Project Outline: Interactive Data Visualization for Job Skills Analysis

Project Team Members

- **Sean Schallberger**
- **Jose Traboulsi**
- **Karla Lopez**
- **Meghdut Noor**

Project Overview

We aim to create an **interactive data visualization dashboard** that helps job seekers understand **which skills are most in demand for a specific job title** and prioritize which skills to learn. The dashboard will:

- Allow users to **input a job title**.
- Display a **hierarchical ranking of the most in-demand skills** for that position.
- Provide an **interactive feature** where users can **tick off skills they already possess**, dynamically adjusting the ranking to recommend **high-impact skills** to focus on.

Project Track

- **Data Visualization Track** (Primary)
- Light **Data Engineering** components for data transformation and storage.

Dataset(s)

We will source data from:

1. **1.3M LinkedIn Jobs & Skills (2024)**
    - Dataset Link
2. **Data Science Job Postings & Skills (2024)**
    - Dataset Link

We may combine, clean, and transform the datasets for improved usability.

Key Features

1. Data Cleaning & Processing

- Standardizing job titles to handle inconsistencies.
- Removing duplicates and irrelevant job postings.
- Aggregating skill frequency per job title for ranking.

2. Interactive Dashboard

- **User Input:** Users enter a job title.
- **Skill Ranking:** Displays a **hierarchical list of in-demand skills**.
- **Customization:** Users tick off the skills they already have.
- **Dynamic Adjustment:** The dashboard updates recommendations based on missing skills.

3. Data Storage & Retrieval

- Store processed job & skills data in **MongoDB**, allowing:
  - **Flexible schema** for varying job descriptions and skills.
  - **Nested data structures** for hierarchical skill categorization.
  - **Indexing for fast queries** on job titles and skills.
- Use **Flask API** to fetch and serve data dynamically.

4. Visualization & User Interaction

- Use **Plotly, D3.js, or another JS library** for interactive elements.
- Display **bar charts, word clouds, or tree maps** for skill demand.
- Include **dropdowns & checkboxes** for user interaction.

Technical Stack

- **Backend & API:** Python, Flask, FastAPI (potentially)
- **Database:** MongoDB (hosted on MongoDB Atlas or locally)
- **Visualization:** JavaScript (Plotly, D3.js)
- **Hosting & Cloud Services:** Local deployment with optional cloud hosting
- **Deployment Options:** Flask app locally, potential cloud integration (see Extras)

Project Milestones & Timeline

Date	Task
Week 1	Project ideation, dataset selection, data cleaning
Day 3-5	MongoDB setup, Flask API development, initial visualizations
Week 2	Interactive dashboard development, final tweaks
Final Days	Presentation preparation & deployment

Deliverables

- **Final Interactive Dashboard**
- **MongoDB Database with Cleaned Data**
- **Flask API to Fetch Data**
- **GitHub Repository** (including code, data, README)
- **Presentation Deck** (10-minute demo)

Next Steps

1. Assign specific roles (**data cleaning, database setup, API, visualization, UI/UX**).
2. Set up **MongoDB Atlas** and test queries.
3. Develop **API endpoints** for dynamic data retrieval.
4. Finalize **frontend design and interactive elements**.

EXTRAS: Cloud Integration with AWS

To **enhance scalability, reliability, and accessibility**, we can implement **AWS or a similar cloud component** as an optional extension:

1. MongoDB Atlas for Cloud Database

- Use **MongoDB Atlas** to manage the database in the cloud.
- Enable **replication and indexing** for better performance.
- Allow secure access via **IAM roles and VPN restrictions**.

2. AWS Lambda for Serverless Data Processing

- Use **AWS Lambda** to automate **ETL processes**, such as:
  - Transforming and cleaning job skills data before inserting it into MongoDB.
  - Running periodic updates to keep the database current.

3. AWS API Gateway + Flask Backend

- Deploy our **Flask API on AWS Lambda** (using **Zappa** or **Serverless Framework**).
- Use **AWS API Gateway** to manage requests securely.

4. AWS S3 + CloudFront for Frontend Hosting

- Host the **interactive dashboard** as a static web app on **AWS S3**.
- Use **CloudFront** for fast, global content delivery.

Alternative Cloud Options

If AWS is not preferred, we could consider:

- **Google Cloud Platform (GCP)**: Firestore for NoSQL, Cloud Functions for serverless processing.
- **Microsoft Azure**: CosmosDB for NoSQL, Azure Functions for automation.