**Project Name: *AI Resume Ranker – The Resume Ranker for HR Legends***

**1. Purpose of the Project**

The AI Resume Ranker project aims to simplify the job application process by helping users evaluate how well their resumes align with specific job descriptions. It provides a match score and actionable feedback, enabling users to optimize their resumes for better chances of success.

This tool is especially useful for:

- Job seekers looking to tailor their resumes to specific job postings.

- Recruiters quickly assessing candidate fit for a role.

**2. Functionality**

The project provides the following core features:

| **Feature** | **Description** |
| --- | --- |
| **Upload Resume** | PDF or DOCX file |
| **Job Description Input** | Recruiter pastes a job description |
| **AI Magic** | NLP model compares resume with JD |
| **Rank & Score (Match Calculation)** | calculate a match score (percentage) between the resume and job description**.** |
| **Bonus** | Based on the match score, users receive recommendations to improve their resumes. |
|  |  |

**3. Factors Used for Calculating Score and Feedback**

The project employs Natural Language Processing (NLP) techniques to evaluate the similarity between the resume and job description:

**1. TF-IDF Vectorization :**

- Converts textual data into numerical representations.

- Weights words based on their frequency and importance in the context**.**

**2. Cosine Similarity :**

- Measures the degree of similarity between the resume and the job description vectors.

- Returns a value between 0 (unrelated) and 1 (identical).

**Feedback Levels**

**1. High Match (> 80%) :**

- Indicates excellent alignment; the resume is well-suited for the job.

- Feedback: "🚀 Excellent match! Your resume aligns very well with the job description."

**2. Moderate Match (50% - 80%) :**

- Indicates potential alignment; minor improvements are needed.

- Feedback: "👍 Good match. Consider adding more role-specific keywords."

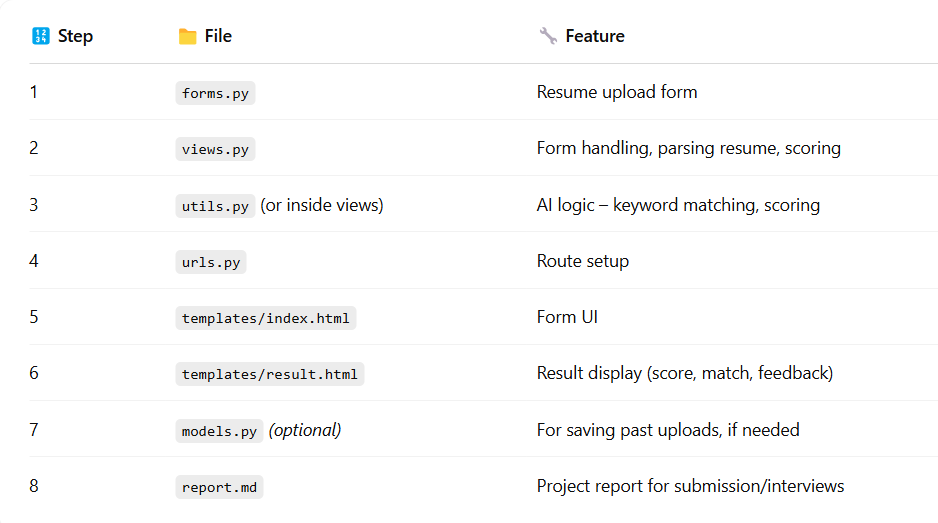
**3. Low Match (< 50 % ) :**

- Indicates poor alignment; significant tailoring is required.

- Feedback: "⚠️ Low match. You might need to tailor your resume more to the job."

**4. Structure of the Project**

**Project Files and Their Roles**



**Folder Structure**

AI-Resume-Ranker/

├── Ranker/

│ ├── templates/

│ │ ├── ranker/

│ │ │ ├── index.html

│ │ │ ├── result.html

│ ├── forms.py

│ ├── views.py

│ ├── utils.py

│ ├── urls.py

│ ├── models.py (Optional)

├── manage.py

├── db.sqlite3

**5. Files employed**

**🔹 Feature 1: Resume Upload Form**

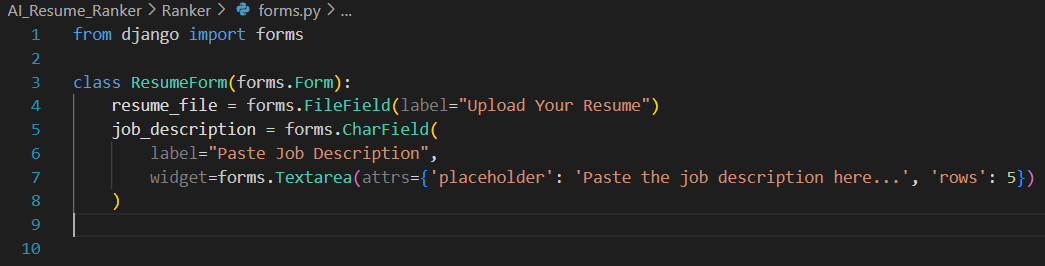
**🧾 Filename: forms.py**

**🧠 Purpose:**

To collect:

* A **resume file** from the user (PDF or DOCX)
* A **job description** (text input for comparison)

**🧑‍💻 Code:**



**🔍 Line-by-Line Explanation (Like You’re Telling the Interviewer)**

| **Line** | **What's Happening** | **How to Say it in Interview** |
| --- | --- | --- |
| from django import forms | Importing Django’s form tools | “I’m using Django’s built-in form system to structure and validate user inputs.” |
| class ResumeForm(forms.Form): | Creating a form class | “This form handles two key fields: resume file and job description.” |
| resume\_file = forms.FileField(...) | Accepts a file upload | “This field ensures users upload a file, useful for handling resumes in PDF or DOCX.” |
| job\_description = forms.CharField(...) | Text input with multiple lines | “This field accepts the JD as a block of text, which is later parsed for keyword matching.” |
| widget=forms.Textarea(...) | Makes it multiline with placeholder and rows | “To improve UX, I used a Textarea widget with a placeholder and custom height.” |

**📄 Report-Ready Description**

### Feature: Resume Upload Form

\*\*File:\*\* `forms.py`

\*\*Objective:\*\*

To create a user-friendly form that accepts:

- A resume file (PDF/DOCX)

- A job description for comparison

\*\*Explanation:\*\*

The `ResumeForm` class inherits from Django’s `forms.Form` and defines two fields:

- `resume\_file` uses `FileField` for secure resume uploads.

- `job\_description` uses `CharField` with a `Textarea` widget for better input readability.

\*\*Why This Matters:\*\*

This modular structure helps keep form-related logic separate from views, encourages cleaner code, and enables Django’s built-in form validation.

\*\*Key Benefits:\*\*

- File uploads handled securely

- Easy form rendering in templates

- Prevents invalid input via Django’s validation system

**Conclusion**

*“I separated my form logic from the views using Django’s forms.Form class, making it easier to maintain, reuse, and validate inputs like resume files and job descriptions. It’s clean, secure, and scalable.”*

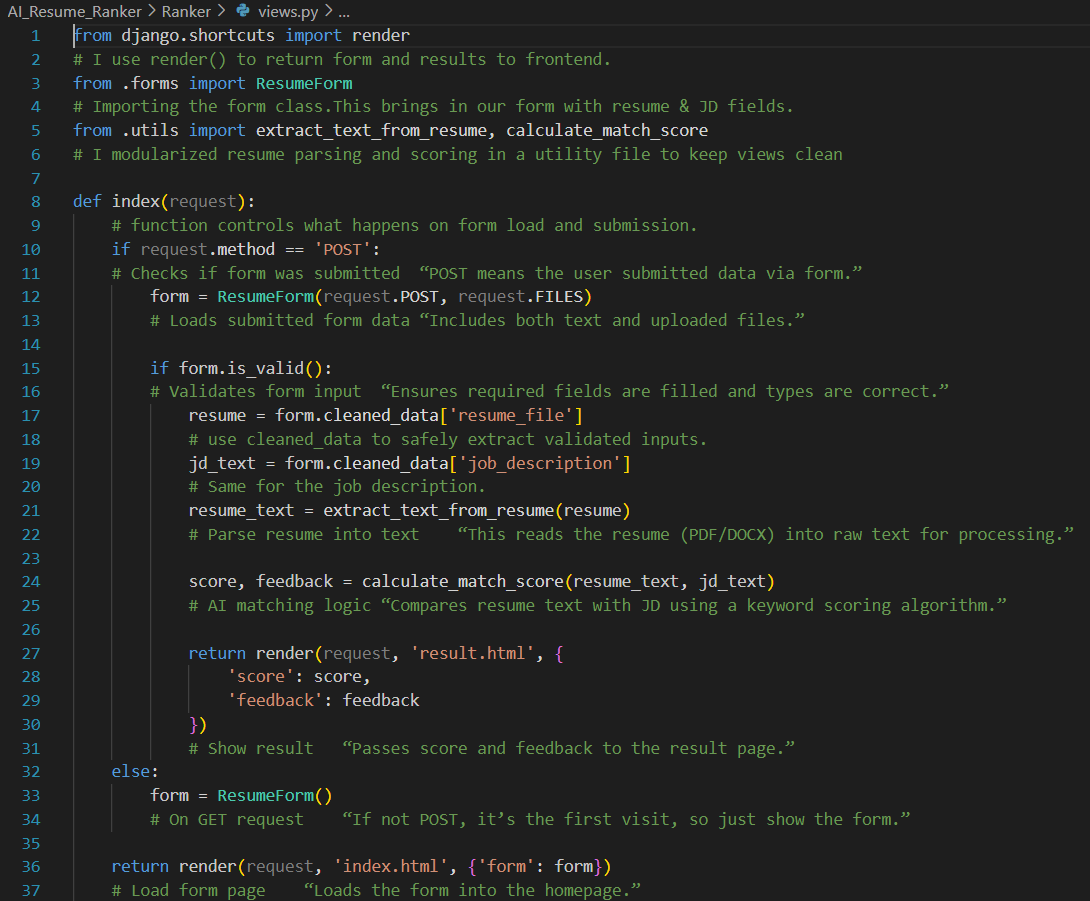
**🔹 Feature 2: Form Handling & Resume Scoring**

**🧾 Filename: views.py**

**🧠 Purpose:**

To handle the form input, process the resume file, compare it to the job description, and return a match score with feedback.

Code:



**Conclusion**

“I kept the view logic clean by offloading resume parsing and match scoring into utility functions. I handled file uploads using Django's request.FILES, validated input securely using cleaned\_data, and used the render() method to switch between form view and result display.”

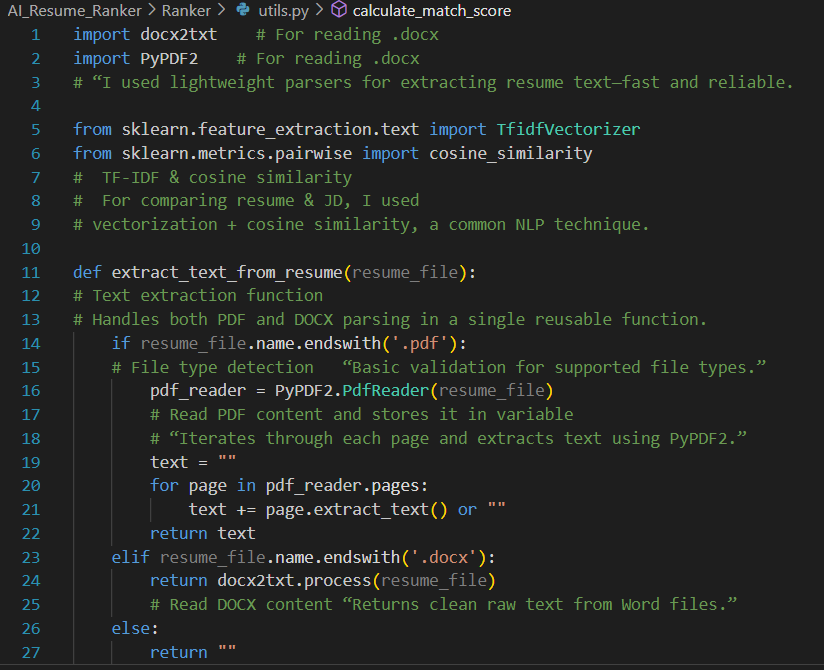
**🔹 Feature 3: Resume Parsing + Match Scoring**

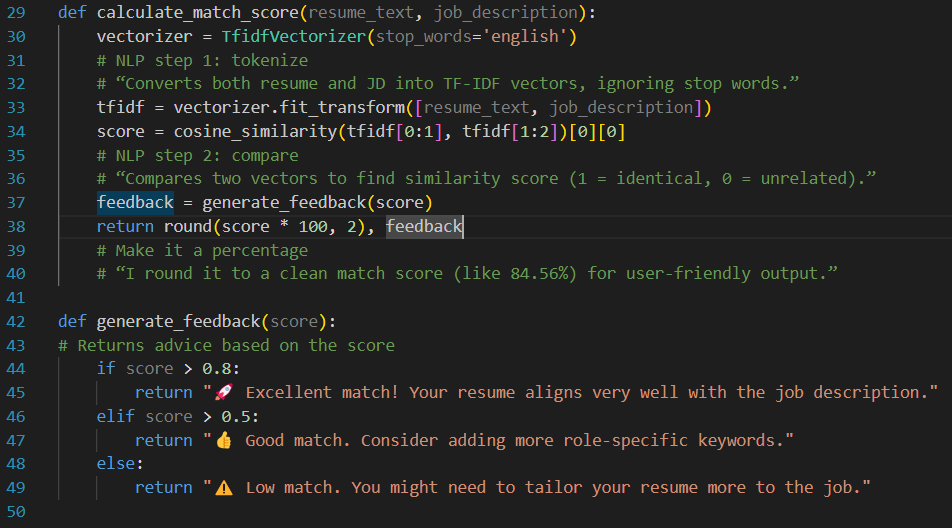
**🧾 Filename: utils.py**

**🧠 Purpose:**

1. Extract text from uploaded resumes (PDF/DOCX)
2. Compare that text with the job description
3. Calculate a **match score** and **custom feedback**

**🧑‍💻 Code:**

****

****

**Conclusion**

💬 *“For semantic comparison, I used TF-IDF and cosine similarity. These are lightweight NLP techniques that don’t need deep learning but still capture how similar two pieces of text are. I extracted resume content based on file type, making my utility flexible and extensible.”*

**6. Learnings**

**Technical Learnings**

1. **What are Django Forms?**

Django Forms are basically Python classes that:

* Define the **structure** of your form (fields, types, widgets)
* Handle **validation** (checking for required fields, file types, etc.)
* Clean and process form data
* Save time by reducing the amount of HTML + backend logic you’d otherwise write

💡 Think of them as your **smart front-desk assistant**:  
They receive input (resume & JD), validate it, and send clean data to the backend.

**🔍 Why Use Django Forms Instead of Raw HTML Forms?**

| **Raw HTML** | **Django Form** |
| --- | --- |
| Manual validation | Auto validation |
| Risky file handling | Safer file fields |
| No Python logic | Logic in Python |
| Static form | Dynamic, reusable form class |

**🧱 Anatomy of a Django Form**

Here’s the **basic structure** of a Django Form:

from django import forms

class MyForm(forms.Form):

name = forms.CharField(max\_length=100)

age = forms.IntegerField()

resume = forms.FileField()

is\_employed = forms.BooleanField(required=False)

**Field Types (Most Common & Interview-Worthy)**

| **Field Type** | **What it does** | **Example Use** |
| --- | --- | --- |
| CharField() | Text input | Name, Job Title |
| IntegerField() | Numeric input | Years of Experience |
| EmailField() | Email input with validation | Contact info |
| BooleanField() | Checkbox | “Currently working?” |
| FileField() | File uploads (PDF/DOCX) | Resume upload |
| ChoiceField() | Dropdown/select | Choose job role |
| Textarea() | Multiline input | Job Description |

**Field Attributes You Should Know**

| **Attribute** | **Meaning** | **Example** |
| --- | --- | --- |
| required=True | Must be filled | resume = forms.FileField(required=True) |
| widget=forms.Textarea | Controls UI style | job\_desc = forms.CharField(widget=forms.Textarea) |
| max\_length=100 | Max characters | Used in text fields |
| label='Upload Resume' | Friendly label name | Appears in form UI |

*"I used Django Forms to securely handle resume and job description inputs. It let me validate file types, manage text inputs, and kept my views clean by of*floading input processing to form classes."

1. **What is TF-IDF?**

**TF-IDF** stands for:

| **Term** | **Full Form** | **Meaning** |
| --- | --- | --- |
| **TF** | Term Frequency | How often a word appears in a document |
| **IDF** | Inverse Document Frequency | How rare the word is across all documents |

In simple words:  
🔍 TF-IDF tells you **which words are *important*** in a document, **but not too common** across all documents.

**🤖 Why TF-IDF?**

Let’s say you have 100 resumes, and you want to find which resume best matches a job description.

💩 Words like: "the", "and", "is" appear **everywhere** → not useful.  
🔥 Words like: "Python", "TensorFlow" appear **only in good resumes** → very useful.

TF-IDF helps identify those 🔥 words and gives them **more weight**!

**📈 How TF-IDF Works (Step-by-Step)**

Let’s break it down with a quick example.

**🗒️ Documents:**

* Resume 1: "Python developer with Django experience"
* Resume 2: "Java developer skilled in Spring"
* Job Desc: "Looking for a Python and Django expert"

**Step 1: TF = Term Frequency**

**Count how often a word appears** in each document.

| **Word** | **Resume 1** | **Resume 2** |
| --- | --- | --- |
| python | 1 | 0 |
| developer | 1 | 1 |
| django | 1 | 0 |
| java | 0 | 1 |

But we normalize it (divide by total words in document), so each word gets a score between 0 and 1.

**Step 2: IDF = Inverse Document Frequency**

Now check how **rare** the word is across *all* documents.

📉 If a word appears in **every document** → IDF is **low**  
📈 If a word appears in **only one** → IDF is **high**

Formula:

plaintext

CopyEdit

IDF(word) = log(Total Documents / Documents Containing the Word)

Example:

* "developer" appears in **2 of 2** → low IDF
* "django" appears in **1 of 2** → higher IDF

**Step 3: TF-IDF = TF × IDF**

Multiply both values:

* If a word is **frequent** in a resume ✅
* And **rare** across all documents ✅  
  → then it's **important**! 💥

**💡 So, Final Score:**

TF-IDF gives higher scores to words that:

* Are common in the current doc ✅
* Are rare in other docs ✅

**🧠 Mental Model**

Think of TF-IDF like **a filter in your brain**:

“Hey, this resume talks a LOT about 'TensorFlow' and the others don’t. That’s probably important.”

**📦 Used in:**

| **Area** | **Use Case** |
| --- | --- |
| NLP | Search engines (Google!) |
| AI | Resume vs JD matching (like your project!) |
| Chatbots | Understanding user input |
| Plagiarism | Finding similarities across texts |

**🎯 Interview QnA**

| **Question** | **Answer** |
| --- | --- |
| What is TF-IDF? | It scores words based on their importance in a document relative to a set of documents. |
| How is it used in NLP? | For keyword extraction, semantic similarity, and document comparison. |
| What does a high TF-IDF score mean? | The word is frequent in this doc, and rare in others. It's important. |
| What libraries support TF-IDF? | sklearn, gensim, TfidfVectorizer in Python. |
| Limitations? | Doesn’t understand meaning/context (e.g. “python” the snake 🐍 vs the language). |

C) **What is Cosine Similarity?**

It tells you **how similar two documents are**, **based on the angle** between them.

Sounds fancy, right? But here’s the simple breakdown:

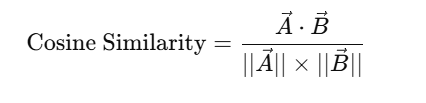
| **🧾 Input** | **📊 Output** |
| --- | --- |
| Two text vectors (e.g. Resume & JD in TF-IDF format) | A score from -1 to 1 |
| 👉 1 | They are **exactly the same** |
| 👉 0 | They are **completely different** |
| 👉 > 0.7 | Usually means **very relevant** |

Cosine similarity tells us **how close the angle between them is**.

* If the angle is **0°** → They are **identical** → Cosine = 1 ✅
* If the angle is **90°** → They are **totally unrelated** → Cosine = 0 ❌

**Formula of Cosine Similarity**

Where:



* A⃗⋅B⃗\vec{A} \cdot \vec{B}A⋅B is the **dot product** of vectors (TF-IDF scores).
* ∣∣A⃗∣∣||\vec{A}||∣∣A∣∣ is the **magnitude** (length) of vector A.
* Result is always between **0 and 1**.

|  |  |
| --- | --- |
| What value does cosine similarity return? | From **0 (no match)** to **1 (perfect match)**. |

|  |  |
| --- | --- |
| What if two resumes have same cosine score? | You might break the tie using **keywords**, **experience**, or **other features**. |

**Future Improvements**

1. Advanced NLP Models:

- Integrate pre-trained models like BERT or GPT for better semantic understanding.

2. Support for Additional File Formats:

- Add support for `.rtf`, `.txt`, and image-based resumes (using OCR).

3. Database Integration:

- Store past results for analysis and allow users to retrieve their history.

4. Interactive UI:

- Enhance the frontend with frameworks like React or Bootstrap.

5. Multi-Language Support :

- Provide resume scoring for job descriptions in languages other than English.

**Conclusion**

The **AI Resume Ranker** project demonstrates the application of AI in solving real-world problems, like improving job application success rates. By combining text parsing, TF-IDF, and Cosine Similarity, this tool provides a simple yet powerful way to assess resume alignment with job descriptions. Future iterations can further enhance its capabilities, making it even more valuable for job seekers and recruiters alike.