# **Project Plan: Employee–Job Matching (Synthetic Data Generation)**

Hi Smita,

I wanted to share the plan we discussed for the employee–job matching project so we are on the same page before we start generating data. Since we don’t have access to real HR data, we’ll first create synthetic data. This might help us test our schema design, run some matching experiments, and later try optimization.

## **Big Picture**

The project has two major data sides:

1. **Employee side (supply)** – details about the employees (skills, experience, preferences, bench time, etc.).
2. **Job side (demand)** – details about projects/roles (budget, required skills, duration, manager preferences, etc.).

Our approach is:

* First, create consistent synthetic datasets for employees and jobs.
* Then, use them to simulate how we can match employees to projects.
* Finally, optimize assignments under constraints like budget, bench time, and availability.

## **Work Split**

* **You** → Focus on **employee data generation**.
* **Me** → Focus on **job data generation** + **adjacent skill mapping**.

We’ll use the same **shared dictionaries** (skills, roles, domains, etc.) so that the employees you generate and the jobs I generate can connect smoothly.

## **Employee Schema (your side)**

Here are the fields we agreed to capture for employees:

* **Basic details**: Employee ID, age, job title, level, years of experience, education, domain expertise.
* **Skills**: Each skill with proficiency and years of experience.
* **Soft skills**: Communication, leadership, teamwork, adaptability, etc. (with ratings).
* **Work history**: Previous projects (inside/outside company), achievements, certifications.
* **Workload info**: Current projects, bandwidth (how much free time left), bench time.
* **Preferences**: Location preference, travel flexibility, preferred project types.
* **Feedback/Performance**: Past manager reviews, efficiency scores.

## **Job Schema (my side)**

And for jobs, I’ll generate the following:

* **Basic details**: Project ID, project name, domain, location, start date, end date, duration.
* **Requirements**: Budget, HR needs (roles and seniority), required skills/technologies, years of experience needed.
* **Context**: Similar past projects, project priority, manager preferences.
* **Constraints**: Bill rate/client revenue potential, deadlines, SLA requirements.

## **Shared Dictionaries**

To keep everything consistent, we’ll create and freeze some **common reference lists**:

* **Skills** (e.g., Python, Java, TensorFlow, React, SQL).
* **Roles & levels** (Junior Dev, Senior Dev, Tech Lead, Architect).
* **Domains** (AI/ML, Cloud, Web/Mobile, Banking IT, ERP, Cybersecurity).
* **Locations** (Chennai, Bangalore, Pune, Austin, London, etc.).
* **Soft skills** (communication, leadership, teamwork, creativity).

We’ll store these in JSON/YAML files that both of us can use when generating data.

## **Adjacent Skill Mapping**

I’ll also take care of defining **adjacent skill relationships**. For example:

* Java ↔ C++ (0.9 similarity)
* Python ↔ R (0.7 similarity)
* TensorFlow ↔ PyTorch (0.85 similarity)

This means when a job asks for one skill, employees with a “close enough” skill can also be considered.

**Integration Plan**

1. We both agree on the shared dictionaries.
2. You generate employees.csv.
3. I generate jobs.csv and a skill\_similarity.json.
4. We write a small script to cross-check: for each job, do we have enough employees with matching (or adjacent) skills?
5. Then we move on to building the matching + optimization logic.

## **Next Steps**

1. Finalize the shared dictionaries (skills, roles, domains, locations).
2. You start with generating ~200–500 employees.
3. I’ll generate ~50–200 jobs + skill mappings.
4. We test integration before scaling up.

Here are rough schema drafts I think the data might look like  
 **Example Schema Draft**

## **Employee Table (Synthetic Data)**

| **Field** | **Type** | **Description** | **Example** |
| --- | --- | --- | --- |
| emp\_id | String | Unique employee identifier | E1234 |
| age | Integer | Employee age | 29 |
| job\_title | String | Current role/job title | Data Scientist |
| level | String | Seniority level (Junior, Senior, Lead, Architect, etc.) | Senior |
| years\_experience | Integer | Total years of experience | 7 |
| skills | JSON/Array | List of skills with proficiency (1–5) and years of use | [{"Python": (5, 6yrs)}, {"TensorFlow": (4, 3yrs)}] |
| domain | String | Main domain expertise | AI/ML |
| education | String | Highest qualification | M.Tech AI |
| soft\_skills | JSON/Dict | Ratings for soft skills (communication, leadership, teamwork, etc.) | {"communication": 4, "leadership": 3} |
| current\_projects | Array | List of project IDs currently assigned | [P1002, P1021] |
| bandwidth | Float | Fraction of time available for new projects (0–1) | 0.3 (30% free) |
| location\_pref | String | Preferred location for work | Chennai |
| travel\_flex | String | Travel flexibility (Low, Medium, High) | High |
| achievements | Text | Awards/achievements | Top Innovator 2023 |
| feedback\_score | Float | Aggregated performance/feedback rating | 4.2 |
| project\_pref | Array | Employee’s preferred types of projects | [AI Healthcare, NLP] |
| bench\_time | Integer | Current bench duration in weeks (0 if not on bench) | 0 weeks |

## **Job Table (Synthetic Data)**

| **Field** | **Type** | **Description** | **Example** |
| --- | --- | --- | --- |
| job\_id | String | Unique project/job identifier | P1002 |
| project\_name | String | Name/title of the project | AI Chatbot Development |
| domain | String | Domain or industry vertical | AI/ML |
| location | String | Location of job/project | Bangalore |
| start\_date | Date | Project start date | 2025-10-01 |
| end\_date | Date | Project end date | 2026-03-01 |
| duration\_months | Integer | Duration of the project in months | 6 |
| budget | Integer | Project budget (currency) | 50,00,000 INR |
| technologies | Array | Required technologies/skills | [Python, TensorFlow, NLP] |
| hr\_requirements | JSON/Dict | Role & number of employees needed | {"Senior DS": 2, "Junior DS": 3} |
| min\_experience | Integer | Minimum years of experience required | 3 yrs |
| manager\_pref | Array | List of preferred employees for this project | [E1234, E1280] |
| priority | String | Priority level (High, Medium, Low) | High |
| similar\_projects | Array | IDs of similar past projects | [P0901, P0856] |
| remote\_possible | Boolean | Whether remote work is allowed | Yes |

## **Repository Overview**

This project is a prototype for employee-to-project matching using synthetic data. The goal is to simulate internal staffing workflows using realistic employee profiles and job descriptions, with skill-based matching, adjacent skill graphs, and allocation optimization.

## **Current Directory Structure**

.

├── data/ # Generated datasets (CSV output)

├── docs/ # Documentation, reports

├── employees/ # Employee generator scripts (Smita)

├── experiments/ # Matching / optimization experiments

├── jobs/ # Job generator scripts (Sathwik)

├── notebooks/ # EDA, clustering, similarity analysis

├── shared/ # Shared dictionaries and project config

│ ├── domains.json # Industry verticals and service lines

│ ├── locations.json # India/global/remote location list

│ ├── roles\_levels.json # Cognizant-style roles and designation ladder

│ ├── settings.yaml # Generator settings, adjacency knobs, data sizes

│ ├── skill\_similarity.json # Skill adjacency graph (data-driven)

│ ├── skills.json # Canonical skill categories and labels

│ └── soft\_skills.json # Soft skill attributes and 1–5 rating scale

├── src/ # Utilities, schemas, validation helpers

├── README.md # High-level project overview

├── requirements.txt # Python dependencies

## **Work Done So Far**

1. Initialized GitHub repository and local environment (Python virtualenv).
2. Created project folder structure.
3. Created .gitignore, requirements.txt, and .vscode/settings.json.
4. Installed all required libraries.
5. Created and committed:  
   * roles\_levels.json with Cognizant-specific roles and levels (PAT to Director+).
   * skills.json with ~100+ realistic skills across programming, AI, cloud, ERP, QA, etc.
   * domains.json with industry verticals and service lines.
   * locations.json split into India/global/remote.
   * soft\_skills.json with soft skill attributes and guidelines.
   * skill\_similarity.json (data-blended adjacency graph from embeddings, co-occurrence, and taxonomy).
   * settings.yaml with generator parameters, adjacency filtering knobs, and size settings.
6. Created project branch:  
   * feature/jobs-and-adjacent-skills for all job-side work.

## **Next Tasks - Mine**

* Scaffold job generator script: jobs/generate\_jobs.py.
* Use settings.yaml to control generation size, skills per job, duration, and budget.
* Use skill\_similarity.json for adjacent skill logic during matching.
* Write reusable skill activation/thresholding code.
* Implement job-to-ideal-employee vector creation for future matching.

## **Next Tasks - Yours**

* Create branch: feature/employees.
* Implement employee generator: employees/generate\_employees.py.
* Use:  
  + roles\_levels.json to assign roles and experience ranges.
  + skills.json to assign hard skills.
  + soft\_skills.json to randomly rate soft skill attributes.
  + domains.json to assign domains.
  + locations.json to assign locations and preferences.
  + settings.yaml to configure employee count, bandwidth, bench probability, etc.
* Output: data/employees.csv

## **Shared Guidelines**

* All dictionaries are frozen and must be reused exactly.
* Do not rename skill labels or add new roles directly. Propose via shared file update.
* All synthetic data should be written to data/.
* Code modules go into src/ (e.g., validation, utils, scoring).

## **Goals After Generators**

* Write a cross-check script to verify coverage: every job requirement must match at least one employee (exact or adjacent).
* Begin similarity modeling (E2E, J2E) and score generation.
* Build optimization engine.
* Define KPIs: bench reduction, match quality, constraint satisfaction.
* Final deliverables: CSVs, matching scripts, optimizer, visualizations, report.