Job Market Analysis

Supriya Jayaraj

University Freiburg, Freiburg, Germany

1 Introduction

The purpose of this project is to analyze the job market for data-related positions from all around the world. This report thoroughly analyzes a large dataset containing information on many data-related positions such as data analyst, data scientist, and data engineer.

In the project's first part, the dataset is collected from the hugging face [3] and analyzed using Python to find key insights that benefit job seekers and employers in this field. It aims to identify the most in-demand skills, popular job locations, remote and on-site work opportunities, job posting timelines, geographical job distribution, relevant company details, salary rates, job types, and the availability of health insurance benefits.

The second part of this project is to visualize the collected data effectively. Various data visualization techniques using Python libraries, such as Matplotlib and Seaborn, are used to create clear, informative, and visually appealing representations. These visualizations will include various plots and potentially interactive elements to make the complex data more understandable.

2 Dataset

The dataset is collected from the URL[1]. It contains 786,000 Job-Market datasets from the year 2023 within a comma-separated file. Each row contains the job market details, which is explained in section 2.1. Notably, data for 2022 and 2024 have been recently removed from the publisher. The collected data from the URL was pre-processed to prepare the dataset for detailed data analysis, involving thorough data cleaning which is explained in section 4.

2.1 Dataset Description

The dataset contains multiple columns such as: `job_title (categorical)` denotes the title of the job; `job_location (categorical)` specifies the location of the job; `job_schedule_type (categorical)` indicates the job schedule type (full-time, part-time, etc.); `job_work_from_home (categorical)` indicates if the job allows remote work; `job_posted_date (Temporal)` records the

[1] The dataset can be found at: `https://huggingface.co/datasets/lukebarousse/data_jobs/blob/main/data_jobs.csv`

date the job was posted; `salary_year_avg (numeric)` represents the average annual salary; `company_name (categorical)` names the company offering the job; `health_insurance (categorical)` indicates if health insurance is provided; `job_skills (categorical)` lists the required skills for the job. It also consists of various other data that can be useful for future implementations.

3 Dataset Extraction

Dataset extraction was performed with the help of two essential libraries: `pandas` and `requests`. The `requests` library was used to download data from a specified URL[1]. The `load_data_from_url` function checks for the existence of the file and if not found, it sends an HTTP GET request to download the data, saving it as a CSV file. The `pandas` library is used for loading data from the CSV file into a DataFrame, enabling selective column filtration and the execution of diverse data pre-processing tasks. In the CSV file, each row contains details such as the abbreviated job title, detailed job title, job location, job type, remote status, search location, job posted date, yearly and hourly salary averages, company name, health insurance, and required skills. Data analysis was streamlined by collecting data from the URL[1] and pre-processing it.

4 Data Pre-Processing

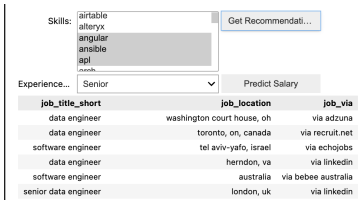
The pre-processing step is responsible for cleaning and transforming the data for analysis and visualization. For instance, it fills the column for missing values by using `fillna()`, which replaces `NaN` entries with default values such as `True` for boolean columns or `0` for numeric columns. The `pd.to_datetime()` function is used to standardize the formats of data, ensuring complete and analyzable data. The `ast.literal _eval()` function safely parses strings into Python lists for columns containing lists of skills. This step also extracts experience levels (entry, mid, senior) from job titles, categorizing job postings based on experience level. Furthermore, the code creates an abbreviated job title based on the extracted experience level using the `apply()` method combined with custom functions. Finally, the pre-processed and clean data is returned as a DataFrame to be saved as CSV files. This automated approach streamlines the data processing workflow, making the data more accessible for further steps. Due to the substantial volume of data and the code processing time, only 20,000 of the datasets were selected for further steps.

5 Data Analysis and Visualization

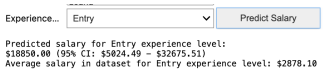
After obtaining the pre-processed data, a detailed data analysis was performed to understand the dataset more deeply. A report containing the analysis result was generated in HTML and JSON format. The analysis and model training of extensive data are done on the Jupyter Notebook to accommodate CPU usage. Matplotlib and Seaborn are utilized to visualize the data effectively.

5.1 Data Analysis

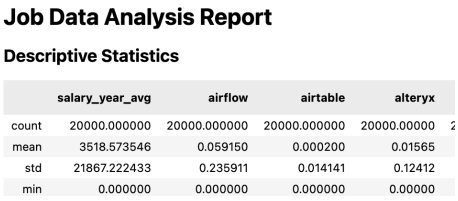
This section contains the full-scale analysis of the 786,000 datasets, which can be reduced to 20,000 or less to minimize runtime and RAM usage. This can be done using the function `jobs_data_sampled = jobs_data_preprocessed .sample(n=20000, random_state=42)`. Next, we use the `parse_skills` function, which efficiently parses the skills column, accommodating comma-separated formats. The `job_recommendation_system` function recommends jobs based on user-specified skills, calculating skill matches to highlight suitable job opportunities. Following this, `perform_comprehensive_analysis` conducts a full-scale analysis by computing descriptive statistics to summarize central tendencies and variability in numerical columns, assessing average salaries by location to reveal regional salary trends, and analyzing job counts by company to identify active employers. It evaluates the performance of a `RandomForestRegressor` model in predicting yearly salaries based on various job features. This function uses the `scikit-learn` Python library.



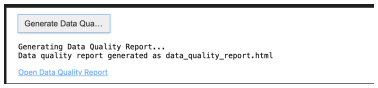
(a) Job Recommendation



(b) Predict Salary



(c) Analysis Report



(d) Quality Report

Fig. 1: Jupyter Widgets

The `create_interactive_widgets` function creates interactive tools for users as shown in Figure 1 to input their skill preferences and receive personalized job recommendations as you can see in Figure 1a or predict salaries as you can see in Figure 1b, while the `display_full_report` function formats and displays a detailed report as shown in Figure 1c. The outcomes of these analyses are saved in both JSON and HTML formats in the results folder, facilitating easy understanding and for further study. Additionally, the user can check the quality of the data for any missing values, categorical values, etc. by `perform_data_quality_checks` function. The user can click generate data

quality as shown in Figure1d; this generates a data quality report stored in `data_quality_report.html` in the results folder.

5.2 3D Plots

3D plots are created for the analysis of employment and salary data. This project uses two 3D plots, a 3D stacked bar plot and a 3D surface plot, to visualize average salaries by country and month and the distribution of job counts by title, location, and schedule type, respectively.

The 3D stacked bar plot shown in Figure 2 provides a visualization of job market segmentation across multiple dimensions, enabling the assessment of the prevalence of job titles in specific locations and the types of schedules associated with those roles.

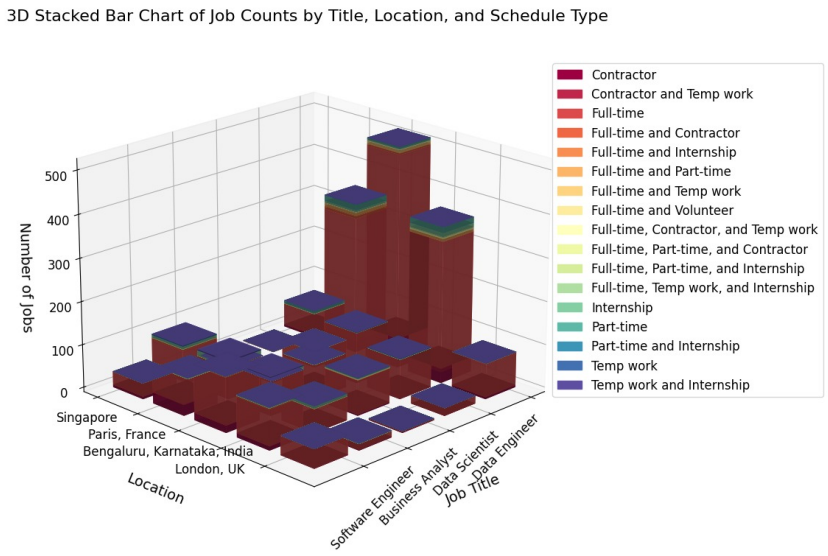


Fig. 2: 3D Stacked Bar Chart of Job Counts by Title, Location, and Schedule Type

On the other hand, the 3D surface plot shown in Figure 3 offers insights into salary trends over geographical and temporal dimensions. The visualization aids in identifying variations in average salaries, revealing seasonal or regional patterns in compensation levels. The colour gradient directly corresponds to salary amounts, with darker purple indicating the lowest salaries and bright yellow representing the highest salaries. The brightest yellow peaks easily identify the highest salary points across all countries and months.

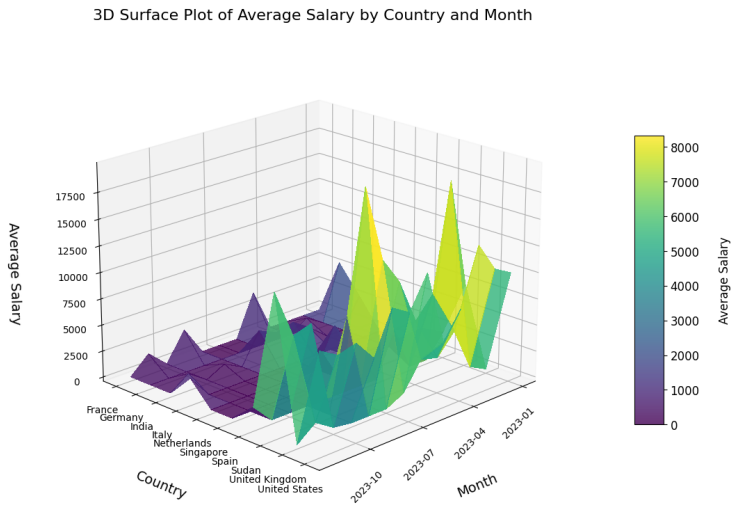


Fig. 3: 3D Surface Plot of Average Salary by Country and Month

5.3 World Map Plots

The interactive global map plot shown in Figure 4 provides an in-depth analysis of job locations, with filters for remote and onsite opportunities. The plot allows users to dynamically interact with remote and onsite jobs by selecting appropriate options across different geographic regions using the check box. This is done using a callback function `on_click` to update the plot based on user input.

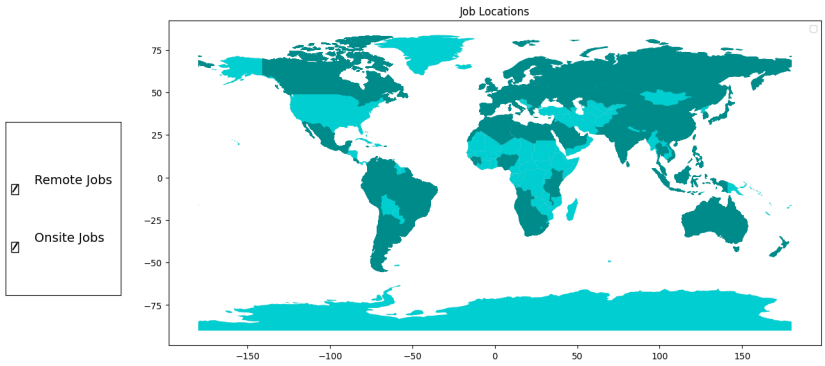


Fig. 4: Remote and On-site

The map colour-codes job locations based on specified filters to provide a clear visual representation of job distribution globally. Remote jobs are represented

with dark turquoise color and on-site are represented with dark cyan color. We can see that remote job opportunities are widely dispersed, while onsite positions tend to cluster in urban centres or specific regions.

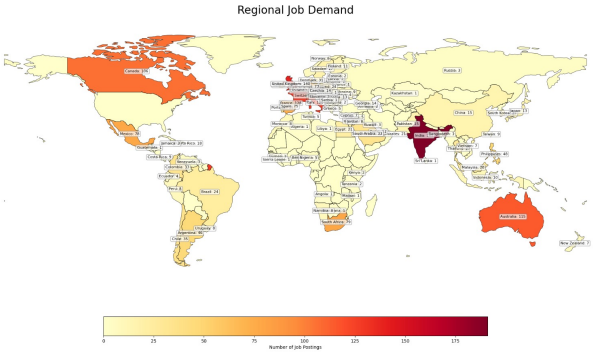


Fig. 5: Regional Job Demand

The second world plot, as shown in Figure 5, uses a choropleth map to depict geographical variations in job demand globally and regionally, colouring regions based on the intensity of job postings. The `cmap=ÝlOrRd´` is used to create the choropleth map. Doing so offers valuable insights into significant employment spot activity across different parts of the world. The map visualizes job demand by shading regions with colours ranging from low to high demand, with lighter shades indicating areas with more significant job opportunities and darker shades representing regions with lower demand. The colour bar displays the number of postings with a gradient from yellow to maroon. The limits are set based on the count of postings.

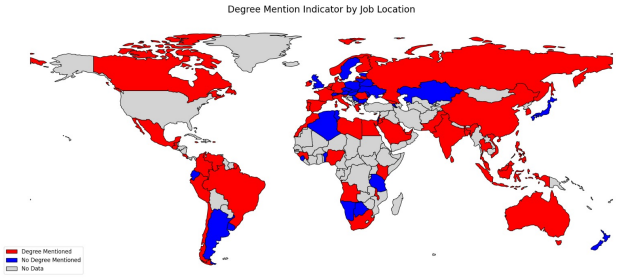


Fig. 6: Degree Mention Indicator by Job Location

The last world plot, as shown in Figure 6, compares job location distribution with the degree mention indicator, allowing for an assessment of whether a degree

is required or not in specific geographic locations for the job. The visualization maps the degree requirement based on the country search location, providing valuable insights into degree requirement concentration in job-seeking areas.

5.4 Calendar Heatmap of Job Postings

The calendar heatmap shown in Figure 7 provides a compact representation of job posting trends throughout 2023. It presents a monthly overview of daily hiring activities. The `ax.set_xticklabels(list(day_abbr)[:days_in_month], rotation=0)` command sets custom date labels. This heatmap aggregates daily job postings onto a calendar grid, revealing seasonal patterns in hiring activities. Each cell in the grid represents a specific day, with the colour intensity indicating the volume of job postings. Darker hues denote periods of higher ac-

Fig. 7: Calendar Heatmap of Job Postings Over Months

tivity, while lighter shades indicate fewer postings. This plot allows for quickly identifying peak hiring periods, weekly patterns, and potential market trends. One can easily determine the days of the week that tend to be most active for

job postings across different months. Additionally, it helps observe any shifts in hiring patterns throughout the year. The heatmap provides both an overview for broad analysis and detailed day-to-day comparisons.

5.5 Bubble Plot of Top Employers' Jobs and Salaries

The bubble plot in Figure 8 highlights companies with significant hiring activities. Each bubble's size is based on the number of job postings, and the colours are randomly assigned. Larger bubbles indicate companies with more job postings. The x-axis displays average salaries in thousands of dollars, allowing for easy compensation comparison between companies. The y-axis shows the number of job postings, clearly showing which companies are hiring actively. The bubble colours also help distinguish between different companies. The `plt.annotate` adds annotations for each bubble. This plot helps to quickly compare companies regarding hiring volume and salary offerings.

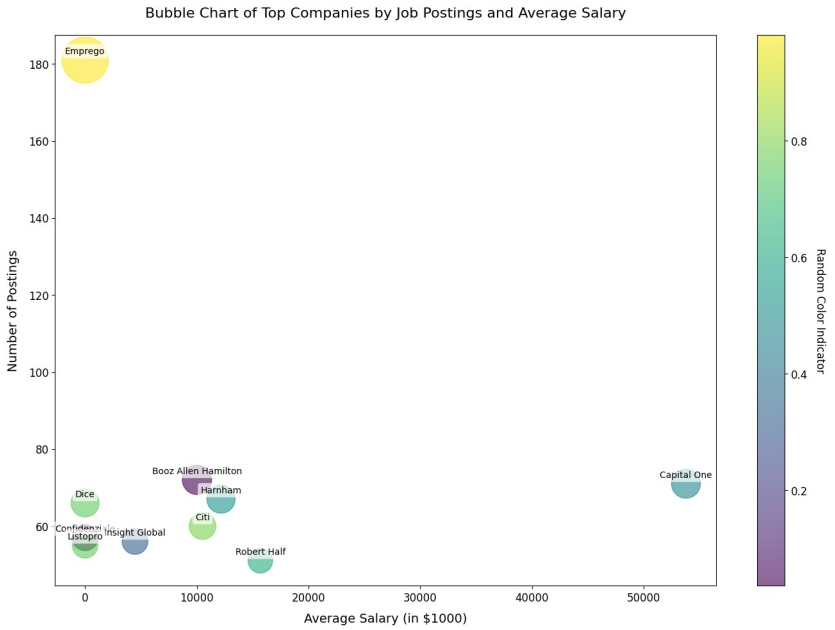


Fig. 8: Plot for Top Companies by Job Postings and Average Salary

5.6 Violin Plot of Salaries and Health Benefits

Next is the violin plot, as shown in figure 9. This depicts the relationship between average annual salaries and the availability of health insurance across job

postings. The violin plot is created using seaborn: `sns.violinplot`. The x-axis represents whether the health insurance is available(labelled 'TRUE') and health insurance is not available(labelled 'FALSE'). The y-axis represents the average annual salary. The plot combines parts of box plots with density plots to provide a more comprehensive view of data distribution. More comprehensive sections of the plot indicate more significant variability in salary offerings. In contrast, vertical positions indicate median, which offers insights into each category's central tendencies and salary spread.

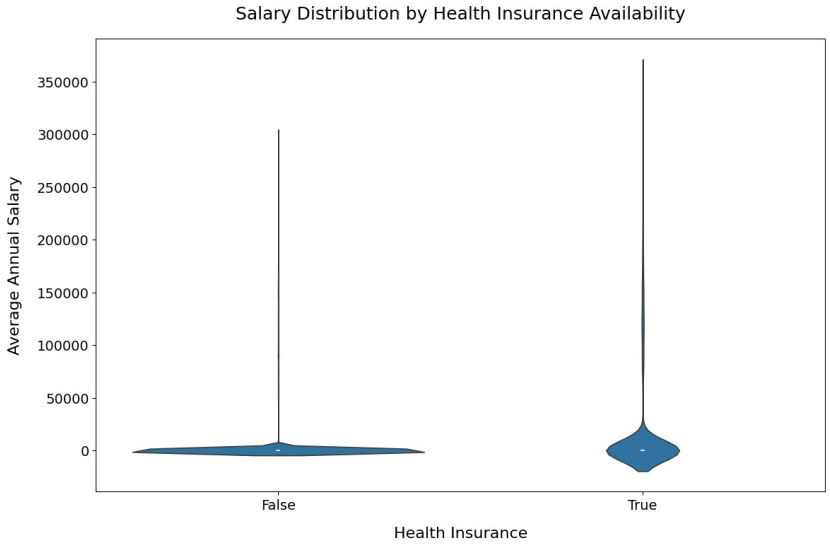


Fig. 9: Salary Distribution by Health Insurance Availability

5.7 Radar Plot for Average Salary

Next is a radar plot, as shown in Figure 10, comparing average salaries across different job titles. The radar chart plots each job title as a spoke around a central point, with axes extending outward representing salary levels. To create the radar plot shape, the `ax.fill` and `ax.plot` is used. The blue line connects the salary points for each job title, forming a polygon. The shape of this polygon allows for a quick comparison of salaries across different roles. Jobs with points further from the centre offer higher average salaries. For instance, the plot shows that Senior Data Scientist has the highest average salary at $6k, while roles like Business Analyst have lower salaries, closer to the centre.

All the plots use `matplotlib` and `seaborn` for creating dynamic visualizations in Python. Map data for the world maps is downloaded from Natural Earth

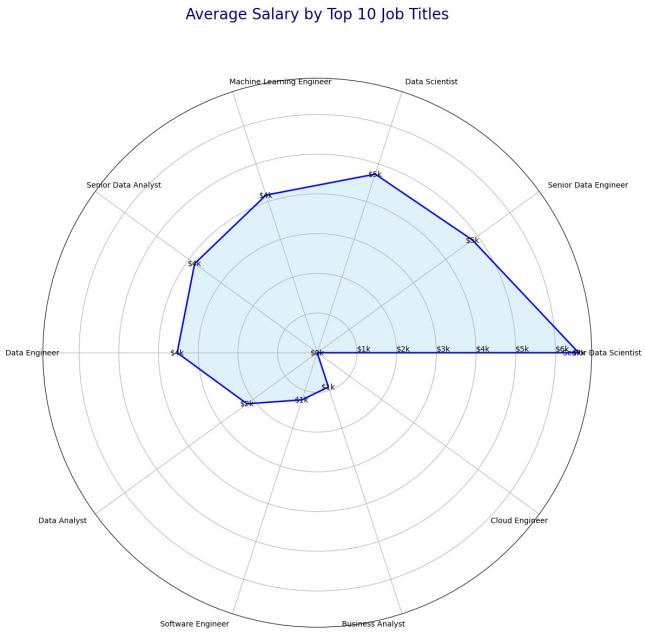


Fig. 10: Average Salary by Top 10 Job Titles

to plot the outlines of countries as seen in section 5.3. There are also waffles, squarify, horizontal bars, pie charts, and another heat map in the artifact's images folder. For this, refer to appendix section.

6 Summary

The project summary derives that with the help of Python, we can visualize and analyze complex data in a more detailed representation, which is easy to interpret and understand helpful information. It can also automate analysis and process data faster and more accurately, which is especially advantageous for handling large datasets and streamlining repetitive tasks.

The Matplotlib and Seaborn libraries are suitable for visualizing diverse datasets, enabling clear visualization of complex plots and enhancing data interpretation. Due to the vast support of libraries by Python, data collection, transformation, and representation were very convenient and efficient. Additionally, for data analysis, as mentioned in section 5.1, Jupyter Notebook played a very vital role due to its easy and fast processing of vast dataset.

Table 1: Used Topics from the Lecture

Topics	
Linux	Not used.
Text Editor	The python code was done in VS code editor.
Git	Git was used as a repository, with no specific details mentioned in the report.
Docker	Docker was to create the image.
Automation	The entire analysis was scripted in python. (see from section 3)
Gnuplot	Not used.
Matplotlib	Seaborn was used for producing plots. (see from section 5.2
LaTeX	The report was written in LaTeX without any noteworthy details mentioned.
Juypter Notebook	Juypter notebook was used for data analysis, training the model. (see section 5.1)

References

1. Open Sources , github, kaggle, geeks -for -geeks.
2. Pandas Documentation, `https://pandas.pydata.org/docs/`
3. Hugging face,`https://huggingface.co/datasets/lukebarousse/data_jobs`
4. Matplotlib Documentation, `https://matplotlib.org/stable/index.html`
5. Pywaffle documentatio, `https://pywaffle.readthedocs.io/en/latest/`
6. Geocoding Geopandas Documentation, `https://geopandas.org/en/stable/docs/user_guide/geocoding.html`
7. seaborn Documentation, `https://seaborn.pydata.org/`
8. squarify Documentation, `https://github.com/laserson/squarify?tab=readme-ov-file#Documentation-for-Squarify`

Appendix

For the complete code, project details, and other different plots, please refer to the Git repository: [Git Repository Link](`https://github.com/freiburg-missing-semester-course/project-supu18-1`)