

Midterm Report: from Web Scraping to Job Advice using the Job Platform “De.Indeed.com”



Hertie School

Prof. Dr. Hannah Béchara

Spring Semester 2019

By Aline Bocamino, Andrea Giuliani,

Mariana Saldarriaga, Toma Pavlov

1 First Step: Scraping the Data

As part of the first step of our project, we scraped data analyst and data scientist job ads from the German version of indeed.com. We chose to use the Beautiful Soup package, since it is one of the more established web scraping tools in Python (initial release was in 2006), as well as one that allows to parse content from within an HTML container with relatively simple commands. Identifying the appropriate HTML snippets for the fields we wanted to scrape was the hardest part of this task. The challenge stemmed from (1) the overall number of lines of HTML code (approx. 21 000 lines) and (2) the elements needed for the parsing would appear repeatedly numerous times throughout the HTML code, but only one of them would return the desired output. We inspected the source page and each element in the HTML tree, tried until Beautiful Soup returned the desired fields. We did two separate scrapings depending on our search query and hence we have two data outputs in the form of csv files: one containing the list of results from the “data analyst” query and another with the “data scientist” results. In separate columns, we scraped the job title, location, company name, rating, date, and link to the ad in this first step.

In the second step of the scraping, we used Python’s requests HTTP library, released under the Apache License 2.0, in order to obtain the full job description. The results page does not show the full job description, but only the basic information, such as job title, date posted, company name, and a limited amount of text from the job ad. Using the requests library and the link to the individual ads that we had already obtained, we were able to send a request from within Python to open each link, scrape the full job description, and add the text to the existing output. We ended up with the following code:

```
def extract_fulltext(url):  
    try:  
        page = requests.get('https://de.indeed.com' + url)  
        soup = BeautifulSoup(page.text, "lxml", from_encoding="utf-8")  
        spans = soup.findAll('div', attrs={'class':  
'jobsearch-jobDescriptionText'})  
        for span in spans:  
            return (span.text.strip())  
    except:  
        return 'NOT_FOUND'  
return 'NOT_FOUND'
```

Unfortunately, we were not able to obtain the type of employment field. The ads do not have the type of employment as a designated field in the HTML, the only choice was including the information in the full description or not including it at all. Furthermore, we would have wished to obtain information on salary amounts, but there was no designated field and the majority of the job ads do not mention it even in the full description. Because the job location was sometimes missing in the list of results, we added a second scrape on location from the full job description and hence why we have more than one location column. Following best practice, we created a unique id for each row, as well as a file that writes the logs to a text file during the scraping process. Overall, we were able to scrape approximately 900 job ads from each of our two queries, or roughly 1800 entries in total (including duplicates), separated in two csv files.

2 Second Step: Cleaning Data & Analyzing Trends

2.1 Diagnose data for cleaning

Cleaning data is a process used to prepare data for analysis. Data almost never comes in clean, especially when doing web scrapping. These are some data problems that we are trying to avoid:¹

1. Inconsistent column names
2. Missing data
3. Untidy
4. Need to process columns
5. Column types can signal unexpected text values

The first step was to visualize the database. For the midterm report, we decided to focus on the dataset containing information on job postings for data scientists and we generated proper labels for the columns. Also, we observed some missing data “NOT FOUND”. The column location was repeated twice and not in the same format as the one we wanted. Finally, we saw some columns that needed further manipulation like job query, job title, full text and date. Our database initially had 866 rows and 12 columns. This means that we started working with a sample of 866 job offers that were scraped.

Secondly, we printed the information of the dataframe, we observed that there weren’t any missing values. We took a deeper look into the database and we saw the missing values were not detected since they were string “NOT FOUND”. This was something that we had to fix. We also saw that all our columns were objects.² Nevertheless, the column date returned

1. We do not worry for outliers since the purpose of our project is to describe the job market in Berlin. Also, we try to avoid missing data. However, NaN will not affect the purpose of our project. Hence, we decided to collect as much information as possible in order to obtain a better overview of the job market supply for data scientists.

2. Objects are stored as strings.

a numerical value, but it was also stored as an object. Therefore, this column had to be converted into an integer.

2.2 Cleaning data

We started by renaming the columns as specified in the code. We defined a dictionary that mapped current column names (as keys) to more usable ones (the dictionary's values). Then, we renamed the columns in our dataframe.

Afterwards, we looked for missing values. For the variable "Company name", we observed only 9 missing values. However, for the columns "Location" and "Rating" we saw 372 and 496 missing values respectively. These missing values corresponded to data not found during the web scraping. Nevertheless, we decided that those columns had valuable data and should not be dropped. We marked the "NOT FOUND" as NaN in Python. Besides, we observed that "Location" had many missing values and that it contained less valuable information than columns "Location2" and "Location3". Therefore, we decided to drop "Location".

```
In [209]: print(df.isnull().sum())
Unnamed: 0      0
Id              0
City            0
Search          0
Job_title       0
Company_name    0
Location       372
Rating          496
Website         0
Days_posted     0
Job_description  0
Location2       0
dtype: int64
```

Since all columns were stored as objects, we converted the variable 'Days posted' into a number. We extracted the number of days that a job had been post for and turned it into a numeric value. We performed the same action with Rating.³

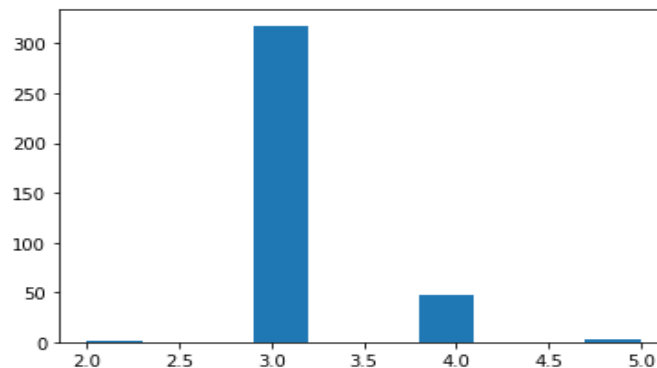
3. As a consequence, the number of missing values decreased.

```

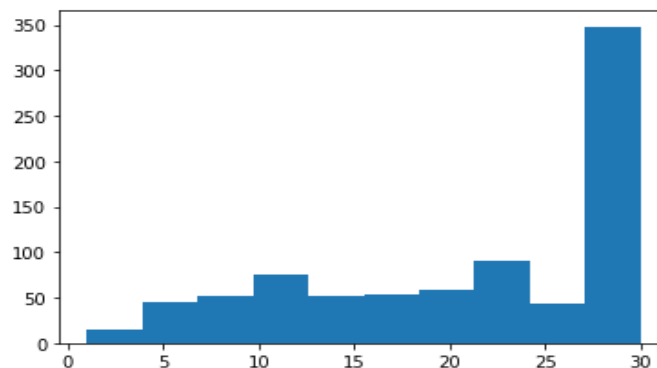
In [217]: print(df1.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 866 entries, 0 to 865
Data columns (total 12 columns):
#   Column              Non-Null Count  Dtype
---  ---
0   Unnamed: 0          866 non-null    int64
1   Id                  866 non-null    object
2   City                866 non-null    object
3   Search              866 non-null    object
4   Job_title           866 non-null    object
5   Company_name        866 non-null    object
6   Rating              370 non-null    float64
7   Website             866 non-null    object
8   Days_posted         866 non-null    object
9   Job_description     866 non-null    object
10  Location2           866 non-null    object
11  Days_posted_2       830 non-null    float64
dtypes: float64(2), int64(1), object(9)
memory usage: 81.3+ KB
None

```

The summary statistics shows that as a general trend, a job offer was posted on average 21 days ago in the last month. The mean of ratings in jobs positions is 3.1. We visualized the distribution of the variables with the following graphs:



Most of the job positions have 3 stars rating.



Most of the job ads in our database were posted 30 days ago or earlier.

The next step we completed was to extrapolate from the column “Location2” the postal code in Berlin and to create another column with this information. Our future goal is to generate a dictionary with the postal code in Berlin. That way we can know the districts where the jobs are offered.

Finally, we did some analysis by examining the values in our dataset. Our dataframe had mostly string data, and “describe()” can only be used on numeric columns. One way to diagnose categorical data is by using the “value counts()” method, which returns the frequency counts for each unique value in a column.⁴

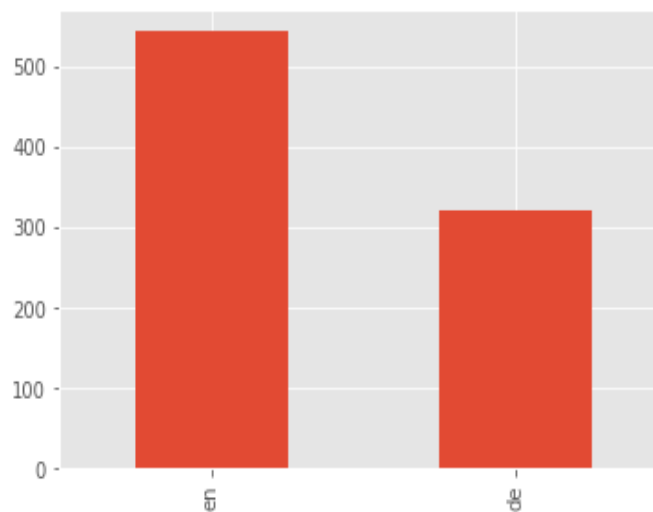
```
In [190]: df.Job_title.value_counts(dropna=False)
Out[190]:
Senior Data Scientist* / Machine Learning Engineer*      49
Junior Data Scientist Advanced Analytics (m/w/d) - Neckarsulm  45
Technische Referentinnen und Referenten (m/w/d) in der Analyseunterstützung und Datengewinnung  43
ML Engineer, Global Data & Insights - Assisted Relocation to Copenhagen, DK  42
Contracts Officer  42
..
Research Scientist, Graphics and Deep Learning  1
Data Scientist (m/f/div)  1
Speech Recognition Scientist  1
Big Data- Engineer - Scientist (m/w/d)  1
2020 Machine Learning Internship - Conversational AI  1
Name: Job_title, Length: 152, dtype: int64
```

```
In [191]: df.Company_name.value_counts(dropna=False)
Out[191]:
Comtravo GmbH  105
LIDL Stiftung & Co. KG  87
Pandora  80
Max Planck Institute for Human Development  70
inovex GmbH  49
...
Takeaway.com  1
Berliner Wasserbetriebe  1
Zero to One Search  1
Fintu Data Science GmbH  1
Omio  1
Name: Company_name, Length: 122, dtype: int64
```

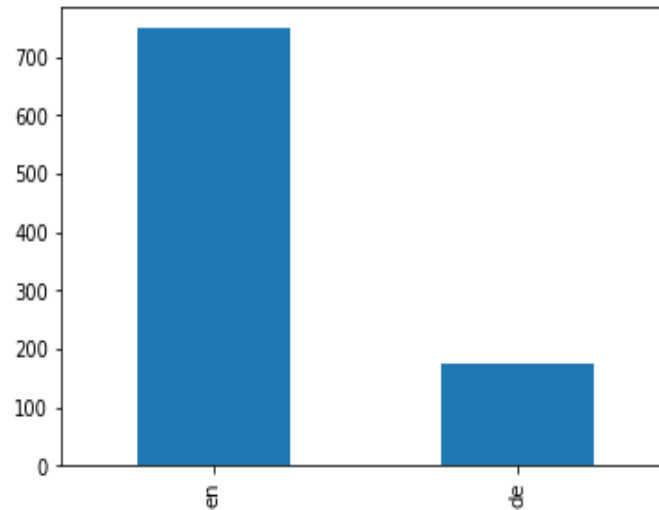
4. This method has an optional parameter called dropna which is “True” by default. What this means is that if you have missing data in a column, it will not give a frequency count of them. Here we set the dropna column to “False” so that if there are missing values in a column (like in Company name), it will give us the frequency count.

2.3 Creating the Language Column

After having cleaned the data and analyzed some initial trends, we created a new column named “Language” in order to classify job ads. Precisely, we aimed at having a clear overview of how many job postings in the two datasets we generated from De.Indeed.com are in German and how many in English. In order to do so, it was necessary to use the package “spacy langdetect”. As mentioned above, for the midterm report we mainly focused on the dataset containing the job postings for data scientists and before using the package on our data, we generated two examples, in order to check whether the function was working correctly. The results show that a majority of job postings is in English, but that there is also a considerable number of job ads in German. To better visualize the results, we created an histogram:



In order to have a more comprehensive understanding of the trends for job postings on De.Indeed.com, we decided to run the same analysis also with the dataset containing the jobs ads for data analysts. The histogram that we generated for this other dataset displays similar results to the previous one:



3 Third Step: Creation of the Job Recommendation Engine

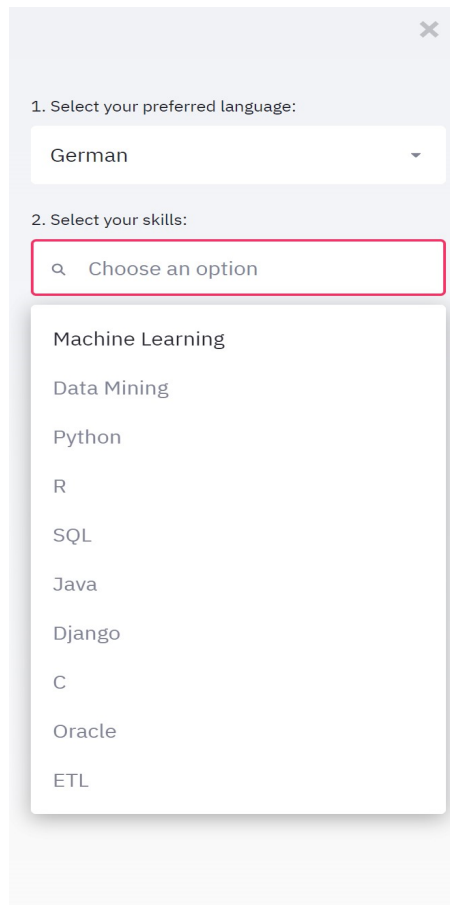
The third part of the project, the Job Recommendation Engine, was composed by two main tasks:

1. To create a user interface which collects the user's preferences and inputs
2. To display a recommendation list, where the jobs postings are filtered according to the user's inputs.

As indicated in the project proposal, both tasks were conducted with Streamlit, which is an open-source app framework for creating dashboards using Python. Since we are still working on extrapolating information from the dataframes that we generated using web scraping, we used a mock dataset in order to deliver a working demo for the midterm report. The first step was to set up Streamlit. In order to visualize the dashboard, we then typed the following in the command line: `streamlit run app.py`. After that, we could open the local URL in the browser and visualize the dashboard.

```
You can now view your Streamlit app in your browser.  
  
Local URL: http://localhost:8501  
Network URL: http://192.168.0.144:8501
```

After these initial settings, we finally started to code. The first step was to create the dropdown menus, where the user can select his/her preferences. As can be seen in the figure below, two dropdown menus were created, with the following fields: “1. Job-posting Language”, and “2. Skills”. Streamlit makes it very easy to create a left panel sidebar with `st.sidebar` as well as to create a dropdown menu with `selectbox` or `multiselect`.



1. Select your preferred language:

German

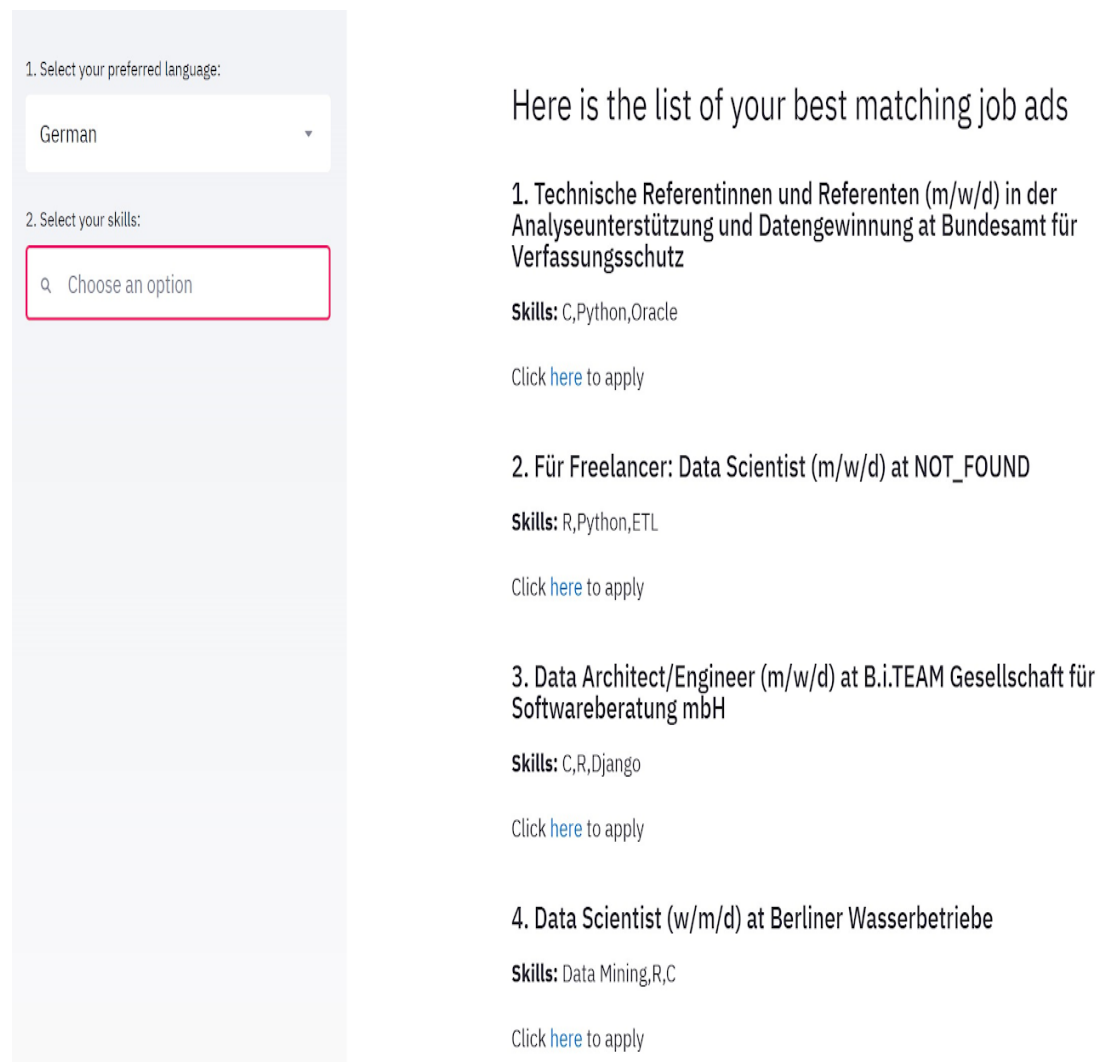
2. Select your skills:

Choose an option

- Machine Learning
- Data Mining
- Python
- R
- SQL
- Java
- Django
- C
- Oracle
- ETL

For the language dropdown menu, we used the unique values in the column “Language” of our dataset. However, for the “skills” attribute, since we are still working on mining the

dataset and creating a column with the skills required, a list was manually created with 10 options. The second task was to display the “Recommendation List”. Right now, the jobs are filtered only by “Language”. As can be seen in the figure above, Streamlit makes it very easy to alter the format of the interface. Therefore, the group could display the list in a very user-friendly way, for example, it added a header in the list, as well as including other important information into the list, such as the company name and the link for applying to the position.



1. Select your preferred language:

German ▼

2. Select your skills:

🔍 Choose an option

Here is the list of your best matching job ads

- 1. Technische Referentinnen und Referenten (m/w/d) in der Analyseunterstützung und Datengewinnung at Bundesamt für Verfassungsschutz**
Skills: C,Python,Oracle
[Click here](#) to apply
- 2. Für Freelancer: Data Scientist (m/w/d) at NOT_FOUND**
Skills: R,Python,ETL
[Click here](#) to apply
- 3. Data Architect/Engineer (m/w/d) at B.i.TEAM Gesellschaft für Softwareberatung mbH**
Skills: C,R,Django
[Click here](#) to apply
- 4. Data Scientist (w/m/d) at Berliner Wasserbetriebe**
Skills: Data Mining,R,C
[Click here](#) to apply

4 Next Steps

1. We have to merge the two dataframes containing information on job ads for data scientists and data analysts, and we need to finish extrapolating information in order to use the data for our job recommendation engine.
2. We need to process the text of our columns “Job title”, “Company name” and “Job description”. A question that came up is if it is better to convert those variables into categorical variables to visualize the data in plots. However, we have a large database from many companies and many job titles that we need to unify. A dictionary might be the best way to simplify the data.
3. We might want to combine the columns “Location2” and “Location3”. We can replace “None” by “Deutschland” in “Location3”, so we don’t have missing values in that column. This will then be our main variable for Location.
4. The logic of this “skills” filter is not as simple as the language filter. Firstly, because it depends on the previous task on mining all job ads and creating a column of “Required Skills”. Secondly, the logic is more complicated, since there will be three different cases: perfect matches, partial matches and no match. This logic must be implemented and it is still being evaluated by the group.
5. In addition, we will implement the sorting of this list by the “Date posted”, this is, to present the most recent postings first. Our initial ideas on sorting the list by salary and company review will not be implemented since most of the job ads do not present such information.
6. Finally, we also plan to deploy our dashboard as a website, so that our app can be accessible to potential users.