

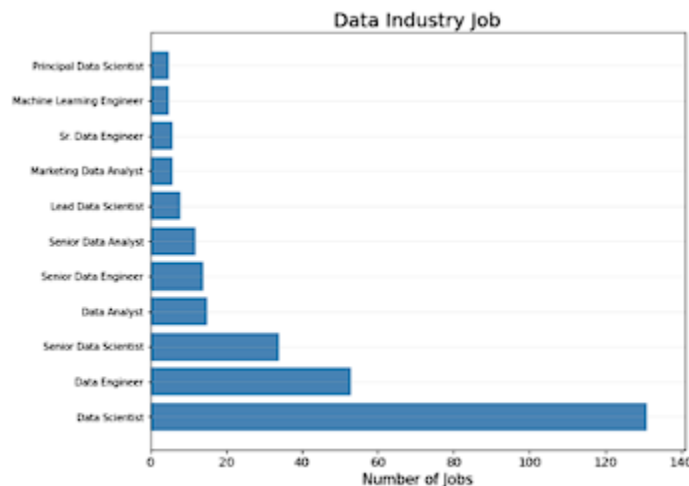
April Gao, Bekhem Horne, Han Le, Matthew Bailey, Raymond Bell

Project -2 Group-4

# Data Analyst ETL Writeup

The tech industry is continuously growing and evolving at a rapid rate. Given the abundance of big data, more companies are seeking data technologists with the requisite skills to transform this big data into useful information to drive better business decisions. This shift in the data landscape has taken shape across nearly every sector, placing a high demand on data analysts, engineers, and scientists. The onset of the pandemic, coupled with this ever-increasing demand for data technologists made remote job opportunities even more widespread. As future data technologists, understanding the distribution of jobs, in terms of salary, location, popularity, and demand allows us to target our search efforts as well as fine tune our career action plans. This knowledge of which job titles offer the highest salary, including location information helps us execute our action plans with a higher level of confidence. That is the goal and desired outcome of this project.

To accomplish this goal, we performed a series of web scraping procedures to extract data from Indeed.com. Next, we transformed and loaded the extracted data into a database created in Postgres SQL. Prior to web scraping we loaded a CSV data set from Kaggle into a Jupyter Notebook where we used Pandas to examine the various data related jobs. The predominant job titles were data analyst, data engineer, and data scientist, which matches those that we mentioned above. We used this to build our search model.



When we started to scrape Indeed.com, we narrowed our job search to Data Analyst. Here we used Splinter and Chrome Driver Manager. After navigating to the website, we used a for loop to iterate fifty pages to return a list of dictionaries of the following items: job title, job link, company, company link, company rating, location, salary, and status.

This for loop contained a series of “finds” and “tags” to locate the specific information that we wanted to extract from the HTML code from the developer page. To begin, we

had to locate what we were wanting to find within indeed.com using the inspect function, locate a commonality for our class and what tag it resided in. For example, to find the salary of each job we had to locate the container that held the text of each job listing shown in the example below. Within a <div>

```
#Within the Salary for loop we had to include an if statement due to
salary_con = job.find('div', {"class": 'salary-snippet-container'})
if salary_con:
    salary = salary_con.text
else:
    salary = None
```

tag we were able to find a common class of ‘salary-snippet-container’ that allowed us to pull the salary text.

Our first attempt to return the job title with company link was easy due to the similarity in the initial code. The only changes needed was changing “.text” to “[href]”. However, once we printed out the

```
for job in jobs:
    title = job.find('a', {"class": 'jcs-JobTitle'}).text
    link = "www.indeed.com" + job.find('a', {"class": 'jcs-JobTitle'})["href"]
```

results, we found that the URL did not return the correct

format. We added into the code shown [www.indeed.com](http://www.indeed.com) plus our code to print off a correctly formatted URL.

## Visualizations and Queries

### Total Jobs by State

dependenciesDependentsProcessesdepartment23...Project2\_job\_b...Project2

Project2\_job\_board/postgres@PostgreSQL 15

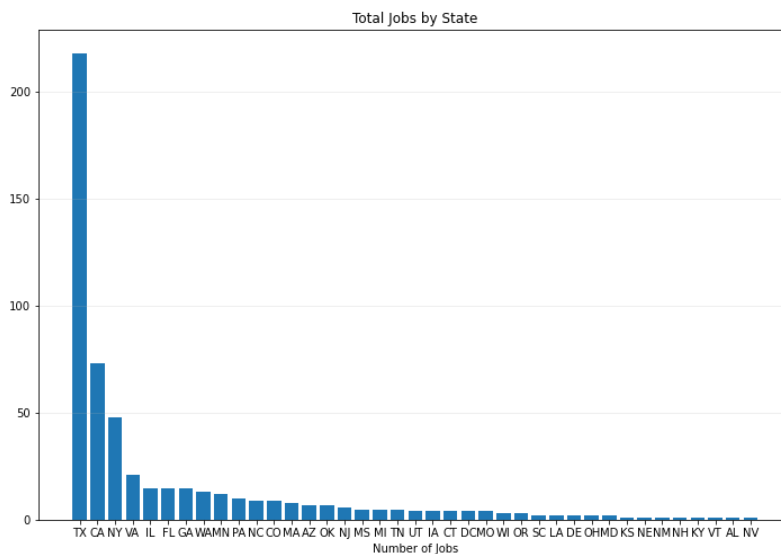
QueryQuery HistoryScratch Pad x

```
1 select
2   st.state,
3   count(j.id) as total_jobs
4 from
5   job j
6   join state st on j.state_id = st.id
7 group by
8   st.state
9 order by
10  total_jobs desc;
```

Data OutputMessagesNotifications

	state character varying (10)	total_jobs bigint
1	TX	173
2	CA	67
3	NY	58
4	IL	21
5	VA	20
6	GA	19
7	CO	14
8	WA	12

Based on the database and tables we created, we decided to run a query to count how many job postings for each state are listed on the first 50 pages of the Indeed website. First, we merged the two tables, “state” and “job,” by joining the “state id” column (in the “job” table) and “id” column (in the “state” table), group by state and sort by “total jobs” with the order by syntax. According to the analysis, there are 43 states shown in the database. Texas had the most at 173, followed by California at 67 and New York at 58.



## Remote Jobs by State

Since the pandemic started in 2020, telework (remote work) has expanded sharply, and remote-friendly jobs have become more common among both listings and searches, according to job-search websites. And as the data market grows and remote work continues to rise, “Data Analysts” will increasingly find opportunities for flexible work, including remote work. Thus, our team analyzed the database further to identify which state offered more remote work. First, we ran a query to join 2 tables, “state” and “job,” group by state, count the number of remote jobs for each state listed on the database, then presented the data with the bar chart below.

Project2\_job\_board/postgres@PostgreSQL 15

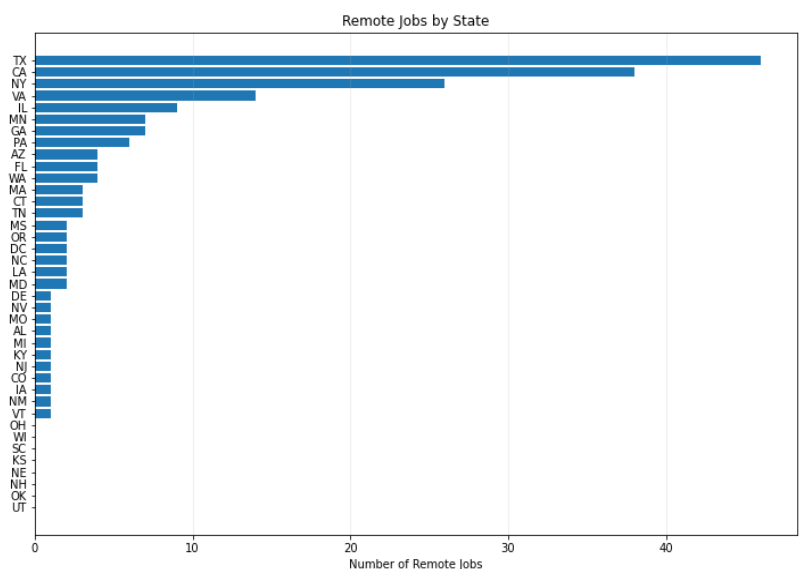
Query Query History

```
1 select
2   st.state,
3   sum(j.is_remote::int) as remote_jobs
4 from
5   job j
6 join state st on j.state_id = st.id
7 group by
8   st.state
9 order by
10  remote_jobs desc;
```

Data Output Messages Notifications

	state character varying (10)	remote_jobs bigint
1	CA	40
2	TX	37
3	NY	25
4	VA	14
5	IL	12
6	GA	8
7	PA	7
8	WA	6

Total rows: 42 of 42 Query complete 00:00:00.045



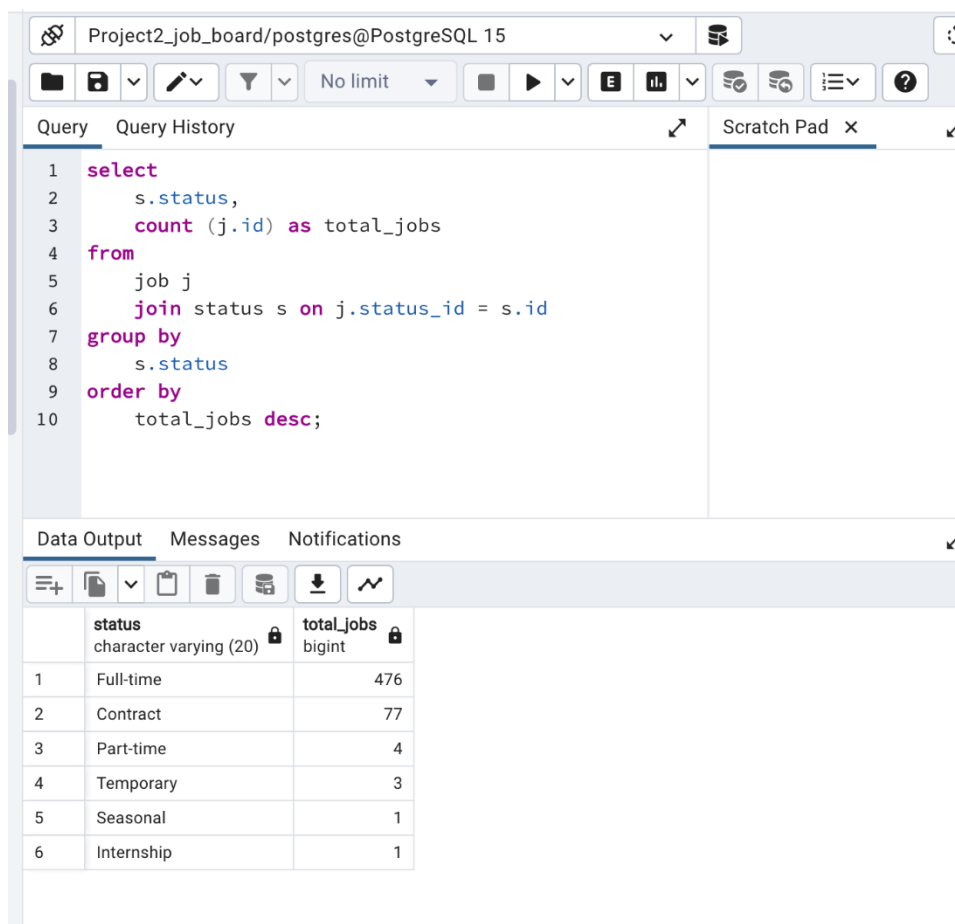
Among 43 states listed in the database, California has offered the most remote “Data Analyst” jobs at 40, followed by Texas at 37 and New York at 25. Of the 67 job postings from California, 60% are remote. 21% and 43% of Texas and New

York postings are remote, respectively. Other states had much fewer remote jobs; some jobs only had 1.

## Employment status

During the ETL process, we were curious to learn about the employment status of the positions listed on the job postings we looped through. So, we merged the two tables, “states” and “job,” via join, group by status, and sort by the number of jobs.

Not surprisingly, 85 % of the jobs posted are full-time. While the contract or part-time could be faster to find and more affordable than bringing on permanent(full-time), it may only work when the need is occasional or short-term. Full-time would be more permanent, able to learn the ins and outs of the company, more convenient when it comes to communications and meetings, and benefit the company with long-term needs.



The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is to 'Project2\_job\_board/postgres@PostgreSQL 15'. The query editor displays the following SQL query:

```
1 select
2     s.status,
3     count (j.id) as total_jobs
4 from
5     job j
6 join status s on j.status_id = s.id
7 group by
8     s.status
9 order by
10    total_jobs desc;
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query in a table format. The table has two columns: 'status' (character varying (20)) and 'total\_jobs' (bigint). The results are as follows:

	status	total_jobs
1	Full-time	476
2	Contract	77
3	Part-time	4
4	Temporary	3
5	Seasonal	1
6	Internship	1

## Jobs by company

Besides analyzing data by state, we also investigated data by company. We merged the 3 tables, “company” and “job” on the company id column and “status” and “job” on the status column, to get the number of full-time jobs that each company is hiring. We came up with 421 companies, and the company that has the most postings is Tesla. We created a bar chart of this data; unfortunately, the chart is not presentable due to the high number of companies.

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

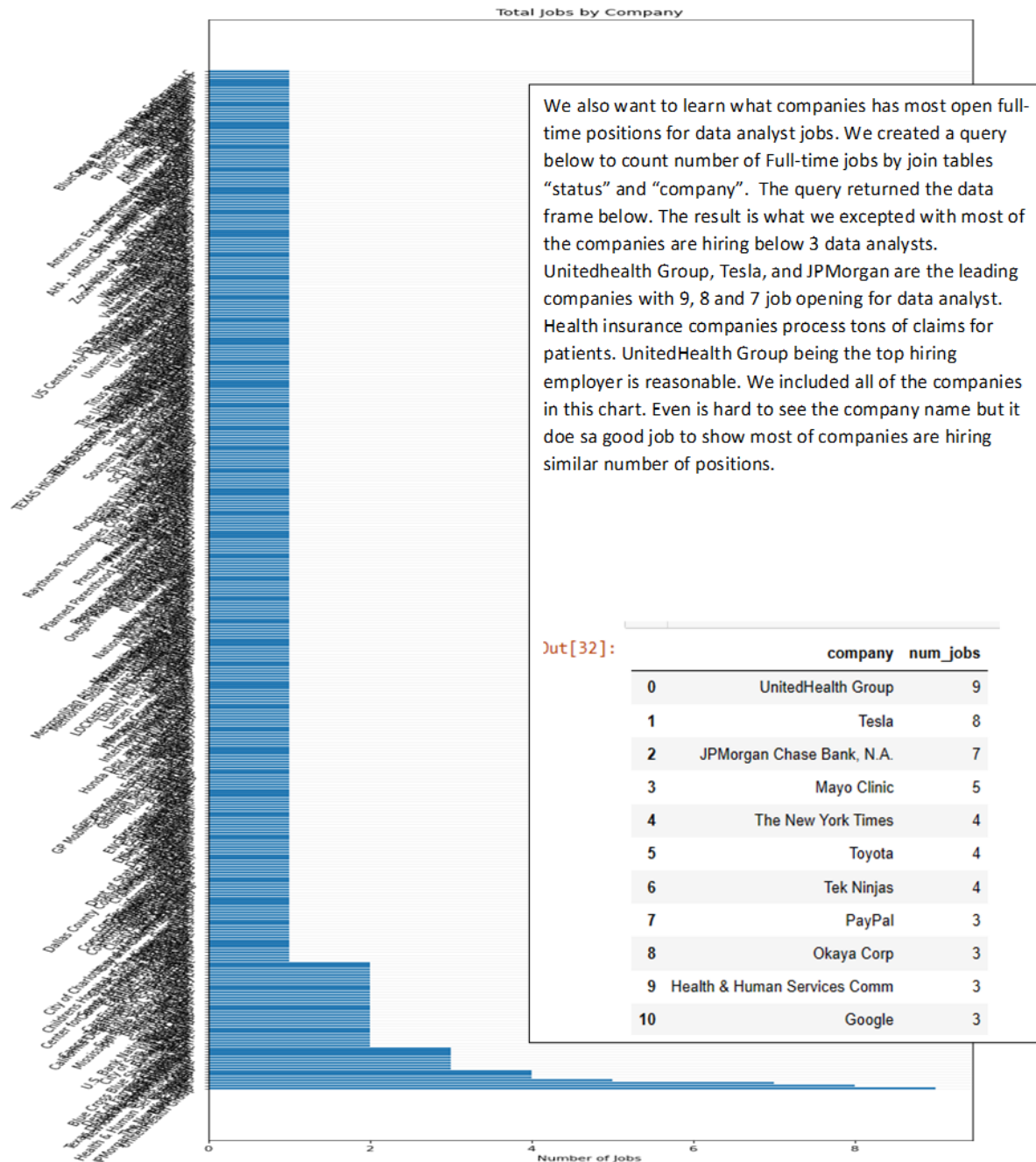
```
1 select
2     c.company,
3     count (j.id) as num_jobs
4 from
5     job j
6     join status s on j.status_id = s.id
7     join company c on j.company_id = c.id
8 where
9     s.status = 'Full-time'
10 group by
11     c.company
12 order by
13     num_jobs desc;
```

The Data Output tab shows the results of the query. The table has two columns: 'company' (character varying (100)) and 'num\_jobs' (bigint). The results are ordered by the number of jobs in descending order.

	company	num_jobs
1	Tesla	6
2	UnitedHealth Group	5
3	The New York Times	4
4	Tek Ninjas	4
5	Vetro Tech Inc	3
6	Tarrant County, TX	3
7	PayPal	3
8	KesarWeb	3

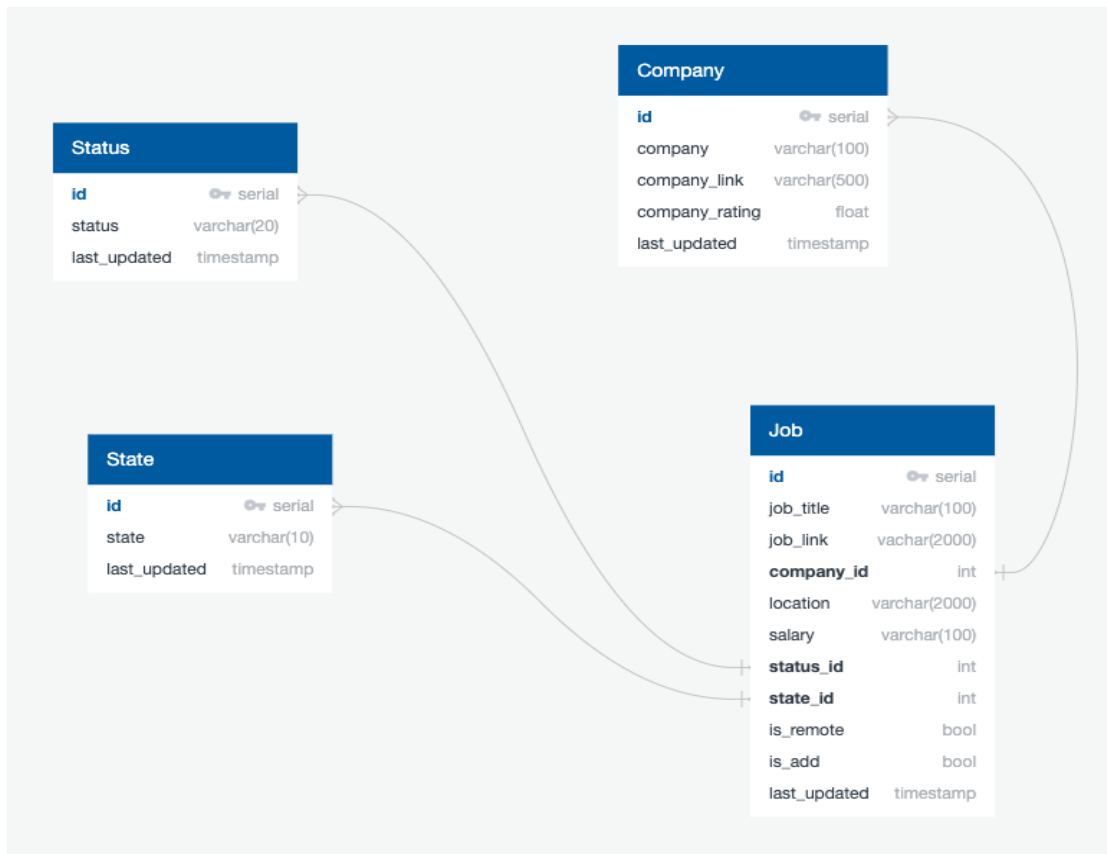
Total rows: 421 of 421    Query complete 00:00:00.057    Ln 6, Col 27

## Total Jobs by Company discrepancies



### Schema Description

We created our Job, Status, Company, and State table using quickdatadiagrams.com. With our Job table being the primary table, we created then fed into it our Status, State, and Company table. Next, we moved to identify our primary and foreign keys in each table. Specifically, we fed our unique serial, state, and company id columns into the unique id columns in our Job table. Initially we ran into a few issues after we exported it to PgAdmin. For instance, we realized a few of our column data types had too short of character count which brought up an error in our database, specifically our Job Links column VARCHAR character count was changed from 500 to 2000. Lastly, we ran into the issue of having incorrect Null/Not Null values specifically for our Company Rating, Company Link, state\_Id, is\_remote, and salary columns. After making these corrections to the schema we ran another export to PgAdmin with success where we were able to run successful search queries as well from our tables.



### Errors, Advanced code, and Attempts

Within our Data Frame we tried to achieve pulling our city locations for each listing and found it to be difficult due to the varying formats per listing. Most have special characters, full spelling, or an abbreviated state, etc. Our goal for this was to look at a specific state that generates the highest amount

```
2 # def getCity(loc):
3 #     if "," in loc:
4 #         city_list = loc.replace("\xa0", " ").split(",")[0].strip().split(" ")
5 #         city = city_list[0] if len(city_list) == 1 else city_list[1]
6 #     else:
7 #         city = None
8
9 #     return(city)
```

of job listings and locate which city has the most jobs. We would like to circle back to this once we have more time to clean our

data.

Our states data needed cleaning to be able to compare jobs to states. Some states had white spaces before and after commas or no white spaces where a space should be. To correct and replace these we used a unique piece of code shown here. Using ".replace("\xa0", " ") achieved the correction of non-

```
1 #State
2 #We had to use .replace(-- ) due to commas and white spaces to correct the format of our states
3 def getState(loc):
4     if "," in loc:
5         state = loc.replace("\xa0", " ").split(",")[1].strip().split(" ")[0].strip()
6     else:
7         state = None
8
9     return(state)
```

breaking spaces and replace them with a space. We then duplicated that with the remaining code

to remove extra spaces and missing commas.

The Data Frame for Status proved to be the most tasking to clean. RegEx was used to within Status due to the type of employments have plus one or even two at times. We need to merge these together to

```
1 #combine with regex s=space d=catch or match all numbers \+ = plus sign matches plus sign, regex meaning.
2 df["status"] = df.status.str.replace("\s\+\d", "")
```

avoid showing

duplicates.

Due to the format of salary on indeed.com having a high variance we were not able to use it in any comparisons. Ideally, we would like to separate out hourly and annually, correct the formats of both and compare the highest paying job per both hourly and annually.

In conclusion, there is more cleaning needed to be able to take a deeper dive into which state holds the most demand for the Data Industry jobs currently. We achieved the main purpose of our project, to see which state is in the highest demand, which is Texas.