resume-analyzer/

resume-analyzer/

**├── 1. app.py**

**├── 2. resume\_parser.py**

**├── 3. utils.py**

**├── 4. templates/**

│ ├── index.html

│ ├── results.html

│ └── error.html

**├── 5. static/**

│ **├── 6. css/**

│ │ └── styles.css

│ └── js/

│ └── scripts.js

**├── 7. uploads/**

│ └── (temporary PDFs)

**├── 8. requirements.txt**

**├── 9. README.md**

**TF-IDF + cosine similarity with a simple example.**

**GOAL: Match Resume to Job Description using TF-IDF**

TF-IDF = Term Frequency-Inverse Document Frequency  
Cosine Similarity = How **similar** two texts are in meaning, based on word usage.

**Example**

**Resume (skills):**

Python, Machine Learning, Data Analysis, Pandas

**Job Description:**

Looking for a Data Analyst with Python and Machine Learning experience. Familiarity with Pandas and data wrangling is a plus.

**Old method: Basic keyword matching**

Count how many exact words match.

Matched words: Python, Machine Learning, Pandas → **Score: 3**

❌ **Limitation**: If resume had “Data Analysis” but JD used “data wrangling”, it won't count — even though they mean similar things.

**New method: TF-IDF + Cosine Similarity**

TF-IDF converts both texts into **vectors** (numerical format) based on how unique each word is.

Then cosine similarity checks how close the two vectors are (angle between them).

So even if words aren’t *exact matches*, their importance and context are considered.

**Outcome**

* Score: **85%** similarity → shows **good match**
* Feedback: “You're missing these key terms from JD: wrangling, analyst” → helps improve resume!

Here’s a line-by-line explanation of your code with a focus on **where and how NLP is being used**:

*# Imports*

import spacy

import re

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

* *spacy:* ***Used for NLP tasks*** *like Named Entity Recognition (NER) and sentence segmentation.*
* *re: For extracting emails and phone numbers using regular expressions.*
* *TfidfVectorizer: Converts text into TF-IDF matrix (used for text similarity).*
* *cosine\_similarity: Measures how similar two texts are (resume vs. job description).*

*🔹 Class Initialization*

class ResumeParser:

def \_\_init\_\_(self):

self.nlp = spacy.load("en\_core\_web\_sm")

self.skills\_db = ['Python','Java', 'SQL', 'Machine Learning', 'Data Analysis']

*Loads SpaCy’s English NLP model. This will allow the use of NLP features like:*

*Named Entity Recognition (NER)*

*Sentence tokenization*

*skills\_db is a list of target keywords to match from resume text.*

*🔹 Entity Extraction Method*

def extract\_entities(self, text):

doc = self.nlp(text)

* *NLP Step: Converts raw resume text into a SpaCy Doc object.*
* *Enables access to named entities and sentence tokens.*

entities = {

'name': None,

'email': None,

'phone': None,

'skills': [],

'education': []

}

* *Initializes a dictionary to store extracted data*
* *Named Entity Recognition (NLP)*

for ent in doc.ents:

if ent.label\_ == 'PERSON' and not entities['name']:

entities['name'] = ent.text

break

* *NLP Step: Identifies person names using NER (ent.label\_ == 'PERSON').*
* *Assumes the first person entity is the candidate's name.*

*🔸 Email Extraction (Regex)*

email\_pattern = r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'

emails = re.findall(email\_pattern, text)

entities['email'] = emails[0] if emails else None

*Phone Extraction (Regex)*

phone\_pattern = r"\(?\d{3}\)?[-.\s]?\d{3}[-.\s]?\d{4}"

phones = re.findall(phone\_pattern, text)

entities['phone'] = phones[0] if phones else None

*Skill Matching (Simple keyword match)*

for skill in self.skills\_db:

if skill.lower() in text.lower():

entities['skills'].append(skill)

* *Not NLP-based — just basic keyword search in lowercase.*

*Education Extraction (NLP: Sentence Parsing)*

degree\_keywords = ['bachelors', 'masters', 'phd', 'degree']

for sent in doc.sents:

if any(keyword in sent.text.lower() for keyword in degree\_keywords):

entities['education'].append(sent.text.strip())

* *NLP Step: Uses SpaCy sentence tokenization (doc.sents).*
* *Looks for education-related keywords in each sentence.*

*🔹 Resume Scoring Function*

def score\_resume(self, entities, job\_description):

resume\_text = ' '.join(entities['skills']) or ''

corpus = [resume\_text, job\_description]

* *Combines extracted skills into one string (resume\_text) and pairs with job description.*

**🔸** *TF-IDF + Cosine Similarity (ML-based scoring)*

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(corpus)

similarity = cosine\_similarity(tfidf\_matrix[0:1], tfidf\_matrix[1:2])[0][0] \* 100

* *Converts both texts into vector format using TF-IDF.*
* *Measures their semantic similarity using cosine similarity.*

*🔸 Feedback Generation*

feedback = f"TF-IDF similarity score: {round(similarity, 2)}%."

if similarity < 50:

missing = set(job\_description.lower().split()) - set(resume\_text.lower().split())

feedback += f" Add or highlight relevant keywords like: {', '.join(list(missing)[:5])}"

* *Gives a feedback message. If the match is poor, shows missing keywords.*

return {"score": round(similarity, 2), "feedback": feedback}

* *Returns score and suggestions for improving the resume.*

| **Feature** | **NLP Used?** | **Technique** |
| --- | --- | --- |
| Name Extraction | ✅ | NER with SpaCy |
| Sentence Parsing (Education) | ✅ | Sentence segmentation with SpaCy |
| Skill Matching | ❌ | Manual keyword match |
| Email/Phone Extraction | ❌ | Regex |
| Resume Scoring | ✅ (indirect) | TF-IDF vectorization (ML-based text similarity) |

**line-by-line explanation** of the resume\_parser.py code that uses **spaCy for entity extraction** and **TF-IDF for resume scoring**.

**1–5: Import Libraries**

import spacy

🔹 Loads the spaCy NLP library. Used to process natural language and extract information like names, education, etc.

import re

🔹 Imports the regular expressions module. Used to match patterns like email addresses and phone numbers.

from sklearn.feature\_extraction.text import TfidfVectorizer

🔹 Loads the **TF-IDF tool** from scikit-learn.  
It transforms text into numbers based on word importance across documents.

from sklearn.metrics.pairwise import cosine\_similarity

🔹 This computes **cosine similarity**, a score between 0 and 1 that shows how similar two pieces of text are.

**🧾 6–9: Initialize the Parser Class**

class ResumeParser:

def \_\_init\_\_(self):

self.nlp = spacy.load("en\_core\_web\_sm")

self.skills\_db = ['Python','Java', 'SQL', 'Machine Learning', 'Data Analysis']

🔸 \_\_init\_\_: This runs when a new parser is created.

* self.nlp: Loads the English spaCy model for Named Entity Recognition (NER).
* self.skills\_db: A basic list of tech skills to match in the resume.

**10–44: Extract Info From Resume Text**

def extract\_entities(self, text):

doc = self.nlp(text)

Applies spaCy NLP processing to the input resume text.

entities = {

'name': None,

'email': None,

'phone': None,

'skills': [],

'education': []

}

🔹 Initializes a dictionary to store extracted info.

for ent in doc.ents:

if ent.label\_ == 'PERSON' and not entities['name']:

entities['name'] = ent.text

break

🔹 Finds the **first name** detected by spaCy’s NER. Assigns it to name.

email\_pattern = r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'

phone\_pattern = r"\(?\d{3}\)?[-.\s]?\d{3}[-.\s]?\d{4}"

🔹 These regular expressions are for matching email and phone formats.

emails = re.findall(email\_pattern, text)

phones = re.findall(phone\_pattern, text)

🔹 Uses re.findall() to search the text for all emails and phones.

entities['email'] = emails[0] if emails else None

entities['phone'] = phones[0] if phones else None

🔹 Picks the **first** email and phone found (if any).

for skill in self.skills\_db:

if skill.lower() in text.lower():

entities['skills'].append(skill)

🔹 Matches any skills from the list that are present in the resume text (case-insensitive).

degree\_keywords = ['bachelors', 'masters', 'phd', 'degree']

for sent in doc.sents:

if any(keyword in sent.text.lower() for keyword in degree\_keywords):

entities['education'].append(sent.text.strip())

🔹 Checks each sentence to find education-related words like "bachelor's", "PhD", etc.

return entities

🔹 Returns all the extracted info as a dictionary.

**🧾 45–62: TF-IDF-Based Resume Scoring (UPDATED SECTION)**

def score\_resume(self, entities, job\_description):

🔹 New method that takes resume entities + job description as input.

resume\_text = ' '.join(entities['skills']) or ''

🔹 Joins all extracted skills into one string.  
If skills are missing, fallback to empty string.

corpus = [resume\_text, job\_description]

The **corpus** is a list with 2 items: the resume and job description.  
TF-IDF will compare these two.

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(corpus)

Converts the text into TF-IDF vectors:

* fit\_transform() learns which words are important.
* Result: a numeric matrix showing how relevant each word is in both texts.

similarity = cosine\_similarity(tfidf\_matrix[0:1], tfidf\_matrix[1:2])[0][0] \* 100

Compares the **resume vs. job description** using cosine similarity.

* Result: similarity score between 0–100%

feedback = f"TF-IDF similarity score: {round(similarity, 2)}%."

Creates a feedback message showing the score.

if similarity < 50:

missing = set(job\_description.lower().split()) - set(resume\_text.lower().split())

feedback += f" Add or highlight relevant keywords like: {', '.join(list(missing)[:5])}"

If the score is low, it:

* Finds which **keywords are in the job description but not in the resume**
* Suggests a few of them as missing keywords

return {"score": round(similarity, 2), "feedback": feedback}

Final return: a dictionary with the score and suggestions.

**Summary of Key Updates (TF-IDF Section)**

| **🔧 Part** | **Purpose** |
| --- | --- |
| TfidfVectorizer() | Converts resume & job description into numeric word weights |
| cosine\_similarity() | Measures how similar the texts are |
| feedback | Shows match percentage and missing keywords |

**Updated app.py as below:**

@app.route('/upload', methods=['POST'])

def upload\_resume():

if 'resume' not in request.files:

return render\_template("error.html", message="No file uploaded.")

file = request.files['resume']

if file.filename == '':

return render\_template("error.html", message="No file selected.")

if file and file.filename.lower().endswith('.pdf'):

filename = secure\_filename(file.filename)

file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

file.save(file\_path)

text = extract\_text\_from\_pdf(file\_path)

if text.startswith("Error:"):

return render\_template("error.html", message=text)

# ✅ NEW: Get job description from form

job\_description = request.form['job\_description']

# ✅ Pass job description to parser

entities = parser.extract\_entities(text)

score\_data = parser.score\_resume(entities, job\_description)

os.remove(file\_path)

return render\_template('results.html', entities=entities, score\_data=score\_data)

return render\_template("error.html", message="Invalid file format. Please upload a PDF file.")