Analyzing Data-Related Job Listings in the US

Building a Data Pipeline with

Apache Airflow & AWS Redshift

A project submitted to Udacity in Partial Fulfilment of the Requirements for the Data Engineering Nanodegree

Shaikah Bakerman

October 2020

Table of Content

Table of Figures

1 Project Summary

In this project, I collected datasets of US data-related job listings, namely business analyst, data engineer, data scientist and data analyst jobs, for the purpose of analyzing these job listings, in order to find information about top companies and sectors that have created data-related jobs, and to get some insight on most frequently-mentioned technical skills and tools in the job descriptions.

I created an ETL process to stage data, test, clean and transform them, and then load them into dimensional and fact tables following a Star schema. Later, I implemented data quality checks to validate that all the tables are successfully loaded with data. Finally, I executed a number of SQL queries to answer interesting questions about the hiring companies, listed jobs and their required or preferred technical skills.

Datasets used in this project are data-related job listings, a database of world cities, and a database of the top technical tools and software used in the data industry. This project uses Amazon Web Services (AWS) S3 and Redshift for data storage and warehousing, respectively. In addition, it uses Apache Airflow to schedule and monitor the data pipeline.

2 The Dataset

In this project, I use 6 datasets, 5 of which were retrieved from **Kaggle**:

- **Business Analyst Jobs**: https://www.kaggle.com/andrewmvd/business-analyst-jobs
- **Data Analyst Jobs**: https://www.kaggle.com/andrewmvd/data-analyst-jobs
- **Data Engineer Jobs**: https://www.kaggle.com/andrewmvd/data-engineer-jobs
- **Data Scientist Jobs**: https://www.kaggle.com/andrewmvd/data-scientist-jobs
- **World Cities Database**: https://www.kaggle.com/max-mind/world-cities-database

- **Data Tools Database**: manually created

2.1 Job Listings

The first 4 datasets were scraped from **glassdoor.com** by Kaggle user '**Larxel**' and loaded into *csv* files. These datasets contain 4092, 2253, 2528 and 3909 job listings (rows), respectively.

Each job listing contains **job title**, **salary estimate**, **description**, company **rating**, company **name**, job **location**, company **headquarters**, company **size**, the year the company was **founded**, **type of company**, **industry**, **sector**, company **revenue**, and **competitors**. There are additional columns (**field**, **index**, and **easy-apply**) that are not taken into consideration in this project.

2.1.1 File Format

The original job listing files are in *csv* format, but for the purpose of varying file formats in this project, they were converted into *json* files.

During the csv-to-json conversion process (*json_script.py*), I added an additional field, **job_id** to each job listing, and I changed the keys to match the staging table column names (alternative to creating a JSONPath file).

2.1.2 Splitting Job Listings

When I first tried to stage job listings into AWS Redshift, I received an error due to the size of the json files (*`The total size of the json object exceeds the max limit of 4194304 bytes`*). This is mainly due to the lengthy job description of each listing. As a result, I split each file into three json files; the first file, *part1*, contains job IDs, job titles and salary estimates, the second file, *part2*, contains only the job descriptions, and the third file, *part3*, contains the rest of the information.

2.2 World Cities

The world cities database contains 3,173,958 rows of the world's cities, their countries, region, population, latitude and multitude (licence *MaxMind_LICENSE.txt)*. I staged the

World Cities database to extract US cities and their states into the us_cities dimension table. The total US cities extracted from the World Cities database is 141,982.

2.2.1 File Format

The World Cities file is in csv format, and has been used as is in the project.

2.3 Data Tools

I manually created the data tools database based on my research on the top, most popular and commonly used data analysis / data engineering / data science software tools, such as Apache Airflow, Tableau, SQL…etc. One column contains the tool's search keyword which was used to extract the tool from a job description. The second column contains the tool's complete identifier. Example: *Kafka* is the keyword & *Apache Kafka* is the complete identifier.

The number of rows has frequently incremented as my project progressed.

2.3.1 File Format

The data tools file was created in Excel and exported and used in the project in csv format.

2.4 Cleaning the Data

Most of the data cleaning process is done programmatically after staging the job listings.

2.4.1 Column Shifting due to Missing Values

When examining the datasets, I found out that some of the job listings were shifted due to the absence of some values like **field** and **index**. As a result, this caused values to be misplaced (e.g., descriptions in the place of job titles, job titles in the place of indices, etc.). Therefore, the data is tested before loading the job listings from the staging area into the dimension and fact tables. If columns are indeed shifted, the data are shifted back into their correct columns and nulls are placed in the empty fields accordingly (*job_processing.py*).

```
if current_columns[1] and not current_columns[1].isdigit():
      self.log.info(f"field 1 is storing the job title instead, so we shift the columns")
    new_columns.append(None) # null field1
    new_columns.append(None) # null index_

  else:
      self.log.info(f"no need to shift columns")
```

I. Column Shifting by Inserting Nulls

2.4.2 Single Quotes in Job Descriptions

Given that PostgreSQL inserts string values using single quotes, a problem occurred when single quotes were present as part of the job descriptions. Therefore, as files were converted to json, all single quotes were also removed from the description string (*json_script.py*). The negative side effect of doing this step is minimal because single quotes were all also removed from the data tools database that I compare against job descriptions.

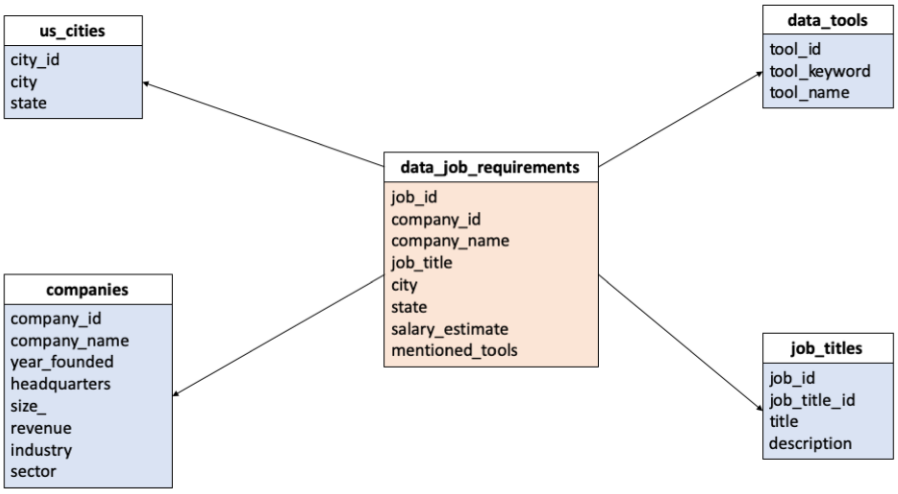
2.4.3 Company Name

- In the case where a company name is not present in the dataset, the job listing is discarded.
- In the case where a company name is present, and there is a rating number attached to the end of the company name string, the rating is removed from the string using a simple *while* loop (*job_processing.py*).

```
def fix_copmany_name(self, name):
    # remove the rating from the end of the string
    # if the company name exists
    if len(name) > 0:
        ind = len(name) - 1
        while ind != -1 and (name[ind] == '.' or name[ind].isdigit()):
            name = name[0: ind]
            ind -= 1

    # remove the extra whitespace and return
    return name.strip()
```

II. Company-Name Cleaning

3 The Data Model



III. The Data Model

I used a **Star Schema** as my data model in this project. The reason why I chose this model is to keep a simple structure since, after staging and loading our data, most of the analysis work is done on one fact table, ***job_data_requirements*****,** which references values from two dimensional tables (*us_cities* and *companies*). The dimensional tables, on the other hand, are never joined together for analysis purposes. Therefore, since the workload is focused on the fact table, the negative effects, like data redundancy, of using the Star Schema are not a major concern in this project, at its current low level of complexity. Another approach might be taken if a more advanced ETL and data analysis process is implemented.

3.1 Data Dictionary

3.1.1 us_cities

Field Name	Data Type	Field Length	Constraint
city_id	BIGINT	8	PRIMARY KEY
city	varchar	256	NOT NULL
state	char	2	NOT NULL

3.1.2 data_tools

Field Name	Data Type	Field Length	Constraint
tool_id	BIGINT	8	PRIMARY KEY
tool_keyword	varchar	256	UNIQUE
tool_name	varchar	256	

3.1.3 companies

Field Name	Data Type	Field Length	Constraint
company_id	BIGINT	8	PRIMARY KEY
company name	varchar	65535	UNIQUE
year_founded	INT	2	
headquarters	varchar	256	
size_	varchar	256	
revenue	varchar	256	
industry	varchar	256	
sector	varchar	256	

3.1.4 job_titles

Field Name	Data Type	Field Length	Constraint
job_id	BIGINT	8	UNIQUE
job_title_id	BIGINT	8	PRIMARY KEY
title	varchar	256	NOT NULL
description	varchar	65535	

3.1.5 data_job_requirements

Field Name	Data Type	Field Length	Constraint
job_id	BIGINT	8	PRIMARY KEY
company_id	varchar	256	PRIMARY KEY
company_name	varchar	256	FOREIGN KEY
job_title	varchar	256	FOREIGN KEY
city	varchar	256	FOREIGN KEY
state	varchar	2	FOREIGN KEY
salary_estimate	varchar	256	
mentioned_tools	varchar	256	

4 The ETL Process

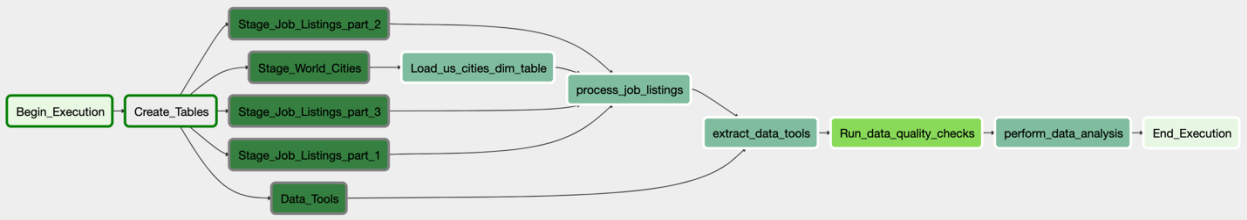
4.1 Tools and Technologies

The entire project is written in Python, including file-format-conversion scripts, with the aid of a few libraries including *pandas*, *json*, *re* (the *RegEx* library*)*, *os,* and *datetime*. In addition to Python, all the data has been stored and retrieved from an AWS S3 bucket named '*dend-capstone-shaikahbakerman*' that is made publicly accessible for the sake of the project. Data is staged and later loaded into dimension and fact tables using an AWS Redshift dc2.large 2-node cluster. Finally, to manage and schedule the tasks, I used Apache Airflow DAGs. Airflow has provided the proper workflow and task dependency management environment that has helped me monitor the process much more efficiently.

```
default_args = {
    'owner': 'Shae Bak',
    'start_date': datetime(2020, 9, 20),
    'email_on_retry': False,
    'retry_delay': timedelta(minutes=5),
    'retries': 3
}

dag = DAG('cap_1.9.44',  # 'Capstone_Project_DAG',
          default_args=default_args,
          description='Job Listing Analysis Pipeline',
          schedule_interval='@weekly',  # run once a week
          max_active_runs=1
          )
```

IV. Airflow DAG Specifications

4.2 The Data Pipeline



V. Overview of the Complete Data Pipeline

4.2.1 Table Creation

The first task is table creation, which is accomplished using a **PostgresOperator,** running the *create_tables.sql* script.

4.2.2 Staging the Data

After table creation, I stage the job listings into 3 different tables (***staging_job_listings_part_1***, ***staging_job_listings_part_2*** *and* ***staging_job_listings_part_3***). As mentioned in 2.1.2, I had to split the job listings because of AWS Redshift's maximum allowed file size. The staging is done using Redshift's COPY command for a faster and more efficient staging process. I also stage the world cities into a separate table (***staging_world_cities***). All staging tasks are done using the **StageToRedshift** operator, the class of which is written in *stage_redshift.py.*

In the staging process, I use Redshift's COPY command to also load the data_tools dimension table.

4.2.3 Loading Dimension Tables

After loading data into the data_tools dimension table, I use the **LoadDimension** operator (*load_dimension.py*) to load US city-state pairs in us_cities from the staged world cities.

```
us_cities_table_insert = ("""
    INSERT INTO us_cities (city, state) (
    SELECT DISTINCT city, region
    FROM staging_world_cities
    WHERE country='us')
""")
```

VI. Inserting data into us_cities Dimension Table

After that, in order to continue data processing, I create a **JobProcessing** operator (*job_processing.py*), where the 3 staged job listing tables are joined first. Then, as mentioned in 2.4.1, I check for misplaced data that was shifted due to missing values. After that, the company name of each job listing is examined. Listings with missing company names are discarded. On the other hand, company names that are present are cleaned as explained in 2.4.3.

A final check in the **JobProcessing** operator, before loading the rest of dimensional tables and a portion of the fact table, is to test if the location of the job is a valid US city and state, by checking to see if it exists in the us_cities dimensional table. If it does not exist, the city and state name are output in the log file, but the listing is still accepted. Examples of cities that were not part of the us_cities table (because they were not part of the World Cities database) are ***Bala Cynwyd, PA***, ***Feasterville Trevose, PA*** and ***City of Industry, CA.***

```
# checking the city and state against our US cities table
# if the city-state pair doesn't exist, we will log it before adding it

# Comparison against the database ignores case sensitivity and whitespace
us_cities_query = """SELECT * FROM us_cities
WHERE REPLACE(LOWER(city), ' ', '') = REPLACE(LOWER(%s), ' ', '')
AND REPLACE(LOWER(state), ' ', '') = REPLACE(LOWER(%s), ' ', '')
"""
get_city = redshift_hook.get_records(us_cities_query, (loc_city, loc_state))
if len(get_city) < 1 or len(get_city[0]) < 1 or get_city[0][0] < 1:
    self.log.info(f"Location: [{new_columns[8]}] Not Found in us_cities Database")
```

VII. Verifying the Existence of a US City-State Pair

After running the above-mentioned tests, the operator loads data into the companies and job_titles dimension tables. In addition, it loads all data into the data_job_requirements fact table, except the mentioned_tools column (which is done in the next step).

4.2.4 Tool Extraction

One of the most important parts of this project is tool extraction. Tool extraction is done by analyzing the description of each job listing and extracting the mentioned technical tools and skills that are part of the data_tools dimension table. This process is used, in analysis, to find the frequently mentioned technical tools based on job descriptions.

Tool extraction is done using the **ToolExtraction** operator (*extracting_data_tools.py*). In the operator, all data_tools are retrieved, and, then for each job description paragraph, with the aid of Python's regular expression operators, I check whether each data tool (the keyword of that tool, not its full identifier) is found in that description or not using the algorithm in the picture below. After running the description against all data tools, the data_job_requirements fact table is updated by adding the list of mentioned tools that are found in that job listing.

```
# find the tools required in each job description
self.log.info(f"Extracting data tools for each job and adding it to the fact table")

for x in range(len(descriptions)):
    description = descriptions[x].lower()

    tools_list = []
    for tool in tool_keywords:

        result = re.findall(r"[^a-zA-Z]"+re.escape(tool.lower())+"[^a-zA-Z]", description)
        if result:
            tools_list.append(tool)

    if tools_list:
        tools_list = ", ".join(tools_list)
    else:
        tools_list = None

    # update the data_job_requirements fact table
    cursor.execute("UPDATE data_job_requirements SET mentioned_tools = %s WHERE job_id = %s",
                   (tools_list, job_ids[x]))
    connection.commit()
```

VIII. Tool Extraction from Job Descriptions

4.2.5 Testing and Debugging

To debug our pipeline when it runs into errors, various log outputs are inserted, containing the job listing ID and the accompanying info in different operators and functions, to speed up the process of locating the point where the error has occurred.

5 Data Quality Check

The **DataQuality** operator (*data_quality.py*) performs the following quality checks and reports whether the check fails or not:

- Check if the following tables are empty:
    - Data_tools
    - Companies
    - Job_titles
    - Data_job_requirements
- Check if the following table columns are null:
    - Data_tools => **tool_id** and **tool_keyword**
    - Companies => **company_id** and **company_name**
    - Job_titles => **job_id**, **job_title_id** and **title**
    - Data_job_requirements => **job_id** and **company_id**

6 Data Analysis

The main goal of creating the job listing data pipeline is to analyze the data-related job requirements and skills mentioned by hiring companies. Another objective is to find top hiring companies, top hiring cities and other similar analyses (*data_analysis.py*).

Below is a list of questions (queries), we asked, and their results:

6.1 Top 10 hiring companies (based on number of job listings):

```
self.log.info(f"Starting Data Analysis:")

self.log.info("")
self.log.info(f"1.Top 10 hiring companies (based on number of job listings):")
query1 = """
            SELECT company_name, COUNT(*) AS number_of_listings
            FROM data_job_requirements
            GROUP BY company_name
            ORDER BY number_of_listings DESC
            LIMIT 10
         """
cursor.execute(query1)
result = cursor.fetchall()

for x in range(len(result)):
    self.log.info(f"\t{result[x][0]}: {result[x][1]} Listings")
```

IX. Analysis Query 1

```
[2020-10-21 02:13:20,867] {data_analysis.py:35} INFO - Starting Data Analysis:
[2020-10-21 02:13:20,867] {data_analysis.py:37} INFO -
[2020-10-21 02:13:20,867] {data_analysis.py:38} INFO - 1.Top 10 hiring companies (based on number of job listings):
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  Staffigo Technical Services, LLC: 264 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  Apple: 97 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  Diverse Lynx: 92 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  Amazon: 91 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  Kforce: 78 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  IBM: 72 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  Solekai Systems Corp: 62 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  Lorven Technologies Inc: 57 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  Robert Half: 50 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:50} INFO -  Apex Systems: 47 Listings
[2020-10-21 02:13:20,995] {data_analysis.py:53} INFO -
```

X. Analysis Query 1 Result

6.2 Top 10 job locations (cities / states) (based on number of job listings)

```
self.log.info("")
self.log.info(f"2.Top 10 job locations (cities / states) (based on number of job listings)")
query2 = """
            SELECT city, state, COUNT(*) AS number_of_listings FROM data_job_requirements
            GROUP BY city, state
            ORDER BY number_of_listings DESC
            LIMIT 10
         """
cursor.execute(query2)
result = cursor.fetchall()

for x in range(len(result)):
    self.log.info(f"\t{result[x][0]}, {result[x][1]}: {result[x][2]} Listings")
```

XI. Analysis Query 2

```
[2020-10-21 02:13:20,995] {data_analysis.py:54} INFO - 2.Top 10 job locations (cities / states) (based on number of job listings)
[2020-10-21 02:13:21,059] {data_analysis.py:65} INFO -  NewYork, NY: 1047 Listings
[2020-10-21 02:13:21,059] {data_analysis.py:65} INFO -  Austin, TX: 956 Listings
[2020-10-21 02:13:21,059] {data_analysis.py:65} INFO -  Chicago, IL: 920 Listings
[2020-10-21 02:13:21,059] {data_analysis.py:65} INFO -  SanDiego, CA: 909 Listings
[2020-10-21 02:13:21,059] {data_analysis.py:65} INFO -  Houston, TX: 830 Listings
[2020-10-21 02:13:21,060] {data_analysis.py:65} INFO -  SanAntonio, TX: 737 Listings
[2020-10-21 02:13:21,060] {data_analysis.py:65} INFO -  Jacksonville, FL: 601 Listings
[2020-10-21 02:13:21,060] {data_analysis.py:65} INFO -  Philadelphia, PA: 580 Listings
[2020-10-21 02:13:21,060] {data_analysis.py:65} INFO -  LosAngeles, CA: 554 Listings
[2020-10-21 02:13:21,060] {data_analysis.py:65} INFO -  Dallas, TX: 523 Listings
[2020-10-21 02:13:21,060] {data_analysis.py:68} INFO -
```

XII. Analysis Query 2 Result

6.3 Top 5 sectors offering data-related jobs

```
self.log.info("")
self.log.info(f"3.Top 5 sectors offering data-related jobs")
query3 = """
            SELECT sector, COUNT(DISTINCT company_name) AS number_of_companies
            FROM companies
            GROUP BY sector
            ORDER BY number_of_companies DESC
            LIMIT 5
         """
cursor.execute(query3)
result = cursor.fetchall()

for x in range(len(result)):
    self.log.info(f"\tThe {result[x][0]} Sector: {result[x][1]} Listings")
```

XIII. Analysis Query 3

```
[2020-10-21 02:13:21,060] {data_analysis.py:69} INFO - 3.Top 5 sectors offering data-related jobs
[2020-10-21 02:13:21,123] {data_analysis.py:81} INFO -  The Information Technology Sector: 1224 Listings
[2020-10-21 02:13:21,124] {data_analysis.py:81} INFO -  The None Sector: 955 Listings
[2020-10-21 02:13:21,124] {data_analysis.py:81} INFO -  The Business Services Sector: 814 Listings
[2020-10-21 02:13:21,124] {data_analysis.py:81} INFO -  The Finance Sector: 316 Listings
[2020-10-21 02:13:21,124] {data_analysis.py:81} INFO -  The Health Care Sector: 229 Listings
[2020-10-21 02:13:21,124] {data_analysis.py:84} INFO -
```

XIV. Analysis Query 3 Result

6.4 Frequency of mentioning ‘Python’ as a preferred programming language, as opposed to ‘Java’, in data-related jobs

```
        self.log.info("")
        self.log.info(f"4.Frequency of mentioning ‘Python’ as a preferred programming language, as opposed to ‘Java’, in data-related
jobs")
        query4 = """
                    SELECT COUNT(*) AS number_of_listings
                    FROM data_job_requirements
                    WHERE mentioned_tools LIKE '%Python%'
                """
        cursor.execute(query4)
        result = cursor.fetchall()

        python_mentions = result[0][0]

        self.log.info(f"=> Python is mentioned in {python_mentions} job descriptions")

        query4 = """
                    SELECT COUNT(*) AS number_of_listings
                    FROM data_job_requirements
                    WHERE mentioned_tools LIKE '%Java%'
                """
        cursor.execute(query4)
        result = cursor.fetchall()

    java_mentions = result[0][0]

    self.log.info(f"=> Java is mentioned in {java_mentions} job descriptions")
    self.log.info("")

    if python_mentions > java_mentions:
        self.log.info(f"Python is mentioned in {round(python_mentions/java_mentions, 2)} more job descriptions than Java")
    else:
        self.log.info(f"Java is mentioned in {round(java_mentions/python_mentions, 2)} more job descriptions than Python")
```

XV. Analysis Query 4

```
.py:84} INFO -
.py:85} INFO - 4.Frequency of mentioning ‘Python’ as a preferred programming language, as opposed to ‘Java’, in data-related jobs
.py:96} INFO - => Python is mentioned in 4359 job descriptions
.py:108} INFO - => Java is mentioned in 2165 job descriptions
.py:109} INFO -
.py:112} INFO - Python is mentioned in 2.01 more job descriptions than Java
.py:117} INFO -
```

XVI. Analysis Query 4 Result

6.5 Frequency of mentioning Apache Airflow, Spark or Hadoop in a job description

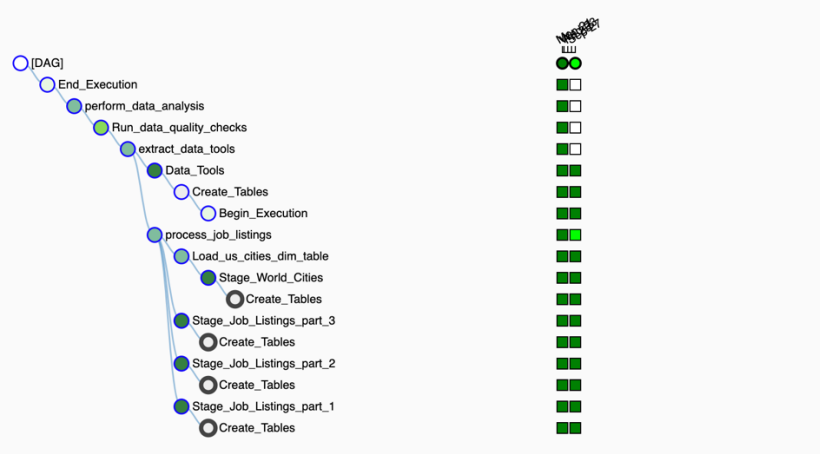
```
        self.log.info("")
        self.log.info(f"5.Frequency of mentioning Apache Airflow, Spark or Hadoop in a job description")
        query5 = """
                    SELECT COUNT(*) AS number_of_listings
                    FROM data_job_requirements
                    WHERE mentioned_tools LIKE '%Airflow%'
                    OR mentioned_tools LIKE '%Spark%'
                    OR mentioned_tools LIKE '%Hadoop%'
                """
        cursor.execute(query5)
        result = cursor.fetchall()

        apache_mentions = result[0][0]

        self.log.info(f"=> {apache_mentions} job descriptions mention Apache Airflow, Spark and/or Hadoop")
```

XVII. Analysis Query 5

```
[2020-10-21 02:13:21,251] {data_analysis.py:117} INFO -
[2020-10-21 02:13:21,251] {data_analysis.py:118} INFO - 5.Frequency of mentioning Apache Airflow, Spark or Hadoop in a job description
[2020-10-21 02:13:21,315] {data_analysis.py:131} INFO - => 2231 job descriptions mention Apache Airflow, Spark and/or Hadoop
[2020-10-21 02:13:23,028] {logging_mixin.py:95} INFO - [2020-10-21 02:13:23,027] {jobs.py:2527} INFO - Task exited with return code 0
```

XVIII. Analysis Query 5 Result

7 Running the Pipeline



XIX. Airflow: One Completed DAG and a Second One in Process

To run the pipeline successfully, there are a few steps that need to be completed.

7.1 Python Libraries

Ensure that you install and import all the required libraries that are mentioned in 4.1.

7.2 AWS S3 and Redshift

Ensure that all data is stored in an S3 bucket, and give it the proper accessibility needed for users. In addition, create a Redshift cluster and fill its database name, user and password information.

7.3 Apache Airflow

Install Airflow and create the required connections by adding your AWS credentials as well as your Redshift cluster information.

7.4 Scheduling Updates

For the project, tasks are scheduled to run once a week. However, this interval can increase or decrease depending on the analysis purpose. For example, if we want to examine the frequency in which companies are posting data-related jobs on glassdoor,

then it is better to run these tasks every day to find out if any new listings were posted by the company on any given day. This also means that we will only be updating our tables with new listings, keeping the old ones unchanged. Thus, we have to access information about job posting date, which is not present in the dataset that I collected for this project.

7.5 Total time of a Completed DAG

The pipeline was gradually tested on smaller to bigger sample sizes of the data, until it included the entire dataset.

The first test was conducted on 100 listings only. The DAG was completed in 4 minutes. The second test was conducted on 2000 listings. The DAG was completed in 44 minutes, with the Job Processing part taking up most of the time duration for a total of 30 minutes.

When I attempted my third test, on 5000 listings, the workflow continued for almost two hours without completing, and being stuck at the job processing part. That is when I decided to remove one part of job processing that, I believe, was causing this delay. The part I am referring to is the one that checks whether each job location exists in the us_cities table. Please refer to *Figure VII* to see the removed code. When the city verification part was omitted, running the DAG with 5000 listings was completed in approximately one hour.

Finally, the complete dataset, of 12,782 listings, was tested and it took approximately 2 hours and 18 minutes for the DAG to complete.

8 Use-Case Scenarios

8.1 Scenario 1: Data was increased by 100x

If the data volume significantly increases, I would upgrade the Redshift cluster with additional nodes to increase parallel processing and computation efficiency. In addition, I believe that I will have to revise all my insertion methods as well as tool extraction algorithm and work on implementing efficient bulk-load algorithms to speed up the insertion process. Finally, wherever applicable, I will avoid truncating tables before each pipeline run. Instead, I will take the approach of only updating the tables and adding new information. This, of course, has to be tested to confirm that, in fact, truncating and loading data would be slower than only updating outdated data and adding new ones.

Another part of the project that I believe is slowing the pipeline down a bit is checking whether the city and state belong to the us_cities table. I will definitely re-consider this verification process if I have a significantly bigger volume of data because running each city against the table will consume a huge amount of time.

8.2 Scenario 2: The pipelines would be run on a daily basis by 7am every day

I will do my best to reduce unnecessary dependencies as much as possible to optimize the workflow. In addition, similar to Scenario 1, I will make sure that I optimize computing capabilities of the cluster.

8.3 The database needs to be accessed by 100+ people

I have to maximize the security of my Redshift cluster by giving limited access to these users based on their user privileges and their assigned responsibilities.

9 Limitations & Future Work

9.1 Salary Extraction

In this project, for future work, I will write a salary extraction algorithm that extracts the salary range of each job listing and converts it to minimum and maximum integer values. This extraction will help analyze the top paid data jobs, top paying companies, and many more salary-related analyses.

9.2 US Cities Database

As mentioned in 4.2.3, when the job location, typically a US city-state pair, is not found in the us_cities table, an announcement is logged indicating that the pair was not found, and then the job location is added normally in the data_job_requirements fact table.

I would like to integrate a user interface to communicate this message to the data pipeline administrator before adding the job to the fact table. This will help confirm, first, if the city-state pair is a valid US city. If the administrator confirms that it's valid, then the pair is added to the us_cities table, and the job listing is added to the fact table. Otherwise, the job listing is discarded.

9.3 Text Extraction and Machine Learning

The text extraction algorithm that I wrote for this project is a very simple algorithm, in which each tool is compared against the job description paragraph. I would like to add a machine learning capability to my algorithm, where the job description text is analyzed against search engine tool results to extract more tools than the ones that already exist in the data_tools database. The program will then add these tools to the database to enrich it with more tools mentioned in these job listings.

I can also use machine learning to validate US city-state pair values, as explained in 9.1, instead of asking the pipeline administrator.

10 Index A: Project Directory

conversion_scripts /

- ⇒ *csv_script.py*
- ⇒ *json_script.py*

dags /

- ⇒ *create_tables.sql*
- ⇒ *main.py*

data /

data_tools /

- ⇒ *data_tools.csv*

job_listings_json /

part1 /

- ⇒ *BusinessAnalyst_part1.json*
- ⇒ *DataAnalyst_part1.json*
- ⇒ *DataEngineer_part1.json*
- ⇒ *DataScientist_part1.json*

part2 /

- ⇒ *BusinessAnalyst_part2.json*
- ⇒ *DataAnalyst_part2.json*
- ⇒ *DataEngineer_part2.json*
- ⇒ *DataScientist_part2.json*

part3 /

- ⇒ *BusinessAnalyst_part3.json*
- ⇒ *DataAnalyst_part3.json*
- ⇒ *DataEngineer_part3.json*
- ⇒ *DataScientist_part3.json*

world_cities /

⇒ *MaxMind_LICENSE.txt*

⇒ *worldcitiespop.csv*

plugins /

helpers /

⇒ *sql_queries.py*

operators /

⇒ *data_analysis.py*

⇒ *data_quality.py*

⇒ *extracting_data_tools.py*

⇒ *job_processing.py*

⇒ *load_dimension.py*

⇒ *stage_redshift.py*