Extraction, Transformation, and Load Technical Report

For

Jobs Data

Georgia Tech

ETL Project

Contents

[1. Introduction 2](#_Toc24752043)

[1.1 Summary 2](#_Toc24752044)

[1.2 Scope 2](#_Toc24752045)

[1.3 Technologies and resource contributions 3](#_Toc24752046)

[1.4 Definitions, Acronyms and Abbreviations 3](#_Toc24752047)

[2. ETL Details 4](#_Toc24752048)

[2.1 Data Import/Extract Sources and Method 5](#_Toc24752049)

[Fetching data by cities – 5](#_Toc24752050)

[2.2 Data Acquisition 8](#_Toc24752051)

[Pagination 8](#_Toc24752052)

[Extracting job listing 9](#_Toc24752053)

[2.3 Data Transform 10](#_Toc24752054)

[2.4 Data Integrity 11](#_Toc24752055)

[2.5 Data Refresh Frequency 12](#_Toc24752056)

[2.6 Data Security 12](#_Toc24752057)

[2.7 Data Loading and Availability 12](#_Toc24752058)

[3. Data Quality 13](#_Toc24752059)

[4. References 13](#_Toc24752060)

# 1. Introduction

## 1.1 Summary

Objective of “Jobs Data” - ETL project is to extract “All” Jobs information from different sites & for different Cities\States in USA; and also automate this process to run every day in-order to get latest jobs data for analysis.

Such data will be ultimately leveraged to perform analysis by Cities\States and also by Salary\Jobs Types. Or even to find out in which jobs are available in a particular time period, or which jobs are having more demand currently.

Focusing on above objective, we have pulled in following data points:

* job title,
* company name
* city, state
* salary
* days of posting

## 1.2 Scope

Scope of this project is limited to Extraction, Transformation & Loading of a data from Glassdoor & Monster job sites. However, we tried structure program in such a way that - we can incorporate more jobs sites easily as per future need.

Program currently pulls data for 100 distinct cities [50 per job site] in USA. However, adding more cities in USA or even other part of the world is an easy task.

Program travers through first 2 pages of both the job sites for all 100 cities. We have assumed that all new/latest jobs will be listed in first 2 pages of any site, this way program will fetch only new jobs every day.

Reporting \ Analysis on this data is limited only till Excel/Power Pivot. Any further reporting or predictive analysis is not in score of this project.

Extracted data needs to be messaged further based on business need – e.g. 5 days ago to 5 or 85K to 85000 – which is not in a scope of this project.

Following are the software requirements for this project to run

1. Python
2. Postgres SQL Database
3. A good internet connection
4. Windows\Linux or Mac OS.

## 1.3 Technologies and resource contributions

* Grettel Canepari
* Katherine Lee
* Mandar Gogate
* Petra Alex
* Preet Puri
* Sweta Shekhar
* **Extraction:** beautiful soup to extract the data from the html pages
* **Transformation:** python
* **Loading:** We used psycopg2 library & PostgreSql database to store the data
* **Editor –** for currently project, team has used VS Code for entire coding.

## 1.4 Definitions, Acronyms and Abbreviations

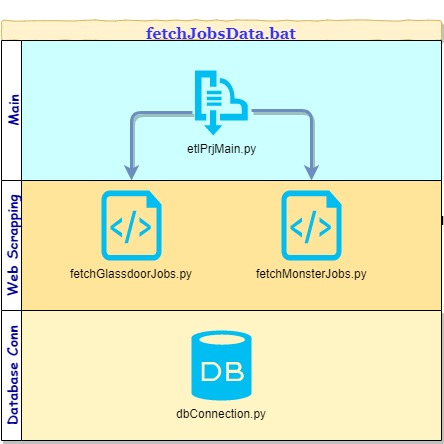
Following acronyms or definitions which you will come across as part of this project scope.

* BS4: Beautiful Soup
* ETL: Extract, Transform and Load
* dbConn: Database connection file
* Conn: Database connection object
* Hdr: header
* Html: hyper text markup language
* SQL: Structured Query Language

# 2. ETL Details

Following is a high-level diagram of Job Search ETL Process. Using windows scheduler - fetchJobsData.bat is scheduled to run everyday at 9.00 PM EST, which then ultimately calls other programs in mentioned order. Details about every program is discussed further in this document

1. etlPrjMain.py
2. fetchGlassdoorJobs.py
3. fetchMonsterJobs.py
4. dbConnection.py

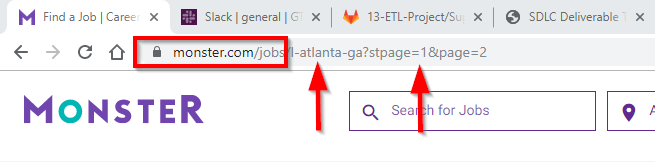


## 2.1 Data Import/Extract Sources and Method

### Fetching data by cities –

Since, the objective of entire process was to pull All jobs for 100 cities from glassdoor and monster, we need to understand the pattern of URL formation.

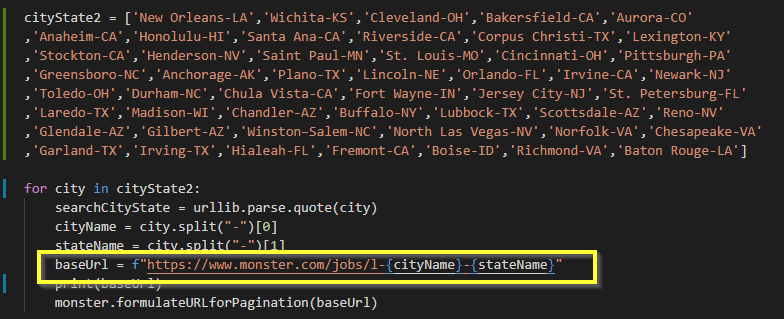
Monster Jobs URL - <https://www.monster.com/jobs/l-atlanta-ga?stpage=1&page=2>



As show in above image, in case of glassdoor, main part of the entire url is monster.com/jobs which is separated by parameter l (location) with values as city-state.

Since, we needed data for 100 different cities, our program should create different URLs in above mentioned format – which then be used to download the .html content of it.

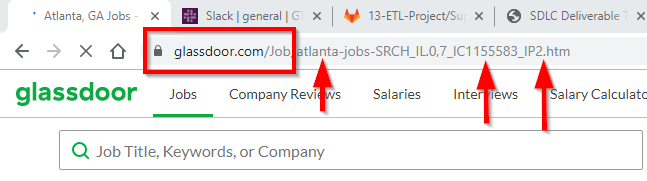
At etlPrjMain.py, we have listed all 100 cities in 2 separate lists [50 for each Glassdoor & Monster], and formulated an url as shown above.



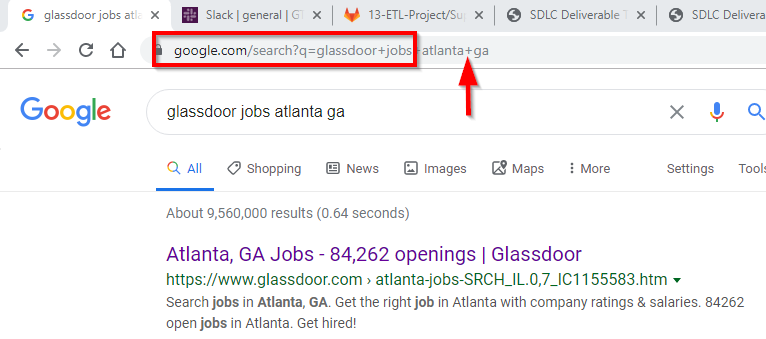
Above code snippet will loop through all cities available, and for each city it will create a base URL and call formulateURLforPagination() function in fetchMonsterJobs.py file for further processing.

Glassdoor Jobs URL - <https://www.glassdoor.com/Job/atlanta-jobs-SRCH_IL.0,7_IC1155583_IP2.htm>

However, formulating a glassdoor URL was not easy, since along with city – glassdoor also adds city/location number in a URL. So, even if we change the city name, we will never know the respective city id for it.

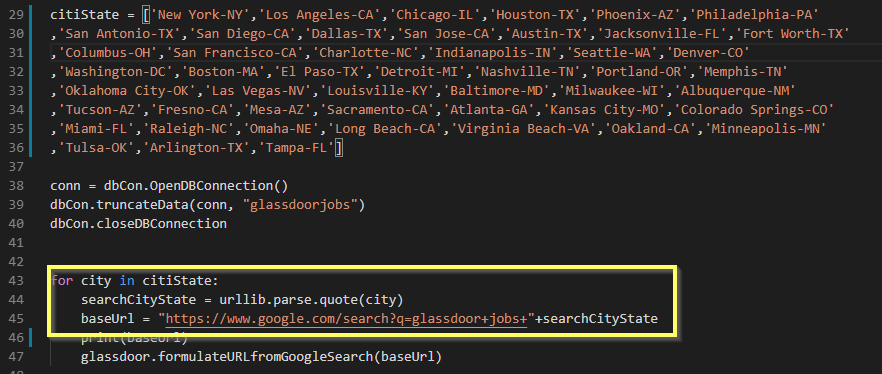


To overcome this, we did the google search first, instead of going directly on glassdoor.com.



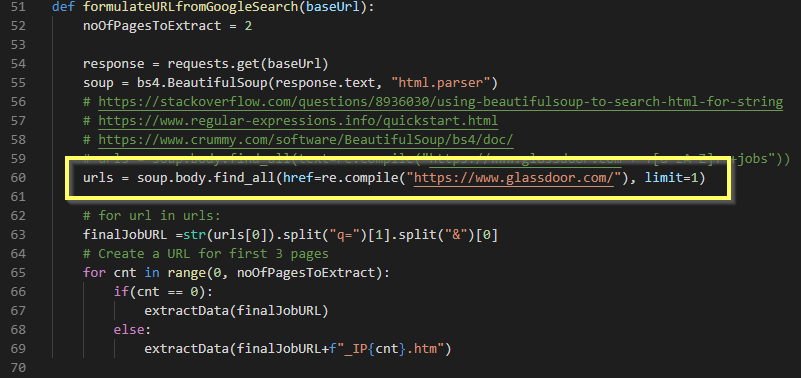
As you can see in above image, using google search url - <https://www.google.com/search?q=glassdoor+jobs+atlanta+ga> – we got very first results and corresponding glassdoor url which we were looking for.

So, for glassdoor we used google search to formulate a base url



As shown in above code snippet, we passed city state combination to google instead of directly to glassdoor [as in case with monster]

In glassdoor case particularly, since our base URL was from google, we needed to fetch actual glassdoor url from google search page.



We just looked for first search results - in downloaded html content – as <https://glassdoor.com> and used that again to download actual jobs .html file. This one extra step was not required in case of monster.

## 2.2 Data Acquisition

### Pagination

As our target was to fetch all job listing from first 2 pages of web-page, we needed to formulate 1 more URL from base url – to navigate through next page. In both – monster & glassdoor – that was easy.

Glassdoor -

for cnt in range(0, noOfPagesToExtract):

        if(cnt == 0):

            extractData(finalJobURL)

        else:

            extractData(finalJobURL+f"\_IP{cnt}.htm")

Monster -

def formulateURLforPagination(baseUrl):

    noOfPagesToExtract = 2

    # Create a URL for first 3 pages

    # https://www.monster.com/jobs/l-atlanta-ga?page=1

    for cnt in range(1, noOfPagesToExtract+1):

        extractData(baseUrl+f"?page={cnt}")

noOfPagesToExtract variable is declared locally, and one has to change the value if in case more pages needs to be extracted. Currently value is set to 2.

### Extracting job listing

Once we have all required URLs ready, next major task was to retrieve all job listing with attributes such as Job Title, Employer, Salary, Date of Posting & Location.

One observation we had is that - what we see on the html page doesn’t match with what is extracted using the code. Due to this, before we wrote the extraction logic, we printed extracted data into .txt file, and then based on that we finalized main logic.

    response = requests.get(url, headers=hdr)

    soup = bs4.BeautifulSoup(response.text, "html.parser")

    # jobListing = soup.find\_all('div', class\_="summary")

    jobListing = soup.find\_all('div', class\_="flex-row")

    for job in jobListing:

        myList = []

        jobName = job.find('div', class\_="summary").header.h2.a.text

        company = job.find('div', class\_="summary").div.text

        location = job.find('div', class\_="summary").find('div', class\_="location").text

        postedOn = job.find('div', class\_="meta flex-col").find('time').text

        myList.append(jobName.replace("\n","").replace("\r",""))

        myList.append(company.replace("\n","").replace("\r",""))

        myList.append(location.replace("\n","").replace("\r",""))

        #Since there is no salary available, defaulting it to 0

        myList.append(0)

        myList.append(postedOn.replace("\n","").replace("\r",""))

        # print(myList)

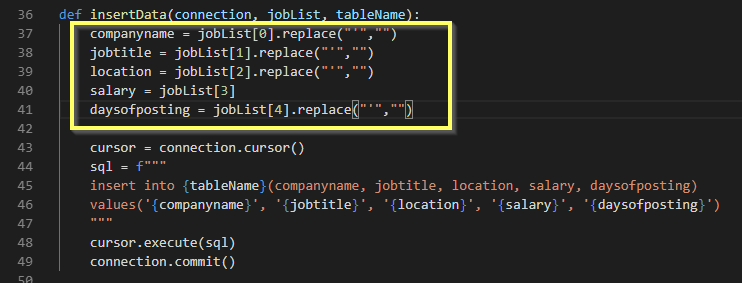
        dbCon.insertData(connection, myList, "monsterjobs")

As you can see in above code snippet of monster, using beautiful soup we extracted required data items from html page and finally called dbCon.insertData method from dbConnection.py for each job listing. Similar process was performed for glassdoor as well.

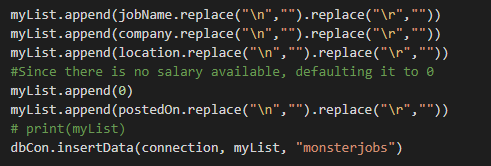
## 2.3 Data Transform

As mentioned earlier, one observation we had to transform is that the url that we obtained from Glassdoor would only provide the location by location ID and not by the explicit city name and state. Due to this, we had to write code to extract the data from google search, which would provide the explicit city name and state within the url address. We then would use this url to scrape the data.

Another item we had to clean up that was causing an error during extraction, is that we had to remove apostrophes from the data before inserting into database.



When we ran Beautiful Soup, it also pulled new line characters(\n) & carriage return characters (\r) along with actual data. Since we only needed actual data and not extra characters, before loading the data into the database, we replaced the new line and carriage return characters with null values.



## 2.4 Data Integrity

For monster & glassdoor jobs, where-ever we could not find job salary for a job listing, we passed default value as 0 there.

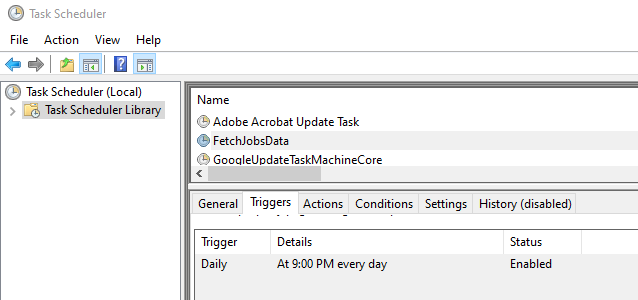
It is assumed that, both mentioned job sites will post or display their new jobs at first or second page on their site.

It is also possible that, we will end-up storing duplicate job everyday if site is not updated or not sorted by job posting date. However, based on date column in respective table, we can distinguish that, or it may be helpful to do analysis - to find out which site updates jobs frequently.

Since job search database tables has CreationDate column - which is defaulted to current date - we can distinguish jobs by each date once data load is finish.

## 2.5 Data Refresh Frequency

Our job is scheduled to run every day at 9:00 p.m. EST using windows scheduler which will run batch file fetchJobsData.bat [python.exe etlPrjMain.py >> log.txt 2>&1]



## 2.6 Data Security

We are storing data in the PostgreSQL database, which is secured by SQL user authentication.

## 2.7 Data Loading and Availability

2 postgres tables - monsterjobs & glassdoorjobs - are created for this data load. Attached are the scripts for both table & database creation.



We have exposed the data to power pivot, which connects to our PostgreSQL database. The user can then access this data and transform it however they’d like. As of today, in single load, this process has pulled 2801 jobs for glassdoor & 1480 jobs for monster.



# 3. Data Quality

Once process gets triggered, it should get finish in less than 10 mins and should insert in total 3000 records for both Monster & Glassdoor jobs site

# 4. References

https://stackoverflow.com/questions/42814637/glassdoor-api-login-not-working-with-python-response-403-bots-not-allowed

https://stackoverflow.com/questions/8936030/using-beautifulsoup-to-search-html-for-string

https://www.regular-expressions.info/quickstart.html

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

http://www.postgresqltutorial.com/postgresql-serial/