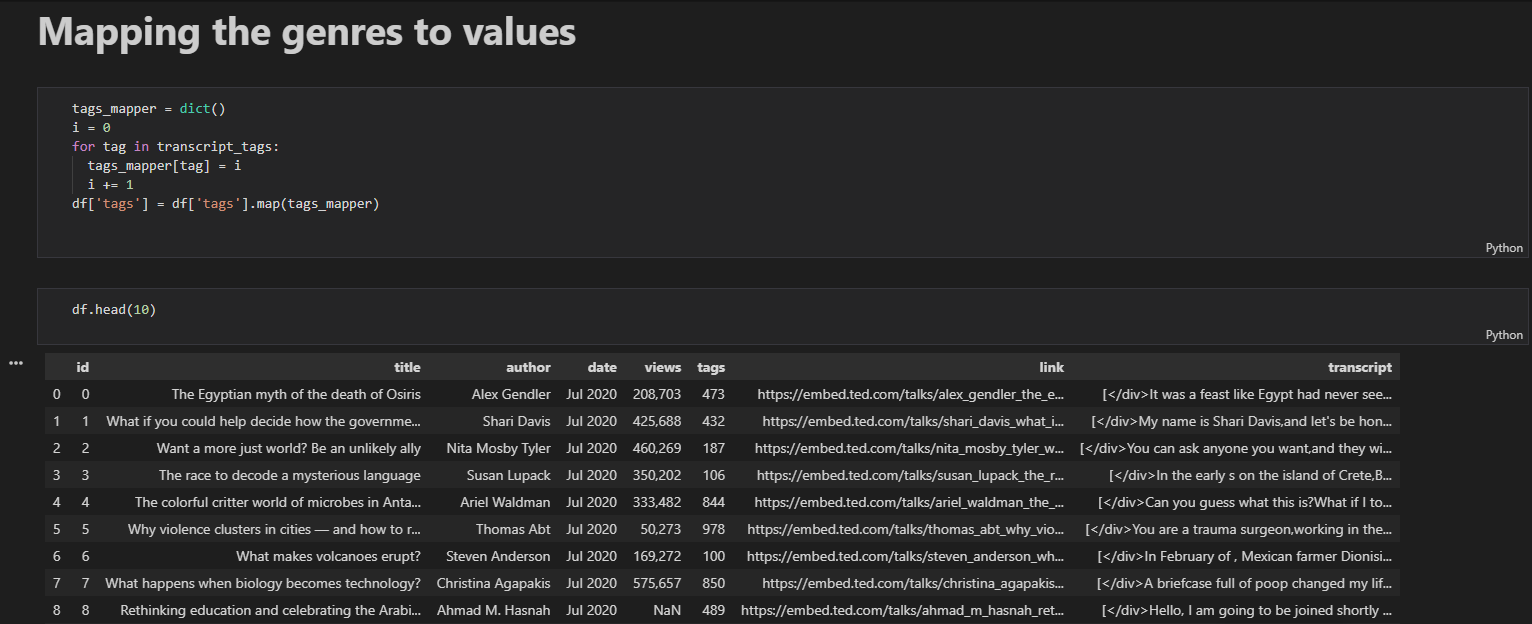
**Figure 4**

**Here, will preprocessing data to format it as we need for working on it:**



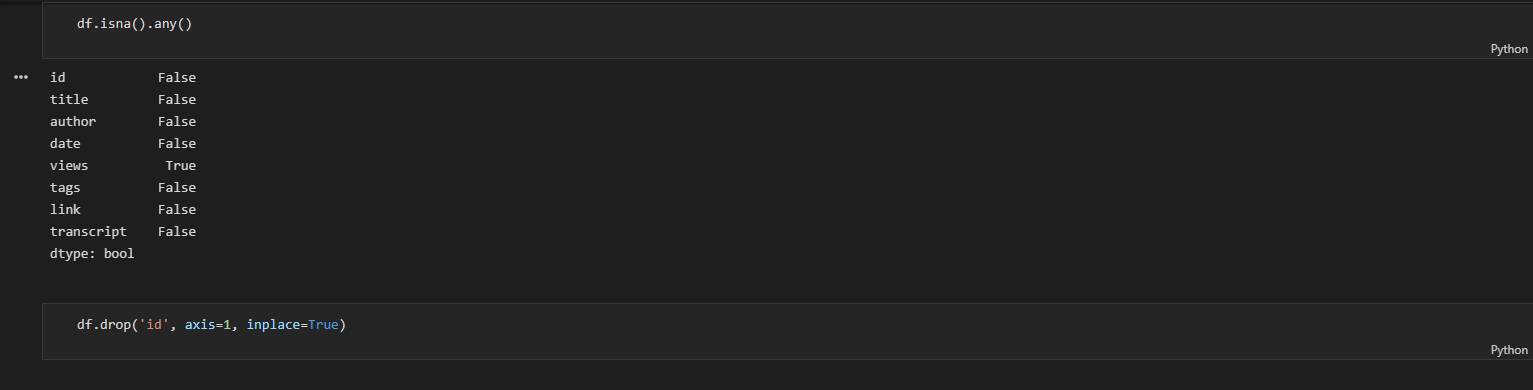
**Figure 5**

Will mapping the tags to values for appropriate form:



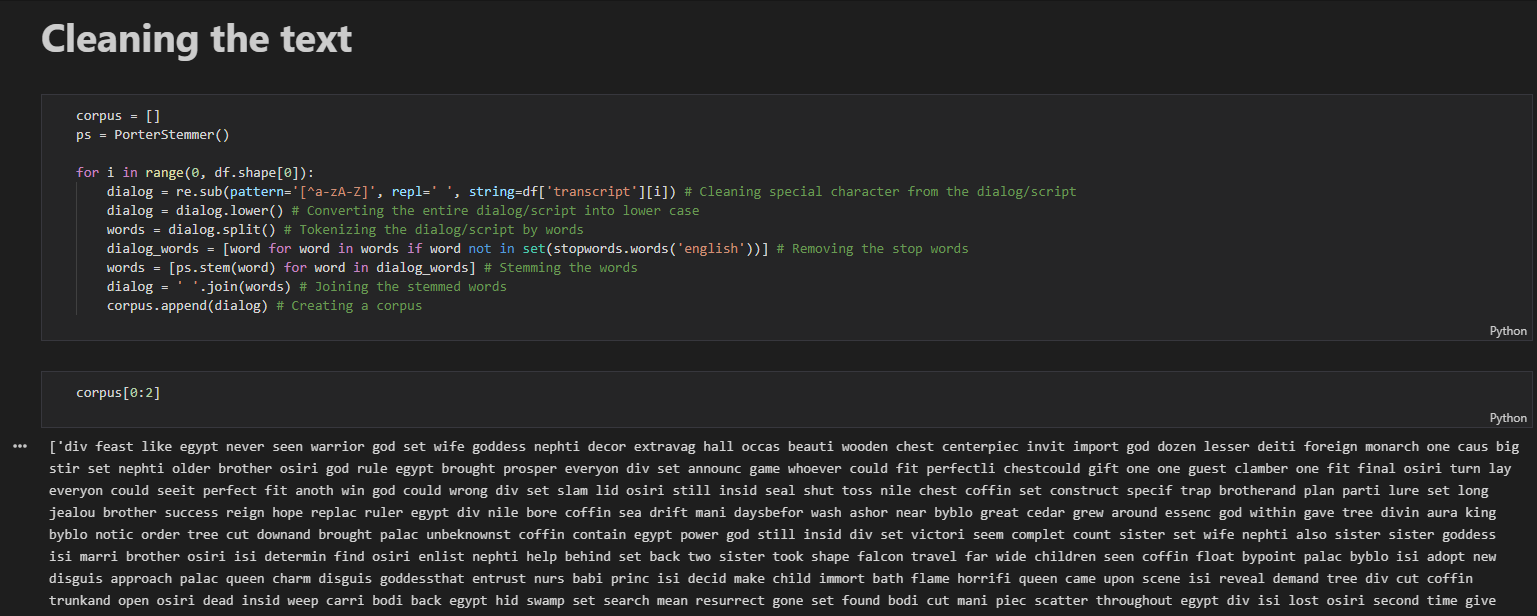
**Figure 6**

**Here will detect missing values, df.isna().any(): It return a boolean same-sized object indicating if the** values**are** NA. NA values, such as None or numpy.NaN, gets mapped to True values. Everything else gets mapped to False values:



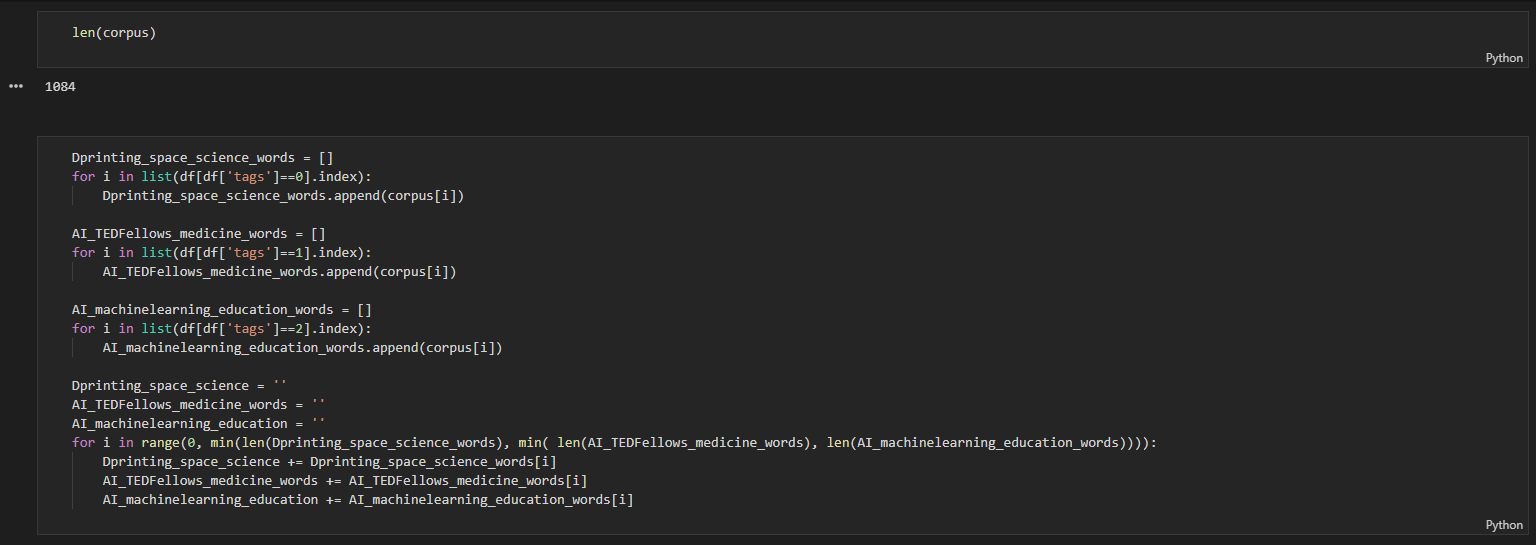
**Figure 7**

Here will cleaning the text in transcript field (remove the symbols and words like: “the, a, an, is, etc….”):



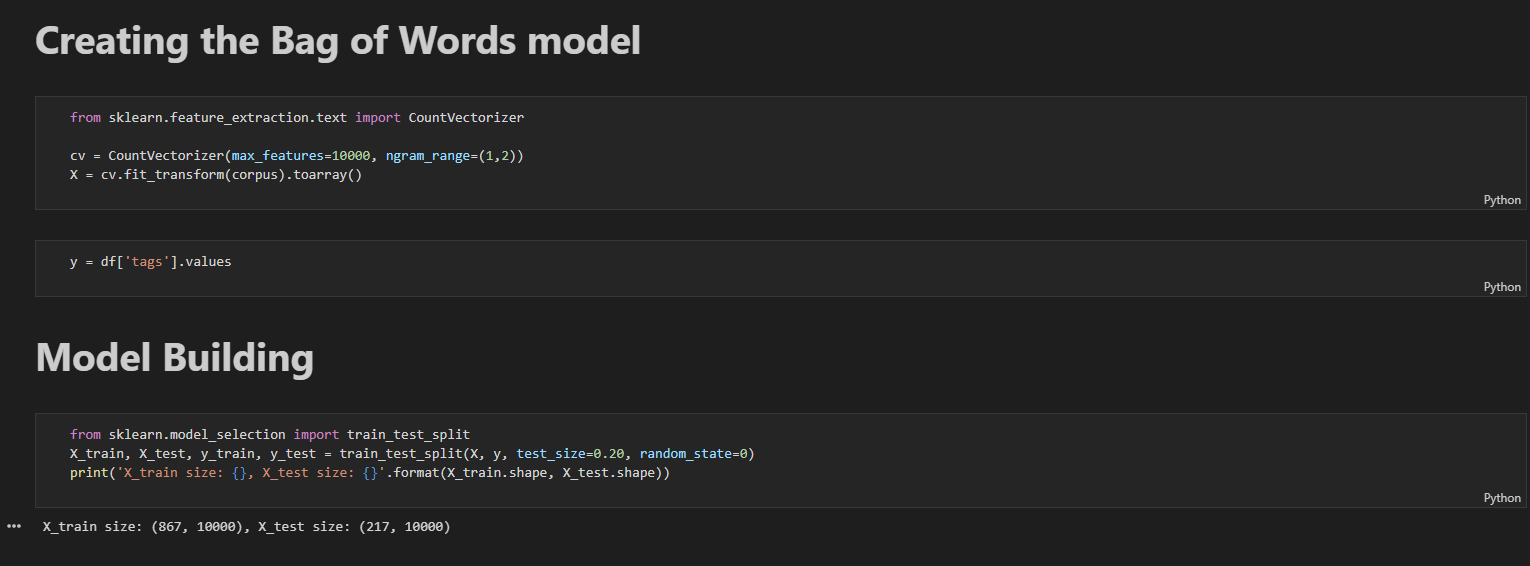
**Figure 8**

Create the arrays for testing it later:



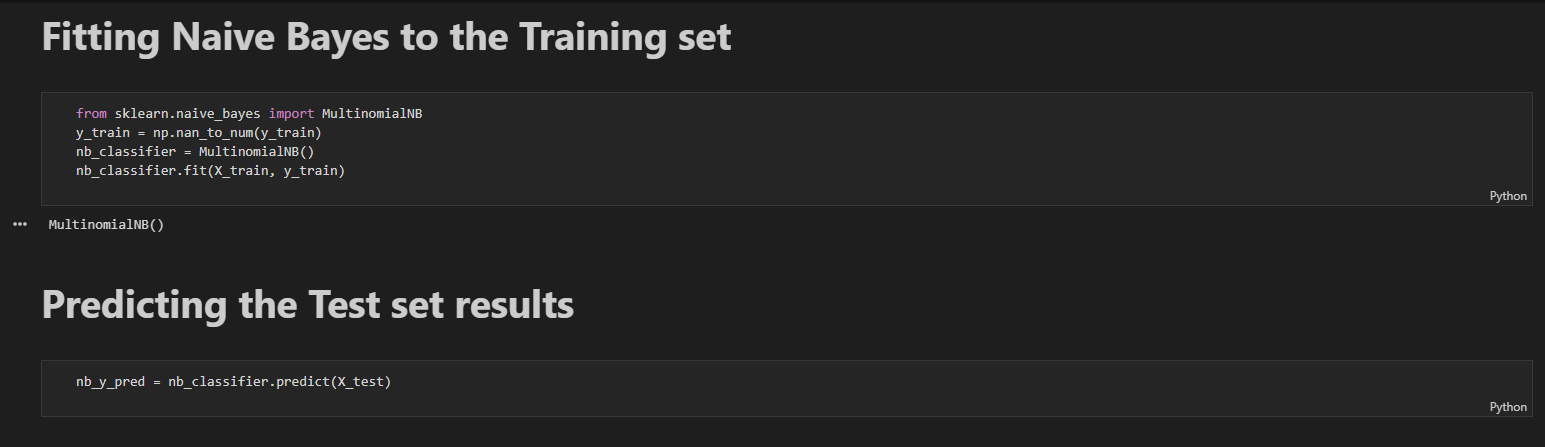
**Figure 9**

Creating our model which will use it for prediction later:



**Figure 10**

Training our model for make it able to prediction:

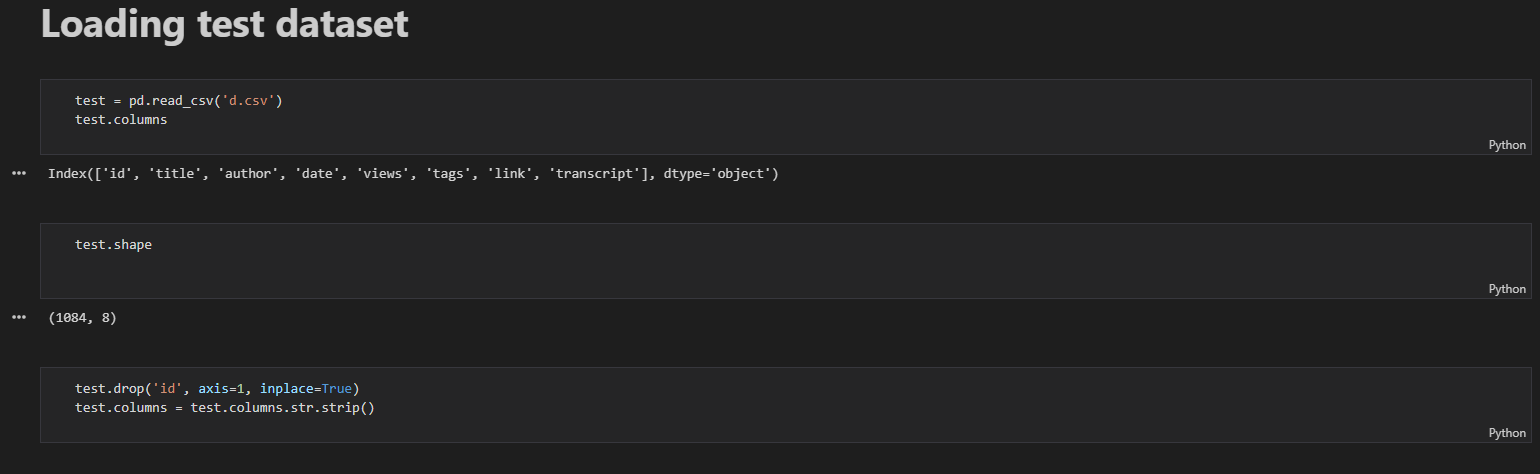


**Figure 11**

Calculating Accuracy for our model, and create prediction function:

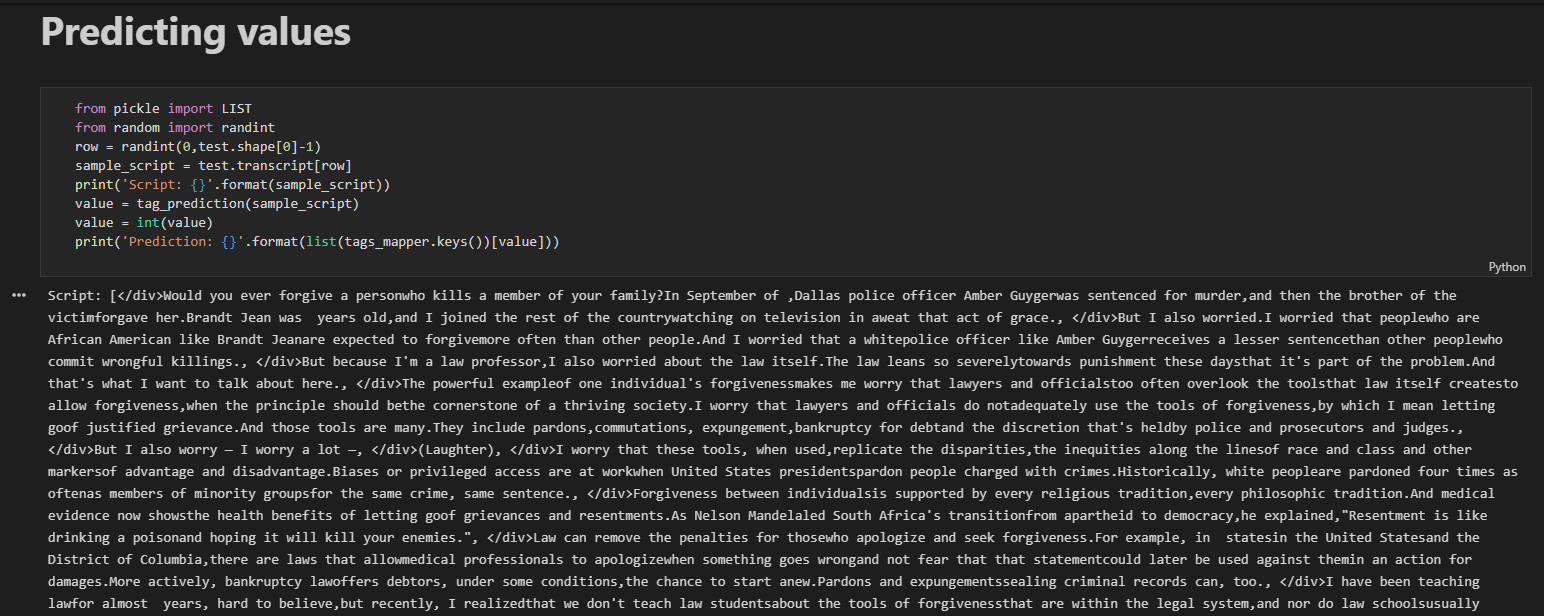


**Figure 12**



**Figure 13**

Make prediction for an example:



C:\Users\KHEDER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot_15.png

**Figure 15**