Aadi Sudan

CS 185 Final Project Proposal

Project Title: Analyzing Job Market Trends

Scraped webpage (the dataset): https://github.com/SimplifyJobs/New-Grad-Positions

Throughout the modern hiring landscape, job seekers face a challenging question: what skills, degrees, and experience do companies truly require for early-career technical roles? Job descriptions vary widely in structure, clarity, and terminology, making it difficult to extract consistent expectations and frustrating when an application is rejected without a clear-cut reason why. This project set out to analyze these requirements at scale by mining skill patterns from software and data-related job postings hosted on the Greenhouse applicant tracking system. The goal was to identify the most commonly requested qualifications using Regex and uncover deeper relationships between qualifications and job types using association rule mining. To reach this point, the project underwent a long series of failed attempts, redesigns, and incremental improvements before landing on the final implementation which showcased promising results.

Naturally, the first - and most challenging - step in the project was data collection. The easy choice would be to take a Kaggle dataset and use that as a basis for association rule mining, but data in the real world is rarely as neatly structured as Kaggle. A more effective and realistic solution was to design a web-scraper to source online job portals and collect data in real-time. The initial hopes for the project were to collect data from the most prominent job portals - LinkedIn, Glassdoor, Indeed, etc. However, the first hurdle showed itself when attempting to design a web-scraper to collect postings from these sites - many professional job boards have built-in anti-scrape software that blocks any unauthorized acquisition of their data. This effectively shuts down any attempts to automate data collection on these sites, which made it incredibly difficult to proceed. Fortunately, an alternative solution presented itself in Simplify.

Simplify is a job aggregation platform designed specifically for students, new graduates, and early-career professionals seeking internships and entry-level roles. Among their many services is a Github repository that collects new grad and internship postings directly from company pages, LinkedIn, and more, and stores them in a unified, easy-to-access space. Scraping this repository proved to be a far more feasible challenge than attempting to bypass more colloquial job portals' anti-scraping software. I designed my web-scraper to specifically access the repository's listings.json file, where information about company name, job title, and link to application were readily available in a json format that was easy to access and parse through. This scraper could be run on a daily basis to continually refresh the latest postings, and it automatically filters all scraped positions to focus only on the ones that are still active. Upon running the scraper in the most recent attempt, 2427 active job postings were collected.

The next step was to access the job postings on file. However, this is where a new issue presented itself: jobs are not posted in a singular, standardized format. Depending on whether the job is on Greenhouse, Workday, or a proprietary application page, the qualifications and details will be located in a different place and in a different format, in some instances under completely different headers. This made it incredibly difficult to actually collect qualification data on various jobs. After much experimentation, I decided to narrow our list down to only applications posted to one site: Greenhouse. This site proved to be the easiest to collect data from when experimenting with others such as Workday, and most postings on Greenhouse shared the same general format and headers, making it easy to design a universal function that could collect relevant data from any company's posting. Upon filtering our dataset, we came away with 302 postings on Greenhouse, a massive hit to our dataset but a necessary one to make the rest of the project much more feasible, and still enough data to come away with association rules we could deem relevant and usable.

The next major challenge involved extracting the actual job requirements from these postings. At first, the approach was straightforward: search for <li> tags on the page and extract their text. Many job descriptions list their qualification requirements in bullet form, making this method effective in a portion of cases. However, this method quickly revealed its limitations. Some postings embed the entire job application form within <li> tags, including fields like “Upload resume,” “Phone number,” or internal legal disclaimers. Others contain bullet points unrelated to job requirements, such as values statements or employee benefits. This polluted the extraction process greatly.

A second attempt used keyword-based regex slicing, extracting text occurring between phrases such as “Requirements,” “Minimum Qualifications,” and “Preferred Qualifications,” ending at markers like “About,” “Benefits,” or “Compensation.” This method worked well for posts that adhered to conventional section headings but failed when companies used less standard phrasing such as “What you’ll need,” “What you bring,” or “Core competencies.” Furthermore, a significant number of postings combined multiple sections together without clear boundaries, making regex slicing unreliable across the dataset.

The final working solution combined both approaches. The function first attempts to extract bullet points containing keywords like “experience”, “degree”, and the names of various programming languages. If this fails, it performs a broader regex extraction window between any known requirement-like terms. This hybrid method significantly improved extraction success while maintaining high precision. Although several postings still produced empty outputs - primarily those that didn’t utilize a bullet-point or header-based job description and instead simply provided 2-3 paragraphs of natural language - the majority yielded clean and usable requirement text. The reason we couldn’t scrape natural language was due to the variety of language used - designing a Regex call to look for any instance of one of 50 different programming languages is a very daunting task.

Once extracted, requirement text underwent a normalization pipeline. The text was lowercased, stripped of Unicode artifacts, and cleaned to remove newline characters and irregular spacing. This produced a consistent text corpus suitable for pattern extraction. The project then focused on three core variable categories: programming languages, academic degree requirements, and minimum years of experience. These were selected because they are both commonly listed and highly relevant to job seekers, making them ideal for quantitative analysis and association rule mining.

Programming languages were detected using a library of Regex patterns for fifteen different languages. In the future, more could be added to this library, but the 15 most popular ones were chosen. Degree searching focused on three primary categories: Bachelor's, Master's, and PhD. These were much easier to find after stripping away any unwanted punctuation. Finally, years of experience were determined specifically by looking for the phrase "years" (+ an optional "of experience") and any slight alterations to it, and focusing on the number preceding it.

Following this preliminary term search, Python emerged as the dominant programming language with 141 mentions, followed by SQL (41), Java (34), C (31), and R (26). This distribution aligns closely with industry norms, with Python typically seen as a necessary universal language while SQL and R appear more in data-oriented roles. Degree requirement extraction revealed that Bachelor's degrees were mentioned 128 times, Master's degrees 36 times, and PhDs 16 times. This surprised me a bit due to how many positions in the data world I've encountered that require a minimum of a Master's degree. I suspect many of those positions weren't targeted by our algorithm due to not being on Greenhouse. Experience extraction showed that many positions require one to two years of experience with 26 postings requiring one year and 35 requiring two.

	Language	Count
0	python	141
1	sql	41
2	java	34
3	c	31
4	r	26
5	javascript	24
6	go	18
7	typescript	14
8	scala	6
9	kotlin	3
10	rust	3

	Degree	Count
0	bachelor	128
1	masters	36
2	phd	16

MinExperience	
1	26
2	35
3	4
4	2
5	8
10	1
13	3
23	2
25	3

To deepen the analysis beyond frequency counts, each job posting was transformed into a “transaction” consisting of its extracted languages, degrees, experience level, and role tags derived from its title (e.g., data roles, software engineering roles, full-stack roles, and machine learning roles). Association rule mining was then applied to discover relationships between these features. The following rules were discovered after identifying frequent itemsets and using the Apriori algorithm.

Generated 28 association rules.

	antecedent	consequent	support	confidence	lift
24	LANG_r	LANG_python, ROLE_OTHER	0.066225	0.769231	4.223776
16	LANG_java	LANG_python, ROLE_SWE	0.086093	0.764706	3.724858
20	LANG_javascript	LANG_python, ROLE_SWE	0.052980	0.666667	3.247312
1	LANG_go	ROLE_SWE	0.056291	0.944444	2.546627
17	LANG_java, LANG_python	ROLE_SWE	0.086093	0.928571	2.503827
13	LANG_java	ROLE_SWE	0.096026	0.852941	2.299895
21	LANG_javascript, LANG_python	ROLE_SWE	0.052980	0.842105	2.270677
0	LANG_javascript	ROLE_SWE	0.066225	0.833333	2.247024
3	DEG_masters	DEG_bachelor	0.109272	0.916667	2.162760
12	LANG_sql	LANG_python	0.132450	0.975610	2.089604
5	LANG_r	LANG_python	0.082781	0.961538	2.059465
26	LANG_r, ROLE_OTHER	LANG_python	0.066225	0.952381	2.039851
27	DEG_bachelor, LANG_sql	LANG_python	0.066225	0.952381	2.039851
19	DEG_masters, LANG_python	DEG_bachelor	0.062914	0.863636	2.037642
18	LANG_java, ROLE_SWE	LANG_python	0.086093	0.896552	1.920274
7	LANG_go	LANG_python	0.052980	0.888889	1.903861
23	DEG_bachelor, LANG_java	LANG_python	0.052980	0.842105	1.803658
9	LANG_java	LANG_python	0.092715	0.823529	1.763872
22	LANG_javascript, ROLE_SWE	LANG_python	0.052980	0.800000	1.713475
2	LANG_javascript	LANG_python	0.062914	0.791667	1.695626

One of the clearest findings is the central role of Python. It appears repeatedly as the predicted consequence of many rules, which means that when a posting lists other languages such as Java, JavaScript, SQL, Go, or R, it is very likely to also require Python. For example, the rule “SQL implies Python” has a confidence close to 0.98, meaning almost every posting asking for SQL also asks for Python. Similarly strong patterns appear for R, Java, and JavaScript. The fact that these rules also show lift values greater than 2 indicates that these relationships are much stronger than random chance. This suggests that Python functions as a kind of anchor language in the early-career job market, one that employers expect even when

the role emphasizes another ecosystem. This lines up accurately with the general consensus around which language is most important to learn and the emphasis placed on Python.

The generated rules also show relationships between certain languages and specific role types. The programming language Go, for example, has a very strong association with software engineering roles, with a confidence over 0.94. Other languages such as Java and JavaScript also tend to appear with software engineering roles, though with slightly lower lift. On the education side, Master's degree requirements curiously do not appear too often, though I expected them to be especially prevalent in data roles. After running a separate test, I found that this dataset only had 68 data-related roles, and upon comparing the ratio of those roles with Master's degrees to the ratio of all roles with Master's degrees, I did find that data-related roles are about 48% more likely to require Master's degrees, but the dataset was incredibly small and likely not enough to make any concrete assumptions.

These findings confirm well-known trends in the industry but quantify them with real data from live postings. They also demonstrate the value of association rule mining for revealing co-occurrence patterns that would be difficult to observe manually. More importantly, the project uncovered how variable job description formatting can be, and how brittle many extraction techniques can become in real-world settings.

The largest limitation of this project was scope. Being unable to access other job portals and needing to filter exclusively to Greenhouse postings (and further, needing to only look at postings that returned our Regex search of requirements) greatly limited the dataset, and an avenue for improvement in the future would be to come up with a more robust scraping function that could collect data from other application portals and take into account the many, many edge cases with regards to qualification wording. In my research, for example, I found that a Selenium-based scraper would allow dynamic rendering of job postings, likely improving

extraction for poorly formatted pages. Additionally, an idea for a future implementation I had was to allow a user to upload their own resume. Then, a function could scrape their resume for their own skills, experience, and degree and compare to the most frequently-recurring industry wants to highlight avenues for improvement or even how closely their skillset matches a given job description.

Despite these limitations, the project demonstrates an end-to-end system for mining real-world job data. It documents not only the final working method but also the numerous failed approaches that ultimately shaped the correct solution. Through this process, strong and interpretable industry patterns were extracted, offering actionable insights to job seekers aiming to understand the evolving expectations of the tech job market.