

# Data-Driven Methods for PE and VC

Neirouz Bouchaira<sup>†</sup>

**Abstract** This project aims to extract and enrich data from the 'Our Startups' section of the Plug and Play Tech Center website. The objective is to identify relevant startups for potential investment by gathering information such as company name, website URL, and description. Additionally, external sources will be used to collect data such as LinkedIn URL, founder information, employee count, employee growth, headquarters country, and founding year. The project will categorize companies based on software/IT services, hardware, B2B/B2C focus, vertical/horizontal software, and AI capabilities. The output will be a well-structured CSV file and Python code for easy comprehension and reproducibility.

**Keywords:** Extract; Enrich; Startups; Data; Web scraping; External sources; Categorize; ML

## 1. Introduction

Private equity (PE) and venture capital (VC) firms play a crucial role in funding and supporting startups and emerging companies. The success of PE and VC investments relies heavily on identifying promising startups with strong growth potential. However, the traditional methods of sourcing and evaluating startups can be time-consuming, manual, and often limited in scope. This is where the utilization of web scraping and data enrichment techniques can provide significant advantages for PE and VC firms.

### 1.1 Benefits for Fortino Capital

- (i) **Efficient Deal Sourcing:** Web scraping enables firms to extract a large amount of data from online sources, saving time and allowing them to review more potential investment opportunities.
- (ii) **Comprehensive Data Collection:** By combining web scraping with external sources, firms can gather extensive information about startups, including team backgrounds, growth trends, market positioning, and financial metrics. Enriched data provides a deeper understanding of the startup's potential.
- (iii) **Data-Driven Decision Making:** Extracting and analyzing data from multiple sources enables firms to make more informed investment decisions, reducing reliance on subjective evaluations.

---

Submitted on June 3rd 2023

<sup>†</sup>[neirouz.bouchaira.pro@gmail.com](mailto:neirouz.bouchaira.pro@gmail.com)

- (iv) **Improved Due Diligence:** Web scraping and data enrichment support thorough due diligence by providing information on management teams, industry relationships, customer base, and technological capabilities.
- (v) **Enhanced Portfolio Management:** The extracted and enriched data facilitates ongoing monitoring and management of the firm's portfolio, tracking performance, identifying growth patterns, and making strategic decisions.

## 2. Data Extraction

### 2.1 Libraries

The code includes the necessary import statements for the **requests** library, used for making HTTP requests, and the **BeautifulSoup** module from the **bs4** package, used for parsing HTML content. These libraries are essential for implementing web scraping functionality and extracting data from the website.

```
1 import requests
2 from bs4 import BeautifulSoup
```

### 2.2 Methodology of Extraction

The methodology for data extraction in this project involves utilizing web scraping techniques to gather information from the 'Our Startups'<sup>1</sup>. The code begins by defining a list of user agents, which are used to randomize the requests and **prevent detection or blocking**. A GET request is then sent to the target URL with a randomly selected user agent from the list. The response from the request is obtained, which contains the HTML content of the webpage.

```
1 # Set a list of user agents
2 user_agents = [
3     'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
4     AppleWebKit/537.36 (KHTML, like Gecko) Chrome
5     /90.0.4430.212 Safari/537.36',
6     'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
7     AppleWebKit/537.36 (KHTML, like Gecko) Chrome
8     /91.0.4472.124 Safari/537.36',
9     'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
10    AppleWebKit/537.36 (KHTML, like Gecko) Chrome
11    /92.0.4515.107 Safari/537.36',
```

<sup>1</sup>Section of the Plug and Play Tech Center website.

```

6     ]
7
8     # Send a GET request to the website with random user
      agent and delay
9     url = "https://www.plugandplaytechcenter.com/startups/
      our-startups/"
10    headers = {'User-Agent': user_agents[randint(0, len(
      user_agents) - 1)]}
11    response = requests.get(url, headers=headers)

```

The HTML content of the webpage is parsed using the BeautifulSoup library and stored in the soup object. The prettify() method is then used to format the HTML code for improved readability. Printing the soup object allows for inspecting the parsed HTML and verifying the successful retrieval of the webpage content.

```

1 soup = BeautifulSoup(response.text, 'html.parser')
2 print(soup.prettify())

```

The code snippet uses the find\_all() method of the soup object to extract specific elements from the parsed HTML. **The classes names used in the code were obtained by inspecting the website page manually.** The extracted elements include startup names, website links, and descriptions.

```

1 startups = soup.find_all('div', class_="pnp__startup-
      item-name")
2 links=soup.find_all('a', class_="pnp__startup-website-
      link")
3 descriptions=soup.find_all('div', class_="cell
      pnp__startup-description")

```

### 2.3 Pre-processing

The name of the start-up is extracted like the following structure: '<div class="pnp\_\_startup-item-name">Trulioo</div>'. Moreover, some startups are extracted twice (from the webpage and the pop-up of each company description). **Pre-processing is, as a consequence, necessary.**

The following code snippet iterates over the startups list and extracts the text content of each element. It filters out duplicate startup names and names that end with an ellipsis before adding them to the startups\_list. This process ensures that the final list contains unique and complete startup names extracted from the webpage.

```

1 startups_list = []
2 for startup in startups:

```

```

3     #keep only the text between the > and < characters
4
5     startup = startup.text
6     if startup not in startups_list and not(startup.
7     endswith('...')):
8         startups_list.append(startup)

```

The code snippet creates a list (`links_list`) to store the extracted website links by iterating over the links list and using the `get('href')` method to retrieve the URLs. The code also creates a list (`descriptions_list`) to store the extracted descriptions by iterating over the descriptions list, extracting the text content, and removing newline characters.

```

1
2 links_list=[]
3 # create a list of all the links
4 for link in links:
5     links_list.append(link.get('href'))
6
7 print(links_list)
8
9 descriptions_list=[]
10 for description in descriptions:
11     #keep only the text between the > and < characters
12
13     description = description.text
14     #remove the \n and \t characters
15     description = description.replace('\n', '')
16     descriptions_list.append(description)

```

**Remark 1.** A dozen of spaces are in the beginning of each 'Description paragraph'. Those spaces were not removed because they don't intervene with the understanding of the data.

## 2.4 Results

The final result of this phase is a **CSV file**.

The code snippet creates a Pandas DataFrame (`dataframe`) from the extracted data and exports it to a CSV file named 'startups.csv'. The DataFrame consists of columns for startup names, website links, and descriptions.

```

1 import pandas as pd
2
3 data={'startups':startups_list,'links':links_list,'
4     descriptions':descriptions_list}

```

```

4 dataframe=pd.DataFrame(data)
5 dataframe.head()
6 dataframe.to_csv('startups.csv', index=False)

```

### 3. Data Enrichment

#### 3.1 LinkedIn link

To enrich the data, a function is created to extract the LinkedIn links of each company. This function opens the website of each company, extracts all links from the webpage, and filters out only the linkedIn links <sup>2</sup>.

```

1 def linkedin(link):
2     user_agents = [
3         'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
4         AppleWebKit/537.36 (KHTML, like Gecko) Chrome
5         /90.0.4430.212 Safari/537.36',
6         'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
7         AppleWebKit/537.36 (KHTML, like Gecko) Chrome
8         /91.0.4472.124 Safari/537.36',
9         'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
10        AppleWebKit/537.36 (KHTML, like Gecko) Chrome
11        /92.0.4515.107 Safari/537.36',]
12
13    # Send a GET request to the website with random user
14    # agent and delay
15    url = link
16    headers = {'User-Agent': user_agents[randint(0, len(
17    user_agents) - 1)]}
18    response = requests.get(url, headers=headers)
19    soup = BeautifulSoup(response.text, 'html.parser')
20    #import a list of links from the website
21    links = [a['href'] for a in soup.find_all('a', href=
22    True)]
23    linkedin_link = [link for link in links if 'linkedin
24    ' in link]
25
26    return linkedin_link

```

An example of the output of the function is the Figure 1:

<sup>2</sup>only the first linkedin link is going to be kept.

```

linkedin('http://coursehero.com')
✓ 0.9s
['https://www.linkedin.com/company/coursehero']

```

Figure 1

The CSV file is then completed by a LinkedIn column that is made using that function.

**Remark 2.** It is not guaranteed that a company puts its LinkedIn link on its website. For that case, a link following the template 'https://www.linkedin.com/company/'+name' is added.

```

1 for i in range(startups.shape[0]):
2     try:
3         startups['linkedin'][i] = linkedin(str(startups
4         ['links'][i]))
5         print(linkedin(str(startups['links'][i])))
6     except:
7         startups['linkedin'][i]='https://www.linkedin.
8         com/company/'+str(startups['startups'][i]).replace('
9         ','-').replace('.', '').replace(',','').replace
10        (' ','').replace(' ','').replace('&','').replace
11        ('/','').replace('\ ','').replace('\n','').replace
12        ('!','').replace('?','').replace(':', '').replace
13        (;','').replace('+','').replace('=','').replace('@
14        ','')

```

### 3.2 Number of employees

#### Methodology:

- Setting up ChromeDriver: The path to the ChromeDriver is passed in explicitly using a variable, as well as an option to run the browser in headless mode for CPU and performance reasons. The Chrome driver instance is created using `webdriver.Chrome()`.
- Logging into LinkedIn: The login page of LinkedIn is accessed using `driver.get()`. The username and password input fields are located using `WebDriverWait()` and the respective element locators. The username and password are entered using the `send_keys()` method. Finally, the login button is located and clicked.
- Scraping Data from LinkedIn: The code iterates over the LinkedIn profiles

of startups stored in the 'linkedin' column of the 'startups' DataFrame. For each profile, the 'about' page is accessed using `driver.get()`. The page source is then parsed using BeautifulSoup to extract specific information like the number of employees, founding date, keywords, and industry.

- Storing Data in DataFrame: The extracted information is stored in the 'employees', 'founding\_date', 'key\_words', and 'industry' columns of the 'startups' DataFrame. If any of the extracted information is not available, the corresponding DataFrame cell is assigned a None value.

- Quitting the Driver: After scraping data from all the LinkedIn profiles, the driver is closed using `driver.quit()`.

**Remark 3.** This methodology outlines the steps involved in scraping LinkedIn data using Selenium and BeautifulSoup, specifically targeting the 'about' page of startup profiles. Additional preprocessing or analysis steps can be performed on the extracted data as per the project requirements.

**Remark 4.** I used my linkedin information that I deleted in the following code.

```

1
2 # Create a Chrome driver instance
3 chrome_options = Options()
4 chrome_options.add_argument("--headless")
5 chrome_options.add_argument("--path-to-extension=C:/
    Users/Neirouz/AppData/Local/Google/Chrome/User Data/
    Default/Extensions/fmkadmapgofadopljbjfkadkoiениhi
    /4.10.1_0")
6 driver = webdriver.Chrome(options=chrome_options)
7
8
9 # Open the login page
10 driver.get("https://www.linkedin.com/login")
11
12 # Find the username and password input fields
13 username_input = WebDriverWait(driver, 10).until(EC.
    presence_of_element_located((By.ID, "username")))
14 password_input = WebDriverWait(driver, 10).until(EC.
    presence_of_element_located((By.ID, "password")))
15
16 # Enter the username and password
17 username_input.send_keys("neirouz.bouchaira@ecl21.ec-
    lyon.fr")
18 password_input.send_keys("*****")

```

```

19
20 # Find and click the login button
21 login_button = WebDriverWait(driver, 10).until(EC.
    element_to_be_clickable((By.XPATH, "//button[@type='
        submit']")))
22 login_button.click()
23
24 for i in range(0, len(startups)):
25     link=startups['linkedin'][i]
26     driver.get(str(link)+"about")
27     print(i)
28     soup = BeautifulSoup(driver.page_source, 'lxml')
29     emp=soup.find('dd', {'class': "t-black--light text-
        body-medium mbl"})
30
31     #save the number of employees in the dataframe in
    the column employees
32     if emp!=None:
33         startups['employees'][i]=emp.text
34         else:
35             startups['employees'][i]=emp
36
37
38
39 driver.quit()

```

## 4. Categorization

Since a training set is not available, an approach is adopted where a code is written for each category of interest. Clustering techniques are then applied to group companies based on their descriptions. This allows for unsupervised categorization, where companies with similar descriptions are clustered together. The resulting clusters provide a way to identify patterns and similarities among companies without relying on a predefined training set. This approach offers flexibility in categorization and can uncover hidden relationships within the data.

```

1 vectorizer = TfidfVectorizer(stop_words='english')

```

### 4.1 Pre-processing

The first step is to convert the preprocessed descriptions of the companies into numerical vectors. This is achieved by using TF-IDF. These vector representations capture the semantic meaning of the documents.

## 4.2 Clustering

The K-means clustering algorithm is applied to group the startups based on their vector representations. The algorithm identifies clusters of similar startups without any predefined categories. The number of clusters can be specified manually or determined automatically by the algorithm.

**Remark 5.** Only the categories

1. Software / IT services / No software or IT services
2. B2B / B2C / Other
3. AI-core / AI-enhanced / Non-AI

are kept for a simplification of the work. As a consequence, we have 27 clusters.

```

1 # Perform clustering
2 num_clusters = 27 # Number of clusters to create
3 kmeans = KMeans(n_clusters=num_clusters)
4 kmeans.fit(X)
5
6 # Assign categories to the clusters
7 cluster_labels = kmeans.labels_
8
9 # Assign categories to individual companies
10 data['category'] = cluster_labels
11
12 # Save the categorized data to a new CSV file
13 data.to_csv('categorized_companies.csv', index=False)

```

## 4.3 Supervised learning

Another method for categorizing the companies is through supervised learning using a pre-trained NLP algorithm, which can be further trained with the available database. The process involves manually categorizing a subset of companies, typically around 50, and using this labeled data to train the spaCy model to categorize the remaining companies automatically.

To train the spaCy model, the training data is prepared by associating each company's description with the corresponding category and subcategory labels. The categories and subcategories defined earlier are used for this purpose. The training data is then shuffled to ensure randomness during training. The training process entails updating the pre-trained spaCy model with the prepared training data. The text classification pipeline within the model is retrieved, and the defined categories are added as labels to the pipeline. The model is trained for a specific number of epochs, with each epoch going

through batches of the training data. The optimizer is employed to update the model parameters based on the calculated losses.

Once the model is trained, it can be tested using example texts. The trained model is applied to sample texts, and category scores are obtained. By applying a threshold (e.g., 0.5), the predicted categories are determined based on the scores. Categories with scores above the threshold are considered as the final predicted categories for the texts.

```

1 import spacy
2
3 # Load the pre-trained spaCy English model
4 nlp = spacy.load("en_core_web_sm")
5
6 # Example company descriptions
7 descriptions = [
8     #descriptions of the companies
9 ]
10
11 # Define the categories
12 categories = {
13     "nature_of_company": ["Software", "IT services", "No
14         software", "No IT services"],
15     "hardware": ["Hardware", "Hardware-enabled", "No
16         hardware"],
17     "business": ["B2B", "B2C", "Other"],
18     "industry": ["Vertical software", "Horizontal
19         software", "Unknown"],
20     "ai": ["AI-core", "AI-enhanced", "Non-AI", "Unknown
21         "]
22 }
23
24 # Prepare training data
25 train_data = []
26 for description in descriptions:
27     doc = nlp(description.lower())
28     category_labels = {category: 0 for category in
29         categories}
30
31     # Check for keywords in the description and assign
32     labels
33     for category, keywords in categories.items():
34         for keyword in keywords:
35             if keyword.lower() in description:
36                 category_labels[category] = 1

```

```

31
32     train_data.append((doc.text, {"cats":
33         category_labels}))
34 # Load the existing model or create a new blank model
35 if "textcat" not in nlp.pipe_names:
36     textcat = nlp.create_pipe("textcat")
37     nlp.add_pipe(textcat, last=True)
38 else:
39     textcat = nlp.get_pipe("textcat")
40
41 # Add labels to the text classification pipeline
42 for category in categories:
43     textcat.add_label(category)
44
45 # Train the model with the prepared training data
46 nlp.begin_training()
47 for _ in range(10): # Number of iterations
48     losses = {}
49     for text, annotations in train_data:
50         nlp.update([text], [annotations], losses=losses)
51     print(losses)
52
53 # Save the updated model
54 nlp.to_disk("trained_model")

```

## 5. Suggestions

- **Data Sources:** To gather comprehensive information about startups, consider expanding beyond LinkedIn and exploring additional sources such as Crunchbase, AngelList, or industry directories.
- **Feature Engineering:** Extract and derive relevant features from the available data. Besides basic information like number of employees and founding date, calculate funding amounts, identify key technologies or products, and analyze the geographical distribution of startups.
- **Sentiment Analysis:** Perform sentiment analysis on company descriptions or news articles to understand the overall sentiment and public perception surrounding startups and industries.
- **Topic Modeling:** Apply techniques like Latent Dirichlet Allocation (LDA) or Non-Negative Matrix Factorization (NMF) to identify key

topics or themes in startup descriptions. This can provide deeper insights into areas of focus or expertise.

- **Network Analysis:** Explore relationships between startups, investors, founders, or industry influencers. Analyze the network structure to uncover patterns, influential players, or potential collaboration opportunities.
- **Benchmarking:** Benchmark startups against industry standards or key performance indicators (KPIs). Evaluate performance relative to peers and gain insights into the competitive landscape.