Project: Executive Report - John Woodward

Exploring Job Market: Insights and Data for Informed Career Development Steps

After feeling unfulfilled in my current career, I decided to pursue a graduate program in Data Science last fall of 2022 to shift careers permanently. However, I must be proactive because my company plans to lay me off on May 5. Thus, I used this project to research the job market by crawling a Wikipedia table and LinkedIn job search results. In this project, I aimed to gain valuable insights into the jobs that interest me (e.g., linear programming, Python) and take informed career development steps.

The data used for this project came from four sources:

1. A Wikipedia page accessed on 2023-04-21 provided a ranked list of U.S. states by their relative U.S. dollar purchasing power to consider differences in the cost of living between states.
2. Another Wikipedia page accessed on 2023-04-21 listed all 50 states with their two-character abbreviations (e.g., Massachusetts: M.A.).
3. A txt file extracted on 2023-04-29 from a zip file (e.g., "US.zip") from a world location database.
4. I used a government URL to obtain the shp files needed for the U.S. state visuals.
5. Job search results accessed from LinkedIn on 2023-04-11 using specific search criteria and a browser plug-in (e.g., Lix) to gather high-level meta-data for job listings.

Due to the significant bandwidth required to scrape hundreds of job descriptions manually, and the website's prohibition on automated scraping, I hired a Fiverr contractor. They competently extracted 2,414,329 characters of lower-level job description data, thus, proving to be a resourceful and fair solution to the ethical dilemma.

It is important to note that the data may have limitations or biases. For instance, Wikipedia data may have inaccuracies, though all 50 states joined adequately. Another example is that the job search results only reflect a specific and non-random sample of job opportunities. Additionally, using a browser plug-in (e.g., Lix) to gather metadata for job listings may have skirted terms of service for LinkedIn. However, it was believed acceptable in this case based on the additional regulation context provided in the data_provenance.pdf.

Overall, the data sets used for this project give an excellent starting point for informed career development steps. Therefore, to help with that, I chose the following exploratory questions:

1. Where are most non-remote jobs located, and given employers who provided salary ranges: what are the salary bands by state, adjusting for purchasing power?
2. What proportion of job listings reference each degree level (e.g., Ph.D., Master, Bachelor)?

Answering these questions is crucial for my goals as a data scientist:

- Clarity: I can simplify my job search by visualizing available data science roles near my family and categorizing them according to my values. This approach increases my chances of finding a fulfilling job, such as creating a geopandas map to show relevant positions within 120 miles of my family in New England, including remote or New England-based jobs requiring a Master's degree.

- Strategy: Identifying the top 10 skills for my dream data scientist job will help me plan what skills to emphasize and become a more competitive candidate. I can use verify_keywords_flashtext to identify the top skills (e.g., ['machine learning', 'random forest', 'NLP', 'ML']), then emphasize my skills or eagerness to learn them.
- Confidence: I can optimize my resume and LinkedIn profile with relevant keywords for search engine optimization (SEO) by researching the current keyword trends for my dream roles. This approach will increase my chances of opportunities and shorten my job search, allowing me to interview and network confidently and efficiently.

Exploratory Data Analysis

The primary dataset was the job search result; the following shows the data:

- ID: Unique identifier for each job listing.
- Link: URL link to the job posting.
- Title: Job title for the position advertised.
- Salary_Combined: Combined salary information for the job, including any specified salary range rates (e.g., $90.5K/yr - $200K/yr, $21/hr - $27/hr). I manually scrapped this; there were few provided salary ranges.
- Location: Geographic location of the job (e.g., Boston, MA; Greater Boston; New York, United States)
- Workplace type: Type of workplace (e.g., remote, on-site, hybrid).
- Description(fivrr): Job description text obtained through manual data collection from a Fiverr contractor.

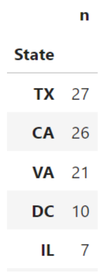
To answer question 1, I had to do the following:

1. In the gather.py - I used a web crawl for purchasing power by the state and merged it by the two-character U.S. state names (e.g., W.I., MA). This was very straightforward and only required pandas read_html with minimal cleanup for some references in the states on one of the Wikipedia tables. Therefore, I opted to use a function for using read_html from Wikipedia for future use.
2. Parse and clean the City and State data from the Location. Specifically, I chose to turn any remote jobs in City and State into None. Likewise, the locations had mild inconsistencies, so I mopped up the exceptions using masking and boolean expressions to quickly change the pandas' dataframes.
3. To extract salary information from 'Salary_Combined', regular expressions and string manipulation were used to parse yearly and hourly wages. The yearly format, such as $90.5K/yr - $200K/yr, required multiplication by 1000 for 'K'. For hourly wages in the format of $21/hr - $27/hr, I converted it using 40 hours per week and 52 weeks per year. Ultimately, I parsed it into three columns: Salary_Lower, Salary_Upper, and Salary_UOM.
4. Finally, I created a function, get_salary_bands_by_state, to allow significant customizability in the future for aggregate analysis given an input of a salary and purchasing power. I joined the

left and right tables and allowed customization on column names, aggregation methods (e.g., median, average), and whatnot.

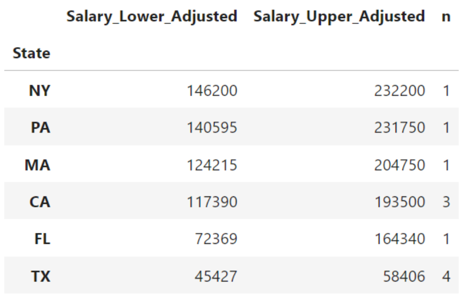
5. I did a much simpler analysis for the non-remote jobs: I masked the non-remote jobs, aggregated them by state, and then grouped it so the count was descending from most to least.

Because of the analysis, here are the outputs to answer question one:



State	n
TX	27
CA	26
VA	21
DC	10
IL	7

The analysis reveals the top five states for non-remote jobs. Notably, Texas, California, and Virginia emerged as the top three states, which accounted for an outsized share of on-site or hybrid job postings.

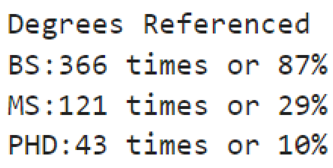


State	Salary_Lower_Adjusted	Salary_Upper_Adjusted	n
NY	146200	232200	1
PA	140595	231750	1
MA	124215	204750	1
CA	117390	193500	3
FL	72369	164340	1
TX	45427	58406	4

The salary ranges adjusted by cost of living were also examined, with New York, Pennsylvania, and Massachusetts ranked at the top. Interestingly, the analysis showed that Florida and California were lower on the list than expected. However, it is worth noting that Texas appeared down on the list due to limited wage information, with only an internship offering a salary range of $21/hr to $27/hr.

To answer question 2 (i.e., what proportion of job listings reference each degree level?), I had to do the following:

1. To check proportions, I needed to classify which jobs mentioned the specific degree levels; thus, it required me to use word searches. Specifically, my online research suggested that flashtext is a better method for word search than regular expressions.
2. Therefore, I used verify_keywords_flashtext to check whether keywords matched each degree.
3. The keywords I used for each degree were "bachelor's", "b.s.", "bs", "bachelor", and "bachelors" for bachelor's degree; "master's", "m.s.", "ms", and "masters" for Master's degree; and "phd", "ph.d.", and "doctorate" for Ph.D.
4. Also, I replaced smart quotes with straight single-quotes to avoid losing word search counts.
5. Afterward, I used an f-string to display the results like so:



```
Degrees Referenced
BS:366 times or 87%
MS:121 times or 29%
PHD:43 times or 10%
```

Deeper Analysis

To answer two further questions for my job search:

1. "What positions are within a reasonable distance from family? (e.g., 250 miles)."
2. "What are the top skills desired for my dream job?"

Task 1: Adding MilesFromHome Column and Plotting Job Locations

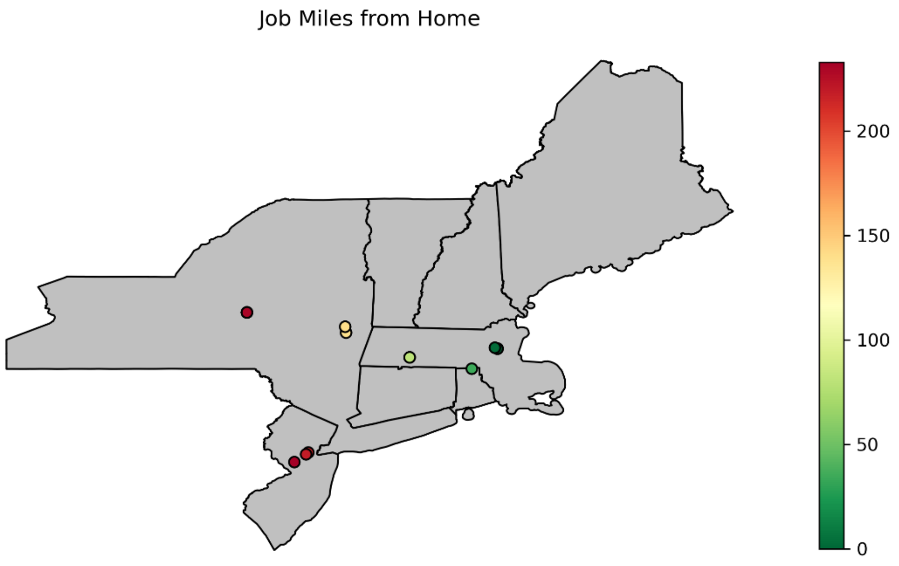
Subtask 1.1: Adding MilesFromHome Column

The first subtask involves adding the MilesFromHome column to the main dataframe. First, the code reads in an open-source city and state latitude and longitude dataset and cleans the data to remove duplicate rows. Then, it merges the latitude and longitude onto the primary dataframe, df. Note that the code expects the US.txt file to be present, as it is manually downloaded from a US.zip file and saved in the project directory.

Next, the code defines a function that calculates the distance between two locations in miles using the geodesic function and the latitude and longitude of each location. The function also includes error handling to ensure the correct measurement unit. Then, curry it with a lambda function to continuously measure the distance from 'Boston' & 'M.A.,' and finally, it calculates the MilesFromHome column and filters jobs within 250 miles of Boston, MA.

Subtask 1.2: Plotting Job Locations

After adding the MilesFromHome column, the code plots the locations of the jobs on a U.S. map using geopandas and matplotlib. To do this, it creates a GeoDataFrame from the jobs data, reads in a shapefile of the U.S. states, and filters the states to include only New England states. Then, it maps the states in the New England region and plots the jobs as points on the map.



As only some non-remote jobs are in the area, only looking for New Hampshire jobs may be a waste of time. Keep your options open and be on the lookout for any type, but expect primarily remote (i.e., 66% of the jobs)

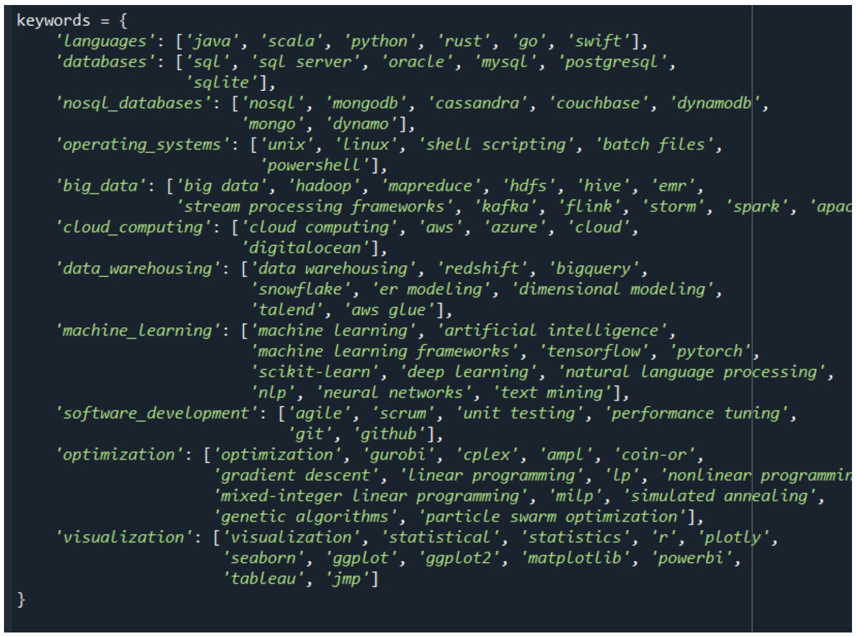
Task 2: Classifying Skills

Subtask 2.1: Categorizing Keywords

The first subtask involves classifying skills by manually creating a simple random sample of ten jobs and identifying skills-related keywords. This manual process outputs the keywords in a dictionary format, which saves them as keywords. The code categorizes the keywords into languages, databases, big data, and machine learning.

Subtask 2.2: Searching for Keywords

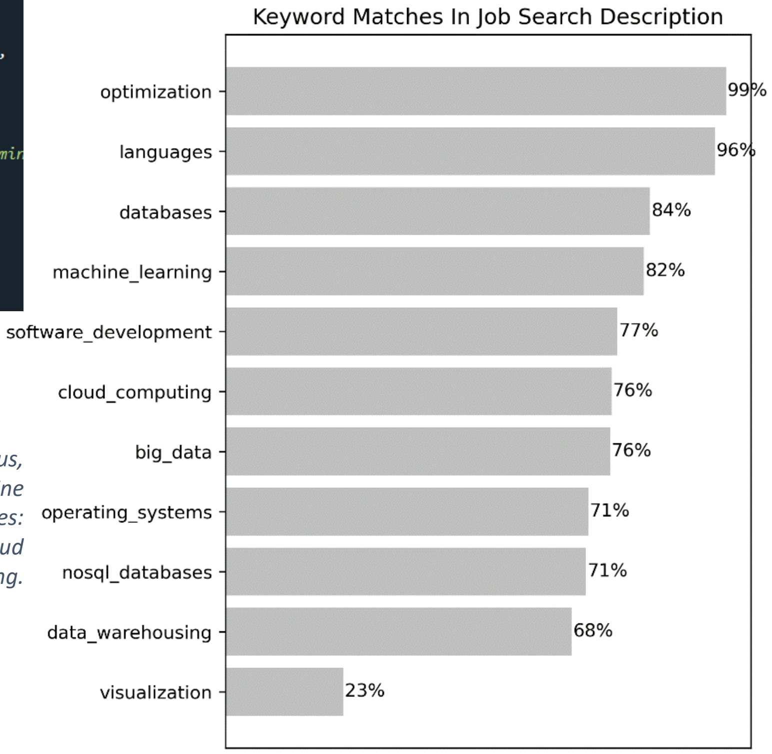
The second subtask involves searching for keywords in the job descriptions and creating new columns in the DataFrame with the name 'keyword_{category}' for each category of keywords. To accomplish this, the code loops through each type of keyword, removes any slashes in the job descriptions, and performs a keyword search.



```
keywords = {
    'languages': ['java', 'scala', 'python', 'rust', 'go', 'swift'],
    'databases': ['sql', 'sql server', 'oracle', 'mysql', 'postgresql',
                  'sqlite'],
    'nosql_databases': ['nosql', 'mongodb', 'cassandra', 'couchbase', 'dynamodb',
                        'mongo', 'dynamo'],
    'operating_systems': ['unix', 'linux', 'shell scripting', 'batch files',
                          'powershell'],
    'big_data': ['big data', 'hadoop', 'mapreduce', 'hdfs', 'hive', 'emr',
                 'stream processing frameworks', 'kafka', 'flink', 'storm', 'spark', 'apac
    'cloud_computing': ['cloud computing', 'aws', 'azure', 'cloud',
                        'digitalocean'],
    'data_warehousing': ['data warehousing', 'redshift', 'bigquery',
                         'snowflake', 'er modeling', 'dimensional modeling',
                         'talend', 'aws glue'],
    'machine_learning': ['machine learning', 'artificial intelligence',
                         'machine learning frameworks', 'tensorflow', 'pytorch',
                         'scikit-learn', 'deep learning', 'natural language processing',
                         'nlp', 'neural networks', 'text mining'],
    'software_development': ['agile', 'scrum', 'unit testing', 'performance tuning',
                             'git', 'github'],
    'optimization': ['optimization', 'gurobi', 'cplex', 'ampl', 'coin-or',
                     'gradient descent', 'linear programming', 'lp', 'nonlinear programmin
                     'mixed-integer linear programming', 'milp', 'simulated annealing',
                     'genetic algorithms', 'particle swarm optimization'],
    'visualization': ['visualization', 'statistical', 'statistics', 'r', 'plotly',
                      'seaborn', 'ggplot', 'ggplot2', 'matplotlib', 'powerbi',
                      'tableau', 'jmp']
}
```

Subtask 2.3: Summarizing and plotting

Next, the code summarizes the keyword matches in the job descriptions and prepares the plot. After, it filters out any 'keyword_' to clean it, pivots the filtered DataFrame, renames the columns, and sorts the resulting DataFrame by count—finally, plotting bars for each category's most to least important.



The first three coincide with the job search filter, thus, are misleading. Instead, I should focus on search engine optimization (SEO) for the top three keyword categories: machine learning, software development, and cloud computing.

Future or Open Questions

1. "What other job titles have similar responsibilities and requirements to a Data Scientist position, and how can we use TF-IDF to measure the similarity of their job description texts?"
    - While researching, I learned about TF-IDF and its benefit in discovering insights. For example, we could use it to compare job titles and see how similar they are to positions like Data Scientist. This could be an exciting place to explore.
2. "How can you compare your resume with job descriptions and find the jobs that best match your skills and experience?"
    - By comparing my resume with the job descriptions of the positions I am interested in, I can prioritize and focus my job search on the most relevant openings. This approach would enable me to save application time and use my expertise more effectively. However, it may require natural language processing and advanced Python skills.