International Journal of Science, Strategic Management and Technology
Volume **02** Issue **11** November-2026 | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

Intelligent Career Recommendation System

A Skill-, Interest-, and Goal-Aware Career Guidance Prototype

Ms. R. Jeyashree
Assistant Professor
Department of Advanced Computing and Analytics
VISTAS
rjeyashree.scs@vistas.ac.in

Dr. S. Prathiba
Assistant Professor
Department of Advanced Computing and Analytics
VISTAS
prathiba.scs@vistas.ac.in

Dr. T. R. Nisha Dayana
Associate Professor
Department of Computer Science and Information Technology
VISTAS
nisha.dayana1984@gmail.com

https://doi.org/10.55041/ijsmt.v2i11.041

Cite this Article: Jeyashree, R., Prathiba, S. & Dayana, T. R. N. (2026). Intelligent Career Recommendation System. International Journal of Science, Strategic Management and Technology, *02*(11), 1-9. https://doi.org/10.55041/ijsmt.v2i11.041

License: This article is published under the Creative Commons Attribution 4.0 International License (CC BY 4.0), permitting use, distribution, and reproduction in any medium, provided the original author(s) and source are properly credited.

Abstract

Choosing a suitable career is difficult for students who have different technical skills, interests, academic experiences, and professional goals. A generic list of job titles does not explain which role is appropriate or what a student should learn next. This paper presents an *Intelligent Career Recommendation System* that produces a ranked shortlist of career paths by comparing a learner profile with a small career knowledge base. The proposed prototype uses a transparent weighted scoring method: skill coverage contributes 50%, interest alignment con-tributes 30%, and preferred-domain alignment contributes 20%. It then explains each recommendation by displaying the com-ponent scores, missing skills, and suggested next steps. The design draws on established career-exploration ideas, including occupational interest profiling, and on structured occupation and skill classifications [1], [2]. A lightweight Python imple-mentation is provided so the ranking process can be inspected and reproduced without a machine-learning training dataset. Three synthetic student profiles demonstrate how recommenda-tions change when skills and interests change. These examples are illustrative outputs of the prototype, not results from a real-world user study. The system is intended as a decision-support tool: its recommendations should encourage exploration and discussion rather than replace a student, mentor, or career counsellor's judgement.

Keywords— Career recommendation, recommender system, skill matching, interest profiling, Python, explainable AI, stu-dent guidance.

1 Introduction

Career planning is an important decision for students, gradu-ates, and people considering a change in profession. A student may know several programming languages but may not know whether to explore software development, data analytics, cy-bersecurity, cloud computing, or artificial intelligence. At the same time, a student may have strong interest in a field but lack some of the skills listed in entry-level role descriptions. A useful career guidance tool should consider both present capabilities and future learning needs.

Traditional guidance often depends on general advice, in-formal discussions, online searches, or static job lists. These resources can be valuable, but they can require substantial effort to compare and may not clearly explain why a role is relevant to a particular student. A recommendation system can organise role information, compare it with a structured student profile, and return a small set of options in a consistent format.

The proposed system is designed for an academic prototype. A learner enters skills already acquired, interests, a preferred career domain, and optional educational information. A ca-reer knowledge base stores role titles, expected skills, interest tags, related domains, and starter learning paths. A scoring component evaluates every role, sorts the roles, and reports the most relevant matches with skill gaps. The prototype is deliberately transparent so that a student can understand why the ranking occurred and can correct an inaccurate input.

1.1 Problem Statement

Students commonly face three connected problems: (i) diffi-culty connecting current skills to a set of possible occupations;
(ii) limited visibility of the skills missing for a chosen role; and
(iii) difficulty converting a career preference into an actionable learning plan. The system addresses these needs by mapping profile information to occupation profiles and showing both ranked options and the reasons behind them.

1.2 Objectives

The objectives of the project are:

- collect and standardise student skills, interests, and pre-ferred career domain;
- maintain a small, editable knowledge base of career roles and their requirements;
- calculate a match score for each role and rank the roles from highest to lowest;
- explain the recommendation through component scores, missing skills, and learning suggestions; and
- provide a simple Python implementation that can later be integrated with a web interface or a larger occupation dataset.

1.3 Scope and Intended Users

The primary users are college students and recent graduates exploring early-career directions. The first version focuses on five illustrative technology and business-technology roles: Data Analyst, Python Developer, Machine Learning Engineer, Web Developer, and Business Analyst. The role profiles are intentionally small for readability and demonstration. A pro-duction system should use a broader and regularly maintained occupation source, validate recommendations with counsellors, and evaluate performance with real user feedback.

2 Related Work and Background

Career exploration tools have long considered work interests alongside a person's abilities and goals. The O*NET Interest Profiler, for example, measures six broad interest areas aligned with the RIASEC structure and links interest results to oc-cupations [1]. This provides a useful design principle for the project: recommendations should not be based on technical skills alone. Interests are useful signals, but they should not be interpreted as a fixed label or a guarantee of career success. Occupation and skill taxonomies can help systems represent roles consistently. The European Skills, Competences, Qual-ifications and Occupations (ESCO) classification organises occupation concepts and related skill or competence concepts, helping different services refer to occupations and skills using shared terminology [2]. A future implementation could map its local role catalogue to an established taxonomy rather than
relying only on manually typed tags.

Research on job recommender systems describes content-based, collaborative, hybrid, knowledge-based, and context-aware approaches. A 2025 systematic literature review identi-fies similarity-based methods, machine-learning methods, and

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

International Journal of Science, Strategic Management and Technology
Volume **02** Issue **11** November-2026 | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

Table 1: Summary of the system design

Aspect	Proposed design
Input	Skills, interest tags, preferred domain Core
method	Weighted content-based scoring Knowledge
base	Career roles, skills, interest tags, learning paths
Output	Ranked roles, score breakdown, skill gaps
Explainability	Fixed weights and visible overlap calcula-tions
Runtime	Python 3; standard library only

knowledge-based methods as common approaches, while also noting ongoing challenges such as bias and the need for more effective systems [3]. A 2026 review similarly highlights inter-operability, standardised skill representation, interpretability, and fairness as important design challenges for AI-based job recommendation [4]. These findings support a prototype that is explainable and explicit about its role data and scoring assumptions.

2.1 Approaches Considered

Content-based matching compares the features of a student with attributes of career roles. It works even when there is no large history of user ratings, so it is suitable for an initial stu-dent project. **Collaborative filtering** learns from similarities between users' choices or behaviours, but it typically requires a meaningful interaction history. **Hybrid recommenda-tion** combines content-based signals with learned models, user feedback, or contextual information. The current prototype uses a transparent content-based baseline; hybrid or machine-learning methods are possible extensions after appropriate data collection and evaluation.

2.2 Identified Gap

For a student-facing tool, the output must be more useful than a ranked list. It should explain the match, identify skill gaps, and offer next steps. The proposed prototype therefore adds a simple explanation and learning-path layer to the ranking calculation. It does not claim to infer an objectively perfect career or to predict employment success.

3 Existing System and Proposed System

3.1 Existing Practice

Common approaches include manual internet searches, generic career quizzes, lists of job descriptions, and informal advice from peers or mentors. Each approach can offer useful context. However, manually comparing many roles is time-consuming, and some tools provide limited explanation of how an outcome

was produced. Job portals frequently prioritise matching a candidate to a current vacancy, whereas this project focuses on exploration of broader career directions and the skills needed to move towards them.

3.2 Proposed System

The proposed system uses four main information groups: stu-dent skills, student interests, preferred domain, and career-role attributes. It normalises the text to reduce differences caused by capitalisation or extra spaces, calculates three independent match scores, and combines them using fixed weights. For each role, it also identifies the expected skills that do not appear in the student's current skill list. The ranked result is presented with reasons and possible learning steps.

4 System Architecture

The system is separated into the student-input layer, prepro-cessing layer, career knowledge base, matching and ranking engine, and explanation/output layer. The modular archi-tecture makes it possible to replace one component without redesigning the whole system. Figure 1 summarises the data flow.

4.1 Main Modules

1) Student Profile Module: collects a list of skills, interest tags, and a preferred domain such as analytics, software, or AI. Education level or completed coursework may be added as optional information, but should not be used as a hidden proxy for personal characteristics.

2) Preprocessing Module: converts comparable text values to lowercase and removes accidental surrounding spaces. In a larger system, this module could map synonyms such as "JS" and "JavaScript" to one canonical skill and validate empty or unknown values.

3) Career Knowledge Base: each role includes a title, required skills, interest tags, a primary domain, related do-mains, and suggested learning steps. Role information should be reviewed periodically because tools, entry requirements, and labour-market terminology change.

4) Matching and Ranking Engine: computes skill cov-erage, interest alignment, and domain fit for every role. It combines the component scores using a weighted sum and sorts the results from high to low.

5) Explanation and Skill-Gap Module: returns the component scores, the skills not yet present in the profile, and a starter learning path. A student can use the gaps to prioritise learning or discuss the result with a mentor.

4.2 Data Flow

First, the student submits their profile. The application checks and normalises the values. It then compares the profile with every career record. The scoring engine produces a score and a list of missing required skills for each role. Finally, the system displays the top-ranked careers and their explanations. No sensitive personal information is necessary for the current ranking method.

5 Methodology and Scoring Algorithm

The prototype uses a transparent weighted score. For a career c, the skill score S_c is the proportion of required skills found in the student's profile. The interest score I_c is the proportion of role interest tags found in the student's interest list. Both are expressed on a 0–100 scale.

$$S_c = 100 \frac{|K_s \cap K_c|}{|K_c|} \quad (1)$$

$$I_c = 100 \frac{|T_s \cap T_c|}{|T_c|} \quad (2)$$

Here, K_s represents the student's skills, K_c the role's re-quired skills, T_s the student's interest tags, and T_c the interest tags associated with the role. If a role has no target tags or a student leaves a field blank, a production system should handle the missing value explicitly; the sample code uses a neutral value for an empty target list and a neutral domain score if no preference is given.

The domain score D_c is 100 when the preferred domain exactly matches the role domain, 60 when it is listed as a related domain, and 0 otherwise. These values are configurable

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

International Journal of Science, Strategic Management and Technology
Volume **02** Issue **11** November-2026 | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

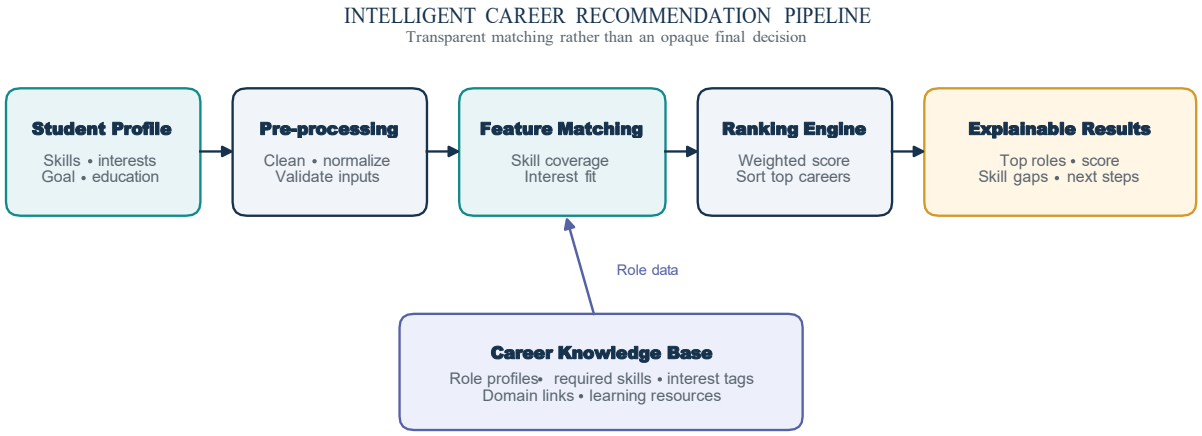


Figure 1: Architecture of the Intelligent Career Recommendation System. The career knowledge base supplies role attributes to the matching component; the ranking engine produces explanations as well as scores.

design choices for the demonstration, not universal career-science constants. The final score is:

$$R_c = 0.50S_c + 0.30I_c + 0.20D_c. \quad (3)$$

The 50% skill weight gives the strongest influence to skills that can be developed and demonstrated. The interest weight prevents the system from recommending a role based only on current technical overlap. Domain alignment adds a simple preference signal. Different weights can produce different ranks; therefore, the chosen weights must be disclosed and evaluated rather than presented as objectively correct.

5.1 Algorithm

1. Read the student's skills, interests, and preferred domain.
2. Normalise profile values and each career record.
3. For every career, compute skill coverage, interest fit, and domain fit.
4. Combine the values using Equation 3.
5. Compute skill gaps as required skills minus the student's current skills.
6. Sort all careers by the final score in descending order.
7. Return the top N careers with explanations and learning steps.

5.2 Pseudocode

```
INPUT student_profile, career_catalogue, top_n
NORMALISE student skills, interests and domain
results = empty list
FOR each career in career_catalogue:
    skill_fit = overlap(student skills, career skills)
    interest_fit = overlap(student interests, career
        interests)
    domain_fit = compare(student domain, career domain)
    score = 0.50*skill_fit + 0.30*interest_fit
          + 0.20*domain_fit
    gaps = career skills NOT IN student skills
    append(score, career, gaps, learning_path) to
        results
SORT results by score descending
RETURN first top_n items with explanations
```

5.3 Why the Method Is Appropriate for a Pro-totype

The method is easy to audit: a developer can explain each component without relying on a hidden model. It also works with a small manually curated role catalogue and does not re-quire user-rating history. Its limitations are equally important: exact text overlap cannot fully understand synonyms, skill depth, transferable capabilities, or nuances of a job description. The approach is therefore a baseline implementation, not proof of an individual's long-term aptitude or employability.

6 Implementation and Coding

The prototype is implemented in Python 3 and uses only built-in data structures. A career is represented as a dictionary with its required skills, interest tags, domain, related domains, and starter learning path. The input profile uses lists of skills and interests. A normalisation function converts values to lowercase so that "Python" and "python" are treated as the same text value.

The main function calculates each component score, iden-tifies missing skills, stores the explanation, sorts the results, and returns the top three recommendations. The data is kept in code for easy demonstration; in a web application, the same data could be stored in a CSV file or database. The complete runnable source code is included below and is also supplied as a separate Python file for convenient execution.

6.1 Running the Prototype

Save the source as career_recommender.py and run:

```
python career_recommender.py
```

A Python 3 interpreter is sufficient; no third-party package needs to be installed. The implementation prints the profile, top three roles, component scores, missing skills, and suggested next steps. For a web-based implementation, the scoring function can be exposed through a Flask or FastAPI endpoint and connected to a HTML form and results page.

6.2 Complete Python Source Code

```
"""Intelligent Career Recommendation System - minimal runnable
    prototype.

Run with: python career_recommender.py
The score is a transparent heuristic, not a validated psychometric
    assessment.
"""

CAREERS = [
    {
        "title": "Data Analyst",
```

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

```
            "required_skills": ["Python", "SQL", "Pandas", "Excel", "Power BI"],
            "interest_tags": ["data analysis", "visualization", "business"],
            "domain": "analytics",
            "related_domains": ["ai", "software"],
            "learning_path": ["Excel", "Power BI", "SQL case studies"],
        },
        {
            "title": "Python Developer",
            "required_skills": ["Python", "SQL", "Git", "API development", "testing"],
            "interest_tags": ["programming", "problem solving", "software development"],
            "domain": "software",
            "related_domains": ["analytics", "ai"],
            "learning_path": ["Git and GitHub", "REST APIs", "unit testing"],
        },
        {
            "title": "Machine Learning Engineer",
            "required_skills": ["Python", "statistics", "machine learning", "SQL", "model evaluation"],
            "interest_tags": ["ai", "data analysis", "experimentation"],
            "domain": "ai",
            "related_domains": ["analytics", "software"],
            "learning_path": ["Statistics", "scikit-learn", "model evaluation"],
        },
        {
            "title": "Web Developer",
            "required_skills": ["HTML", "CSS", "JavaScript", "Git", "responsive design"],
            "interest_tags": ["design", "ui", "frontend", "problem solving"],
            "domain": "software",
            "related_domains": ["design"],
            "learning_path": ["CSS and JavaScript", "Responsive layouts", "Frontend project"],
        },
        {
            "title": "Business Analyst",
            "required_skills": ["SQL", "Excel", "requirements analysis", "documentation", "communication"],
            "interest_tags": ["business", "analysis", "stakeholder communication"],
            "domain": "analytics",
            "related_domains": ["business", "software"],
            "learning_path": ["Excel", "Requirements gathering", "Dashboard storytelling"],
        },
]


def normalize(items):
    """Make comparison case-insensitive and remove accidental whitespace."""
    return {str(item).strip().lower() for item in items if str(item).strip()}


def percentage_overlap(student_items, target_items):
    """Return the percentage of a career profile matched by the student."""
    target = normalize(target_items)
    if not target:
        return 50.0
    return 100.0 * len(normalize(student_items) & target) / len(target)

def domain_fit(preferred_domain, career):
    """Score exact domain fit as 100, related as 60, otherwise 0."""
    preferred = str(preferred_domain).strip().lower()
    role_domain = career["domain"].lower()
    related = normalize(career.get("related_domains", []))
    if not preferred:
        return 50.0
    if preferred == role_domain:
        return 100.0
    if preferred in related:
        return 60.0
    return 0.0

def recommend_careers(student, careers=CAREERS, top_n=3):
    """Rank roles using skill fit (50%), interest fit (30%), and domain fit (20%)."""
    student_skills = normalize(student.get("skills", []))
    student_interests = normalize(student.get("interests", []))
    preferred_domain = student.get("preferred_domain", "")
    ranked = []

    for career in careers:
        skill_score = percentage_overlap(student_skills, career["required_skills"])
        interest_score = percentage_overlap(student_interests, career["interest_tags"])
        domain_score = domain_fit(preferred_domain, career)
        total = 0.50 * skill_score + 0.30 * interest_score + 0.20 * domain_score
        gaps = [skill for skill in career["required_skills"] if skill.lower() not in student_skills]
        ranked.append({
            "title": career["title"],
            "score": round(total, 1),
            "skill_score": round(skill_score, 1),
            "interest_score": round(interest_score, 1),
            "domain_score": round(domain_score, 1),
            "skill_gaps": gaps,
            "learning_path": career["learning_path"],
        })

    ranked.sort(key=lambda item: item["score"], reverse=True)
    return ranked[:max(1, top_n)]


def show_profile(label, student):
    print(f"\n{label}")
    print("-" * len(label))
    print("Skills:", ", ".join(student["skills"]))
    print("Interests:", ", ".join(student["interests"]))
    print("Preferred domain:", student["preferred_domain"])
    for rank, result in enumerate(recommend_careers(student), start=1):
        print(f"{rank}. {result['title']} - {result['score']:.1f}%")
        print(f"   Skill fit: {result['skill_score']:.1f}% | "
              f"Interest fit: {result['interest_score']:.1f}% | "
              f"Domain fit: {result['domain_score']:.1f}%")
        print("   Skill gaps:", ", ".join(result["skill_gaps"]) or "None identified")
        print("   Suggested next steps:", ", ".join(result["learning_path"]))


if __name__ == "__main__":
    profiles = {
        "Student A - Analytics preference": {
            "skills": ["Python", "SQL", "Pandas", "HTML", "Java"],
            "interests": ["data analysis", "programming", "problem solving", "visualization", "ai"],
            "preferred_domain": "analytics",
        },
        "Student B - Front-end preference": {
            "skills": ["HTML", "CSS", "JavaScript", "Git", "Python"],
            "interests": ["design", "ui", "frontend", "problem solving"],
            "preferred_domain": "software",
        },
        "Student C - AI preference": {
            "skills": ["Python", "SQL", "statistics", "machine learning", "Pandas"],
            "interests": ["ai", "data analysis", "experimentation"],
            "preferred_domain": "ai",
        },
    }

    for profile_name, profile in profiles.items():
        show_profile(profile_name, profile)
```

7 Worked Example

Consider Student A, who has basic knowledge of Python, SQL, Pandas, HTML, and Java. The student's interests include data analysis, programming, problem solving, visualisation, and AI. The preferred domain is analytics. The role catalogue contains five example roles. For the Data Analyst role, three of the five required skills match: Python, SQL, and Pandas. Therefore, the skill score is $3/5 \times 100 = 60$. Two of the three interest tags match, giving an interest score of approximately 66.7. Analytics is the exact preferred domain, so the domain score is 100. The final score is:

$$R_{\text{Data Analyst}} = 0.50(60) + 0.30(66.7) + 0.20(100) \approx 70.0.$$

For Python Developer, two of five required skills match, the interest fit is 66.7, and the domain is related rather than exact. Its score is $0.50(40) + 0.30(66.7) + 0.20(60) = 52.0$. The same calculation produces the rank for each career.

The system does not only state that Data Analyst ranks first. It gives the student an actionable direction: strengthen Excel and Power BI, practise SQL with realistic datasets, and build a small dashboard project. A student interested in software development could choose a different preferred domain and receive a different ranking even with the same technical skills.

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

International Journal of Science, Strategic Management and Technology
Volume **02** Issue **11** November-2026 | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

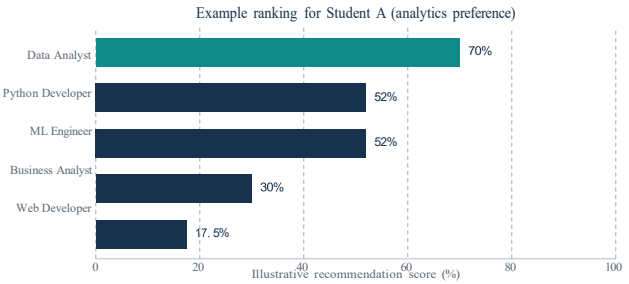


Figure 2: Illustrative ranking scores generated for Student A. The chart shows output of the rule-based prototype, not benchmark accuracy.

Table 2: Top results for Student A

Career	Score	Main skill gaps
Data Analyst	70.0%	Excel, Power BI Python
Developer	52.0%	Git, API development, testing
Machine Learning Engineer	52.0%	Statistics, machine learning, model evaluation
Business Analyst	30.0%	Excel, requirements analysis, documentation, communication
Web Developer	17.5%	CSS, JavaScript, Git, responsive design

8 Case Study: Three Synthetic Student Pro-files

This case study demonstrates the operation of the system with three fictional profiles. The profiles are deliberately varied to check whether the expected ranking behaviour is visible when the inputs change. They are not real participants, and the scores must not be interpreted as measured effectiveness or career outcomes.

8.1 Profile A: Analytics-Oriented Student

Profile A knows Python, SQL, Pandas, HTML, and Java and selects analytics as the preferred domain. The output ranks Data Analyst first at 70.0%, followed by Python Developer and Machine Learning Engineer at 52.0%. The explanation lists Excel and Power BI as the two missing Data Analyst skills. The suggested learning plan is practical for a student: develop spreadsheet and dashboard skills, complete a SQL case study, and publish the results in a portfolio.

8.2 Profile B: Front-End-Oriented Student

Profile B lists HTML, CSS, JavaScript, Git, and Python, with interests in design, UI, front-end development, and problem solving. The preferred domain is software. Web Developer ranks first at 90.0%, with responsive design as its primary gap. Python Developer ranks second at 50.0%. This result is expected because the profile strongly matches the web role's skills, interests, and domain. The result should prompt the student to practise responsive layouts and create a complete front-end project.

8.3 Profile C: AI-Oriented Student

Profile C lists Python, SQL, statistics, machine learning, and Pandas. The interest tags are AI, data analysis, and exper-imentation; the preferred domain is AI. Machine Learning Engineer ranks first at 90.0%, with model evaluation as the missing listed skill. Data Analyst ranks second at 52.0%. The recommended next step is to learn model evaluation methods, build a small supervised-learning project, and document its evaluation metrics and limitations.

8.4 Case-Study Observations

The synthetic examples show that role ranking changes with the student's current profile and preferred domain. The gap list turns a recommendation into a learning plan, and the component scores allow the student to question a result. For example, a student may disagree with the preferred domain or may possess a skill that has not been entered. The appropriate response is to update the profile and rerun the recommendation rather than treating the first rank as a final decision.

9 Results and Discussion

The prototype was run locally with the three example profiles. Its observable output is a ranked list with component scores and skill gaps. Student A receives Data Analyst (70.0%), Student B receives Web Developer (90.0%), and Student C receives Machine Learning Engineer (90.0%) as their top recom-mendation. These values are generated by the stated weighted rules and role catalogue. They measure profile-to-role over-lap under the chosen assumptions; they are not prediction accuracy, probabilities of getting a job, or evidence that the recommendation is objectively optimal.

The main benefit is explainability. Every score can be re-computed from visible overlap counts and domain rules. The program also identifies learning gaps without requiring a train-ing dataset. The main limitation is that a binary skill match treats beginner and advanced ability equally. Interests are represented by manually assigned tags, so terminology choices influence the ranking. In addition, the current catalogue con-tains only five roles, and the manually selected weights have not been validated against expert ratings or longitudinal student outcomes.

9.1 Suggested Evaluation Plan

A future evaluation should involve students and career counsel-lors under an approved, transparent process. Useful evaluation dimensions include:

- **Relevance:** whether users find the top-three roles relevant after reviewing their profiles;
- **Coverage:** whether the system returns useful suggestions for profiles across several technical and non-technical back-grounds;
- **Explanation quality:** whether students understand why each role was recommended and what a missing skill means;
- **Ranking metrics:** Precision@K or NDCG@K only when defensible relevance labels are collected; and
- **Fairness and usability:** whether results differ unjustifi-ably across groups and whether users can correct inaccurate profile information.

No user study or benchmark dataset was collected for this paper, so numerical claims about accuracy, fairness, or job placement are outside the scope of the current prototype.

10 Functional Requirements and Validation

The system's functional requirements describe observable be-haviour rather than a specific interface. FR-1 requires the application to accept a student profile containing skills, inter-ests, and a preferred domain. FR-2 requires case-insensitive normalisation before comparisons. FR-3 requires a separate skill, interest, and domain score for each career. FR-4 requires descending ranking by the weighted score. FR-5 requires the

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.

Table 3: Illustrative top recommendation per synthetic case-study profile

Profile	Dominant inputs	Preferred domain	Top score	Recommended next step
A	Python, SQL, Pandas; analytics and visualisation interests	Analytics	Data Analyst, 70.0%	Excel, Power BI, SQL case study
B	HTML, CSS, JavaScript, Git; UI and front-end interests	Software	Web Developer, 90.0%	Responsive design and portfolio website
C	Python, SQL, statistics, ma-chine learning; AI interests	AI	ML Engineer, 90.0%	Model evaluation and a docu-mented ML project

Table 4: Logic-level validation checks

Test	Input / condition	Expected behaviour
T1	Student A profile	Data Analyst ranks first at 70.0%
T2	Student B profile	Web Developer ranks first at 90.0%
T3	Student C profile	Machine Learning En-gineer ranks first at 90.0%
T4	Change "Python" to "python"	Match score remains unchanged
T5	Preferred domain left blank	Domain score uses neu-tral value 50
T6	Student lacks Excel and Power BI	Both appear in Data Analyst skill gaps Scores
T7	Inspect returned re-sults	are sorted in de-scending order

response to include missing skills and suggested learning steps. FR-6 requires career records and their weights to remain ed-itable so the prototype can be adapted to other student groups. Important non-functional requirements include transparency, reasonable response time for a small catalogue, maintainability of role records, and minimisation of personal data. Because the current program stores the sample catalogue in memory and performs a small number of set operations, it should respond quickly for the included five-role example. Performance for a much larger catalogue would need to be measured rather than assumed.

The first three checks are confirmed by the example runs included in this paper. The remaining checks follow directly from the implementation and should be kept as regression tests if the scoring logic is modified. A future code release should automate these checks with Python's built-in unittest framework, ensuring that changes to the catalogue or weights do not silently break ranking and explanation behaviour.

11 Limitations, Ethics, and Future Enhance-ments

11.1 Limitations

The current version has a small role catalogue and uses exact tag overlap. It does not parse a resume, verify skill proficiency, monitor current vacancies, or incorporate regional demand and salary data. Its scores depend on hand-selected weights and a manually curated list of role attributes. A high score only means that the profile overlaps with the sample role description under the current rules.

11.2 Responsible Use

Career recommendations can influence a student's education and employment choices. The system should avoid presenting the output as a definitive assessment, should let users correct or remove information, and should minimise the personal data it stores. Sensitive characteristics such as religion, caste, gender, health, or family background are unnecessary for this prototype and should not be used as recommendation features. If external labour-market data is introduced, its source, coverage, and date should be disclosed. Important decisions should remain under the control of the student with support from a qualified counsellor or mentor.

11.3 Future Enhancements

The project can be extended in stages:

1. add a web interface with profile forms, career cards, and an editable preference panel;
2. load role and skill data from a maintainable CSV/database and map skill names to a taxonomy such as ESCO;
3. use synonym dictionaries or natural-language embeddings to recognise related skill phrases;
4. add a skill-proficiency self-rating, projects, certifications, and coursework as optional evidence;
5. provide a personalised learning roadmap with courses, mile-stones, and portfolio projects; and
6. evaluate rankings with consent-based feedback, counsellor review, and explicit fairness checks before adding more complex machine-learning models.

12 Conclusion

This paper presents an Intelligent Career Recommendation Sys-tem that compares student skills, interests, and preferred do-main with structured career-role profiles. A weighted, content-based algorithm ranks the available roles and explains the result through component scores, missing skills, and suggested next steps. The sample Python code demonstrates the com-plete scoring flow and can be run without third-party depen-dencies. Three synthetic case-study profiles show how the top recommendation changes when the student's technical skills and interests change. Because the sample catalogue and weights are illustrative and have not been empirically validated, the current system should be regarded as a career-exploration aid rather than a predictor of career success. A larger role database, better skill normalisation, user feedback, usability testing, and fairness evaluation are the next steps towards a more useful real-world tool.

References

[1] O*NET Resource Center, "O*NET Interest Profiler," National Cen-ter for O*NET Development. Available: https://www.onetcenter. org/IP.html. Accessed: Oct. 29, 2026.
[2] European Commission, "The ESCO Classification," European Skills, Competences, Qualifications and Occupations, version 1.2.1. Avail-able: https://esco.ec.europa.eu/en/classification. Updated Dec. 10, 2025; accessed Oct. 29, 2026.
[3] D. Çelik Ertuğrul and S. Bitirim, "Job recommender systems: a systematic literature review, applications, open issues, and chal-lenges," *Journal of Big Data*, vol. 12, article 140, Jun. 2025, doi: 10.1186/s40537-025-01173-y.
[4] "Representing jobs and job seekers in AI-based recommender sys-tems: A literature review," *Engineering Applications of Arti-ficial Intelligence*, vol. 174, article 114450, Jun. 2026, doi: 10.1016/j.engappai.2026.114450.

© Author(s). This work is peer-reviewed, openly published, and permanently archived
This article is openly accessible and reusable with proper attribution.