**Code:**

import streamlit as st

import pandas as pd

import re

import PyPDF2

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KNeighborsClassifier

from sklearn.multiclass import OneVsRestClassifier

from collections import Counter

# Function to extract text from PDF

def extract\_text\_from\_pdf(pdf\_file):

    pdf\_reader = PyPDF2.PdfReader(pdf\_file)

    text = "".join([page.extract\_text() for page in pdf\_reader.pages if page.extract\_text()])

    return text

# Function to extract email from text

def extract\_email(text):

    emails = re.findall(r'[\w\.-]+@[\w\.-]+', text)

    return emails[0] if emails else "N/A"

# Function to extract name (Assumption: First two words are name)

def extract\_name(text):

    words = text.split()

    return " ".join(words[:2]) if words else "N/A"

# Function to clean resume text

def clean\_resume(text):

    text = re.sub(r'http\S+', ' ', text)

    text = re.sub(r'[^a-zA-Z ]', ' ', text)

    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Load pre-trained classification dataset

def load\_training\_data():

    df = pd.read\_csv("UpdatedResumeDataSet.csv", encoding='utf-8')

    df['cleaned\_resume'] = df['Resume'].apply(clean\_resume)

    le = LabelEncoder()

    df['Category'] = le.fit\_transform(df['Category'])

    vectorizer = TfidfVectorizer(sublinear\_tf=True, stop\_words='english')

    X = vectorizer.fit\_transform(df['cleaned\_resume'])

    y = df['Category']

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

    model = OneVsRestClassifier(KNeighborsClassifier())

    model.fit(X\_train, y\_train)

    return model, vectorizer, le

# Streamlit UI

st.title("AI-powered Resume Screening & Ranking System")

uploaded\_files = st.file\_uploader("Upload Resume PDFs", accept\_multiple\_files=True, type=["pdf"])

if uploaded\_files:

    job\_description = st.text\_area("Enter Job Description:")

    if st.button("Process Resumes"):

        model, vectorizer, label\_encoder = load\_training\_data()

        tfidf\_vectorizer = TfidfVectorizer()

        job\_desc\_vector = tfidf\_vectorizer.fit\_transform([job\_description])

        results = []

        for pdf in uploaded\_files:

            resume\_text = extract\_text\_from\_pdf(pdf)

            cleaned\_text = clean\_resume(resume\_text)

            name = extract\_name(cleaned\_text)

            email = extract\_email(cleaned\_text)

            # Predict Job Category

            vectorized\_input = vectorizer.transform([cleaned\_text])

            predicted\_category = label\_encoder.inverse\_transform(model.predict(vectorized\_input))[0]

            # Calculate Similarity

            resume\_vector = tfidf\_vectorizer.transform([cleaned\_text])

            similarity = cosine\_similarity(job\_desc\_vector, resume\_vector)[0][0]

            results.append((name, email, similarity, predicted\_category))

        # Rank resumes

        results.sort(key=lambda x: x[2], reverse=True)

        # Display results

        df\_results = pd.DataFrame(results, columns=["Name", "Email", "Similarity", "Predicted Category"])

        df\_results.insert(0, "Rank", range(1, len(df\_results) + 1))

        st.write(df\_results)

        # Save results to CSV

        df\_results.to\_csv("ranked\_resumes.csv", index=False)

        st.success("Results saved as ranked\_resumes.csv")

**Step-by-Step Guide: AI-powered Resume Screening & Ranking System**

**1. Project Overview**

This project automates resume screening and ranking using Natural Language Processing (NLP) and Machine Learning. It takes multiple resumes as input, analyzes their content, predicts job categories, and ranks them based on their similarity to a given job description.

**2. Technologies Used**

* Python
* Pandas
* Streamlit (for UI)
* Scikit-learn
* TfidfVectorizer
* PyPDF2 (for PDF processing)
* Regular Expressions (for extracting emails & names)

**3. Project Workflow**

1. **Input:** Upload multiple PDF resumes.
2. **Processing:** Extract text, name, and email from resumes.
3. **Resume Screening:** Predict the most relevant job category for each resume.
4. **Resume Ranking:** Compute similarity scores between resumes and the job description.
5. **Output:** Display ranked resumes with Name, Email, Similarity Score, and Predicted Category.
6. **Save Results:** Export the results to a CSV file.

**4. Implementation Details**

**4.1. Extracting Information from PDFs**

* Extract text from PDF resumes.
* Extract emails using regex.
* Extract names based on assumptions (first two words).
* Clean resume text for further processing.

**4.2. Resume Screening (Job Category Prediction)**

* Load a pre-trained dataset (UpdatedResumeDataSet.csv) with categorized resumes.
* Clean and vectorize resume text using TfidfVectorizer.
* Train a KNeighborsClassifier model for job classification.
* Predict the category of new resumes.

**4.3. Resume Ranking (Similarity Calculation)**

* Accept a job description as input.
* Convert job description and resumes into numerical vectors using TfidfVectorizer.
* Compute cosine similarity scores.
* Rank resumes based on similarity scores.

**5. Streamlit UI Workflow**

**5.1. User Inputs**

* Upload multiple PDF resumes.
* Enter job description.

**5.2. Processing**

* Extract text, name, and email from resumes.
* Predict job category.
* Compute similarity score against job description.

**5.3. Displaying Results**

* Show results in a table with **Rank, Name, Email, Similarity Score, and Predicted Category**.
* Save results to ranked\_resumes.csv.

**6. How to Run the Project**

**6.1. Install Dependencies**

pip install streamlit pandas scikit-learn PyPDF2

**6.2. Run the Streamlit App**

streamlit run app.py

**6.3. Upload Resumes & Enter Job Description**

* Click on the file uploader to upload PDF resumes.
* Enter a job description in the text box.
* Click **Process Resumes** to analyze and rank them.

**6.4. View & Download Results**

* The ranked resumes will be displayed in the Streamlit app.
* A CSV file (ranked\_resumes.csv) will be generated for further use.

**7. Expected Output**

| **Rank** | **Name** | **Email** | **Similarity** | **Predicted Category** |
| --- | --- | --- | --- | --- |
| 1 | Alice | [alice@email.com](mailto:alice@email.com) | 0.87 | Data Scientist |
| 2 | Bob | [bob@email.com](mailto:bob@email.com) | 0.75 | NLP Engineer |
| 3 | Carol | [carol@email.com](mailto:carol@email.com) | 0.68 | Machine Learning |

**8. Future Enhancements**

* Improve entity extraction (e.g., better name extraction methods).
* Use a deep-learning-based model for job category prediction.
* Enhance UI with more features (filtering, visualization, etc.).
* Support additional resume formats (DOCX, TXT, etc.).

**9. Conclusion**

This project automates resume screening and ranking, making recruitment more efficient. By integrating NLP and machine learning, we can process and evaluate resumes quickly, saving time for HR professionals.

This guide provides a structured roadmap for implementing the AI-powered Resume Screening & Ranking System.   
  
