# Data Science Project: Scraping YouTube Data using Python and Selenium to Classify Videos

## Introduction

I’m an avid YouTube user. The sheer amount of content I can watch on a single platform is staggering. In fact, a lot of my [data science](https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2?utm_source=blog&utm_medium=scraping-classifying-youtube-video-data-python-selenium) learning has happened through YouTube videos!

So, I was browsing YouTube a few weeks ago searching for a certain category to watch. That’s when my data scientist thought process kicked in. Given my love for web scraping and machine learning, could I extract data about YouTube videos and build a model to classify them into their respective categories?

I was intrigued! This sounded like the perfect opportunity to combine my existing Python and data science knowledge with my curiosity to learn something new. And Analytics Vidhya’s internship challenge offered me the chance to pen down my learning in article form.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/05/youtube-data-scraping.png)

Web scraping is a skill I feel every data science enthusiast should know. It is immensely helpful when we’re looking for data for our project or want to analyze specific data present only on a website. Keep in mind though, web scraping should not cross ethical and legal boundaries.

In this article, we’ll learn how to use web scraping to extract YouTube video data using Selenium and Python. We will then use the NLTK library to clean the data and then build a model to classify these videos based on specific categories.

You can also check out the below tutorials on web scraping using different libraries:

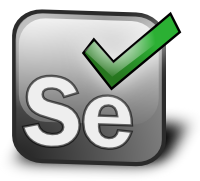
* [Beginner’s guide to Web Scraping in Python (using BeautifulSoup)](https://www.analyticsvidhya.com/blog/2015/10/beginner-guide-web-scraping-beautiful-soup-python/?utm_source=blog&utm_medium=scraping-classifying-youtube-video-data-python-selenium)
* [Web Scraping in Python using Scrapy (with multiple examples)](https://www.analyticsvidhya.com/blog/2017/07/web-scraping-in-python-using-scrapy/?utm_source=blog&utm_medium=scraping-classifying-youtube-video-data-python-selenium)
* [Beginner’s Guide on Web Scraping in R (using rest)](https://www.analyticsvidhya.com/blog/2017/07/web-scraping-in-python-using-scrapy/?utm_source=blog&utm_medium=scraping-classifying-youtube-video-data-python-selenium)

Note: BeautifulSoup is another library for web scraping. You can learn about this using our free course- [Introduction to Web Scraping using Python](https://courses.analyticsvidhya.com/courses/introduction-to-web-scraping?utm_source=blog&utm_medium=webscraping_selenium).

## Table of Contents

1. Overview of Selenium
2. Prerequisites for our Web Scraping Project
3. Setting up the Python Environment
4. Scraping Data from YouTube
5. Cleaning the Scraped Data using the NLTK Library
6. Building our Model to Classify YouTube Videos
7. Analyzing the Results

## Overview of Selenium

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/05/selenium.png)

Selenium is a popular tool for automating browsers. It’s primarily used for testing in the industry but is also very handy for web scraping. You must have come across Selenium if you’ve worked in the IT field.

We can easily program a Python script to automate a web browser using Selenium. It gives us the freedom we need to efficiently extract the data and store it in our preferred format for future use.

Selenium requires a driver to interface with our chosen browser. Chrome, for example, requires ChromeDriver, which needs to be installed before we start scraping. The Selenium web driver speaks directly to the browser using the browser’s own engine to control it. This makes it incredibly fast.

## Prerequisites for our Web Scraping Project

There are a few things we must know before jumping into web scraping:

* Basic knowledge of **HTML** and **CSS** is a must. We need this to understand the structure of a webpage we’re about to scrape
* **Python** is required to clean the data, explore it, and build models
* Knowledge of some basic libraries like **Pandas**and **NumPy** would be the cherry on the cake

## Setting up the Python Environment

Time to power up your favorite Python IDE (that’s Jupyter notebooks for me)! Let’s get our hands dirty and start coding.

**Step 1:** Install Python binding:

#Open terminal and type-

**$ pip install selenium**

**Step 2:** Download Chrome WebDriver:

* Visit [https://sites.google.com/a/chromium.org/chromedriver/download](https://sites.google.com/a/chromium.org/chromedriver/downloads)
* Select the compatible driver for your Chrome version
* To check the Chrome version you are using, click on the three vertical dots on the top right corner
* Then go to Help -> About Google Chrome

**Step 3:** Move the driver file to a PATH:

Go to the downloads directory, unzip the file, and move it to usr/local/bin PATH.

**$ cd Downloads**

**$ unzip chromedriver\_linux64.zip**

**$ mv chromedriver /usr/local/bin/**

We’re all set to begin web scraping now.

## Scraping Data from YouTube

In this article, we’ll be scraping the video ID, video title, and video description of a particular category from YouTube. The categories we’ll be scraping are:

* Travel
* Science
* Food
* History
* Manufacturing
* Art & Dance

So let’s begin!

* First, let’s import some libraries:

|  |  |
| --- | --- |
|  | from selenium import webdriver |
|  | import pandas as pd |
|  | from selenium.webdriver.common.by import By |
|  | from selenium.webdriver.support.ui import WebDriverWait |
|  | from selenium.webdriver.support import expected\_conditions as EC |

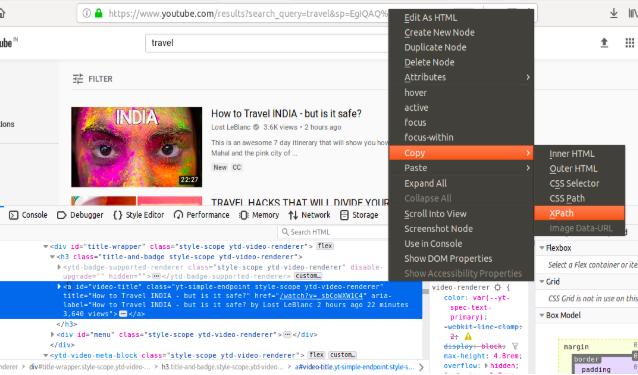
[**view raw**](https://gist.github.com/shubham-singh-ss/2beb917f4987ad9887a8866a7f06e899/raw/a45995963bed006af6098bd27683c459769f333a/gistfile1.py)[**gistfile1.py**](https://gist.github.com/shubham-singh-ss/2beb917f4987ad9887a8866a7f06e899#file-gistfile1-py) hosted with ❤ by [**GitHub**](https://github.com/)

* Before we do anything else, open YouTube in your browser. Type in the category you want to search videos for and set the filter to “videos”. This will display only the videos related to your search. Copy the URL after doing this.
* Next, we need to set up the driver to fetch the content of the URL from YouTube:

|  |  |
| --- | --- |
|  | driver = webdriver.Chrome() |
|  | driver.get("YOUR\_LINK\_HERE") |

[**view raw**](https://gist.github.com/shubham-singh-ss/00596c8768b9b2a0d5922364325f0d78/raw/1bb0b1d32dc5b77d69de1c286fa23dfdcc773831/driver.py)[**driver.py**](https://gist.github.com/shubham-singh-ss/00596c8768b9b2a0d5922364325f0d78#file-driver-py) hosted with ❤ by [**GitHub**](https://github.com/)

* Paste the link into to driver.get(“ Your Link Here ”) function and run the cell. This will open a new browser window for that link. We will do all the following tasks in this browser window
* Fetch all the video links present on that particular page. We will create a “list” to store those links
* Now, go to the browser window, right-click on the page, and select ‘inspect element’
* Search for the anchor tag with id = ”video-title” and then right-click on it -> Copy -> XPath. The XPath should look something like : **//\*[@id=”video-title”]**

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/05/Web-Scraping-selenium.png)

With me so far? Now, write the below code to start fetching the links from the page and run the cell. This should fetch all the links present on the web page and store it in a list.

**Note: Traverse all the way down to load all the videos on that page.**

|  |  |
| --- | --- |
|  | user\_data = driver.find\_elements\_by\_xpath('//\*[@id="video-title"]') |
|  | links = [] |
|  | for i in user\_data: |
|  | links.append(i.get\_attribute('href')) |
|  |  |
|  | print(len(links)) |

[**view raw**](https://gist.github.com/shubham-singh-ss/fb89f5bf299f143dc3dd0efb537d79c9/raw/9b44187746f3c4051e281a3cc4f51dc3e2b1ecde/fetch_links.py)[**fetch\_links.py**](https://gist.github.com/shubham-singh-ss/fb89f5bf299f143dc3dd0efb537d79c9#file-fetch_links-py) hosted with ❤ by [**GitHub**](https://github.com/)

The above code will fetch the “href” attribute of the anchor tag we searched for.

Now, we need to create a dataframe with 4 columns – “**link**”, “**title**”, “**description**”, and “**category**”. We will store the details of videos for different categories in these columns:

|  |  |
| --- | --- |
|  | df = pd.DataFrame(columns = ['link', 'title', 'description', 'category']) |

[**view raw**](https://gist.github.com/shubham-singh-ss/e16843279e47d77e3ea8dae456ec18b9/raw/00c64a62e6a3b60381038c0555ee4580415029eb/df.py)[**df.py**](https://gist.github.com/shubham-singh-ss/e16843279e47d77e3ea8dae456ec18b9#file-df-py) hosted with ❤ by [**GitHub**](https://github.com/)

We are all set to scrape the video details from YouTube. Here’s the Python code to do it:

|  |  |
| --- | --- |
|  | wait = WebDriverWait(driver, 10) |
|  | v\_category = "CATEGORY\_NAME" |
|  | for x in links: |
|  | driver.get(x) |
|  | v\_id = x.strip('https://www.youtube.com/watch?v=') |
|  | v\_title = wait.until(EC.presence\_of\_element\_located( |
|  | (By.CSS\_SELECTOR,"h1.title yt-formatted-string"))).text |
|  | v\_description =  wait.until(EC.presence\_of\_element\_located( |
|  | (By.CSS\_SELECTOR,"div#description |
|  | yt-formatted-string"))).text |
|  | df.loc[len(df)] = [v\_id, v\_title, v\_description, v\_category] |

[**view raw**](https://gist.github.com/shubham-singh-ss/e3a33116b7055246d8c44d50c52d866c/raw/32eb3da93eaef54b14aacc2daab9f912afd571e1/dataframe.py)[**dataframe.py**](https://gist.github.com/shubham-singh-ss/e3a33116b7055246d8c44d50c52d866c#file-dataframe-py) hosted with ❤ by [**GitHub**](https://github.com/)

Let’s breakdown this code block to understand what we just did:

* **“wait”** will ignore instances of NotFoundException that are encountered (thrown) by default in the ‘until’ condition. It will immediately propagate all others
* Parameters:
  + **driver:** The WebDriver instance to pass to the expected conditions
  + **timeOutInSeconds:** The timeout in seconds when an expectation is called
* **v\_category** stores the video category name we searched for earlier
* The “for” loop is applied on the list of links we created above
* **driver.get(x)** traverses through all the links one-by-one and opens them in the browser to fetch the details
* **v\_id** stores the stripped video ID from the link
* **v\_title** stores the video title fetched by using the CSS path
* Similarly, **v\_description** stores the video description by using the CSS path

During each iteration, our code saves the extracted data inside the dataframe we created earlier.

We have to follow the aforementioned steps for the remaining five categories. We should have six different dataframes once we are done with this. Now, it’s time to merge them together into a single dataframe:

|  |  |
| --- | --- |
|  | frames = [df\_travel, df\_science, df\_food, df\_manufacturing, df\_history, df\_artndance] |
|  | df\_copy = pd.concat(frames, axis=0, join='outer', join\_axes=None, ignore\_index=True, |
|  | keys=None, levels=None, names=None, verify\_integrity=False, copy=True) |

[**view raw**](https://gist.github.com/shubham-singh-ss/add2ccb4aa87aeb9197a9180b17cf737/raw/64f32ee0a2265eccfcc2c9a6f376f2960680715e/concat.py)[**concat.py**](https://gist.github.com/shubham-singh-ss/add2ccb4aa87aeb9197a9180b17cf737#file-concat-py) hosted with ❤ by [**GitHub**](https://github.com/)

Voila! We have our final dataframe containing all the desired details of a video from all the categories mentioned above.

## Cleaning the Scraped Data using the NLTK Library

In this section, we’ll use the popular NLTK library to clean the data present in the “title” and “description” columns. NLP enthusiasts will love this section!

Before we start cleaning the data, we need to store all the columns separately so that we can perform different operations quickly and easily:

|  |  |
| --- | --- |
|  | df\_link = pd.DataFrame(columns = ["link"]) |
|  | df\_title = pd.DataFrame(columns = ["title"]) |
|  | df\_description = pd.DataFrame(columns = ["description"]) |
|  | df\_category = pd.DataFrame(columns = ["category"]) |
|  | df\_link[‘link’] = df\_copy['link'] |
|  | df\_title [‘title’]= df\_copy['title'] |
|  | df\_description[‘description’] = df\_copy['description'] |
|  | df\_category[‘category’] = df\_copy['category'] |

[**view raw**](https://gist.github.com/shubham-singh-ss/ee58ee2048901088e7200c4548a8d2eb/raw/f125c96dd152f4864cadaa8cb5bfeaf0e75073bb/separating.py)[**separating.py**](https://gist.github.com/shubham-singh-ss/ee58ee2048901088e7200c4548a8d2eb#file-separating-py) hosted with ❤ by [**GitHub**](https://github.com/)

Import the required libraries first:

|  |  |
| --- | --- |
|  | import re |
|  | import nltk |
|  | nltk.download('stopwords') |
|  | from nltk.corpus import stopwords |
|  | from nltk.stem.porter import PorterStemmer |

[**view raw**](https://gist.github.com/shubham-singh-ss/c0cf4bfa6fb2cdf8a2bb34611f34bf45/raw/16cf8b80af11759b25d306ded160d683191e96e9/cleaning.py)[**cleaning.py**](https://gist.github.com/shubham-singh-ss/c0cf4bfa6fb2cdf8a2bb34611f34bf45#file-cleaning-py) hosted with ❤ by [**GitHub**](https://github.com/)

Now, create a list in which we can store our cleaned data. We will store this data in a dataframe later. Write the following code to create a list and do some data cleaning on the “**title**” column from **df\_title**:

|  |  |
| --- | --- |
|  | corpus = [] |
|  | for i in range(0, 8375): |
|  | review = re.sub('[^a-zA-Z]', ' ', df\_title['title'][i]) |
|  | review = review.lower() |
|  | review = review.split() |
|  | ps = PorterStemmer() |
|  | review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))] |
|  | review = ' '.join(review) |
|  | corpus.append(review) |

[**view raw**](https://gist.github.com/shubham-singh-ss/3979366a7113a685f8a4ecafa0679a99/raw/3cb1c41d1ad55863e4fa256c4b49b247df786289/corpus1.py)[**corpus1.py**](https://gist.github.com/shubham-singh-ss/3979366a7113a685f8a4ecafa0679a99#file-corpus1-py) hosted with ❤ by [**GitHub**](https://github.com/)

Did you see what we did here? We removed all the punctuation from the titles and only kept the English root words. After all these iterations, we are ready with our list full of data.

We need to follow the same steps to clean the **“description”** column from **df\_description:**

|  |  |
| --- | --- |
|  | corpus1 = [] |
|  | for i in range(0, 8375): |
|  | review = re.sub('[^a-zA-Z]', ' ', df\_description['description'][i]) |
|  | review = review.lower() |
|  | review = review.split() |
|  | ps = PorterStemmer() |
|  | review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))] |
|  | review = ' '.join(review) |
|  | corpus1.append(review) |

[**view raw**](https://gist.github.com/shubham-singh-ss/c9c531e711c5452ff1911bcdffadf424/raw/d8c88dd769bdc45724d881cb2d1e4603c4cd3b74/corpus2.py)[**corpus2.py**](https://gist.github.com/shubham-singh-ss/c9c531e711c5452ff1911bcdffadf424#file-corpus2-py) hosted with ❤ by [**GitHub**](https://github.com/)

Note: The range is selected as per the rows in our dataset.

Now, convert these lists into dataframes:

|  |  |
| --- | --- |
|  | dftitle = pd.DataFrame({'title':corpus}) |
|  | dfdescription = pd.DataFrame({'description':corpus1}) |

[**view raw**](https://gist.github.com/shubham-singh-ss/a28f247fefb65200925d9ef7dec1a51f/raw/c95b94d9e10931fb85a93995c1f124d52a045ee6/convert.py)[**convert.py**](https://gist.github.com/shubham-singh-ss/a28f247fefb65200925d9ef7dec1a51f#file-convert-py) hosted with ❤ by [**GitHub**](https://github.com/)

Next, we need to label encode the categories. The “LabelEncoder()” function encodes labels with a value between 0 and n\_classes – 1 where n is the number of distinct labels.

|  |  |
| --- | --- |
|  | from sklearn.preprocessing import LabelEncoder |
|  | dfcategory = df\_category.apply(LabelEncoder().fit\_transform) |

[**view raw**](https://gist.github.com/shubham-singh-ss/a8bb261f31d0e5e63bd4e4fc59f5666e/raw/1222b6a0b644330e310b52189fb9cd98a760f9c1/encoding.py)[**encoding.py**](https://gist.github.com/shubham-singh-ss/a8bb261f31d0e5e63bd4e4fc59f5666e#file-encoding-py) hosted with ❤ by [**GitHub**](https://github.com/)

Here, we have applied label encoding on df\_category and stored the result into dfcategory. We can store our cleaned and encoded data in into a new dataframe:

|  |  |
| --- | --- |
|  | df\_new = pd.concat([df\_link, dftitle, dfdescription, dfcategory], axis=1, join\_axes = [df\_link.index]) |

[**view raw**](https://gist.github.com/shubham-singh-ss/a46326360c4b0adbdd16ef35910347e8/raw/063d96105dd3a98bb196961d3f2cd0f6e9490e41/new_dataframe.py)[**new\_dataframe.py**](https://gist.github.com/shubham-singh-ss/a46326360c4b0adbdd16ef35910347e8#file-new_dataframe-py) hosted with ❤ by [**GitHub**](https://github.com/)

We’re not quite all the way done with our cleaning and transformation part.

We should create a bag-of-words so that our model can understand the keywords from that bag to classify videos accordingly. Here’s the code to do create a bag-of-words:

|  |  |
| --- | --- |
|  | from sklearn.feature\_extraction.text import CountVectorizer |
|  | cv = CountVectorizer(max\_features = 1500) |
|  | X = cv.fit\_transform(corpus, corpus1).toarray() |
|  | y = df\_new.iloc[:, 3].values |

[**view raw**](https://gist.github.com/shubham-singh-ss/fae2a0032db371d76fffe9ff75374fba/raw/260e903bb14de2334c81c07f6fe4a83655e5ca2a/bag_of_words.py)[**bag\_of\_words.py**](https://gist.github.com/shubham-singh-ss/fae2a0032db371d76fffe9ff75374fba#file-bag_of_words-py) hosted with ❤ by [**GitHub**](https://github.com/)

Note: Here, we created 1500 features from data stored in the lists – corpus and corpus1. “X” stores all the features and “y” stores our encoded data.

We are all set for the most anticipated part of a data scientist’s role – model building!

## Building our Model to Classify YouTube Videos

Before we build our model, we need to divide the data into training set and test set:

* **Training set:**A subset of the data to train our model
* **Test set:** Contains the remaining data to test the trained model

Make sure that your test set meets the following two conditions:

* Large enough to yield statistically meaningful results
* Representative of the dataset as a whole. In other words, don’t pick a test set with different characteristics than the training set

We can use the following code to split the data:

|  |  |
| --- | --- |
|  | from sklearn.model\_selection import train\_test\_split |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0) |

[**view raw**](https://gist.github.com/shubham-singh-ss/5d0c1761ab33ac86ae8bdf27d9d950ce/raw/46cba5a475f7074400347a70f729fd452a8e9fb1/split.py)[**split.py**](https://gist.github.com/shubham-singh-ss/5d0c1761ab33ac86ae8bdf27d9d950ce#file-split-py) hosted with ❤ by [**GitHub**](https://github.com/)

Time to train the model! We will use the [random forest algorithm](https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/?utm_source=blog&utm_medium=scraping-classifying-youtube-video-data-python-selenium) here. So let’s go ahead and train the model using the RandomForestClassifier() function:

|  |  |
| --- | --- |
|  | from sklearn.ensemble import RandomForestClassifier |
|  | classifier = RandomForestClassifier(n\_estimators = 1000, criterion = 'entropy') |
|  | classifier.fit(X\_train, y\_train) |

[**view raw**](https://gist.github.com/shubham-singh-ss/838f274017a842ae8ef971f547f92833/raw/a936ff210626190ef8e29821f3403556883b9d52/model.py)[**model.py**](https://gist.github.com/shubham-singh-ss/838f274017a842ae8ef971f547f92833#file-model-py) hosted with ❤ by [**GitHub**](https://github.com/)

Parameters:

* **n\_estimators**: The number of trees in the forest
* **criterion**: The function to measure the quality of a split. Supported criteria are “gini” for Gini impurity and “entropy” for information gain

Note: These parameters are tree-specific.

We can now check the performance of our model on the test set:

|  |  |
| --- | --- |
|  | y\_pred = classifier.predict(X\_test) |
|  | classifier.score(X\_test, y\_test) |

[**view raw**](https://gist.github.com/shubham-singh-ss/988dab8ca52c378b36439cf668669acd/raw/4f71cad95e51c190e79e78402f9c1ac2ca9be236/result.py)[**result.py**](https://gist.github.com/shubham-singh-ss/988dab8ca52c378b36439cf668669acd#file-result-py) hosted with ❤ by [**GitHub**](https://github.com/)

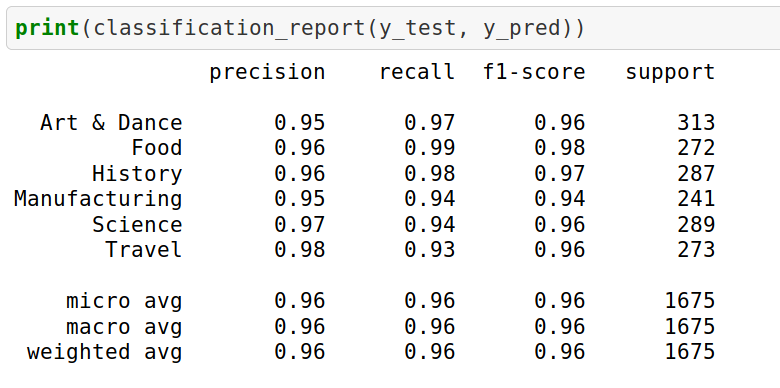
**We get an impressive 96.05% accuracy.** Our entire process went pretty smoothly! But we’re not done yet – we need to analyze our results as well to fully understand what we achieved.

## Analyzing the Results

Let’s check the classification report:

|  |  |
| --- | --- |
|  | print(classification\_report(y\_test, y\_pred)) |

[**view raw**](https://gist.github.com/shubham-singh-ss/6da87bd7bb77df81ab15c703ea666a6e/raw/470ffd3ddffcffea9c4fcb3b7b02a83cc9113d53/analyse.py)[**analyse.py**](https://gist.github.com/shubham-singh-ss/6da87bd7bb77df81ab15c703ea666a6e#file-analyse-py) hosted with ❤ by [**GitHub**](https://github.com/)

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/05/analysis.png)

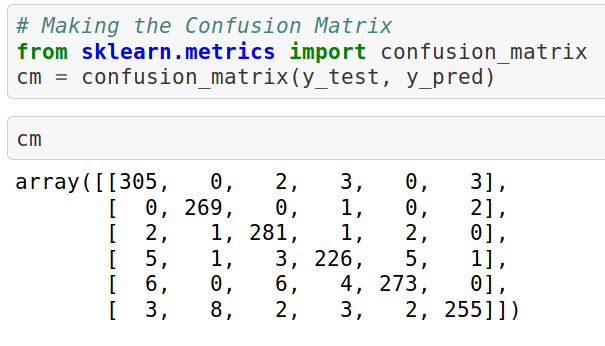
The result will give the following attributes:

* **Precision** is the ratio of correctly predicted positive observations to the total predicted positive observations. Precision = TP/TP+FP
* **Recall**is the ratio of correctly predicted positive observations to all the observations in the actual class. Recall = TP/TP+FN
* **F1 Score** is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. F1 Score = 2\*(Recall \* Precision) / (Recall + Precision)

We can check our results by creating a confusion matrix as well:

|  |  |
| --- | --- |
|  | from sklearn.metrics import confusion\_matrix |
|  | cm = confusion\_matrix(y\_test, y\_pred) |
|  | cm |

[**view raw**](https://gist.github.com/shubham-singh-ss/4f40a9850bccac1026cd40dba51fde2d/raw/eaf0981003e52940f467bb08469d1f38f081394b/confusion_matrix.py)[**confusion\_matrix.py**](https://gist.github.com/shubham-singh-ss/4f40a9850bccac1026cd40dba51fde2d#file-confusion_matrix-py) hosted with ❤ by [**GitHub**](https://github.com/)

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/05/matrix.png)

The confusion matrix will be a 6×6 matrix since we have six classes in our dataset.

## End Notes

I’ve always wanted to combine my interest in scraping and extracting data with NLP and machine learning. So I loved immersing myself in this project and penning down my approach.