Automated Resume Screening System

Project Description:

The Automated Resume Screening System is a machine-learning-based application that intelligently interprets resumes, identifies their important elements, and evaluates candidates according to job requirements. The main intention is to replace manual screening by automatically extracting skills, qualifications, and work details, followed by predicting suitability using NLP and classification models.

This application reduces manual workload, assists HR departments in shortlisting profiles more quickly, and provides a consistent evaluation method. Instead of reading resumes manually, this system performs automated document processing and delivers instant classification results.

Project Scenarios

Scenario 1: Recruitment Filtering

Organizations regularly receive a huge number of applications. The system reads these resumes, pulls out key fields such as relevant skills, experience, academic background, and relevant projects, and categorizes the resume as eligible or not suitable for the mentioned role.

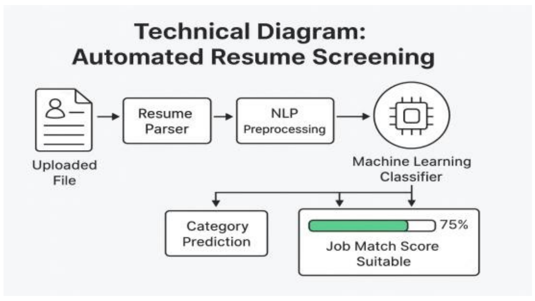
Scenario 2: Campus Hiring

During mass campus drives, thousands of students upload resumes. This system separates them based on skill relevance, certifications, academic performance, and domain match, helping a recruitment team identify and filter high-matching candidates quickly.

Scenario 3: Job Portal Skill Matching

Job portals can incorporate this system to evaluate resumes automatically. When a job seeker uploads their CV, the algorithm calculates suitability and provides a matching percentage, along with possible job roles and recommendations.

Technical Diagram:



Prerequisites:

Software Requirements:

- Jupyter Notebook
- Python 3.10+
- VS Code (optional)

Required Python Packages:

- numpy
- pandas
- scikit-learn
- nltk / spacy
- matplotlib
- seaborn
- pickle-mixin
- Flask

Prior Knowledge:

- Text preprocessing (tokenization, stopwords, stemming, TF-IDF)
- Classification algorithms (Logistic Regression, SVM, Random Forest, Naive Bayes)
- Evaluation metrics
- Flask basics for deployment

Project WorkFlow:

- User uploads a PDF/DOCX resume
- System extracts text
- NLP preprocessing (cleaning, stopwords removal, lemmatization)
- Feature extraction using TF-IDF / CountVectorizer
- ML model predicts: Suitable / Not Suitable

HR dashboard displays result and resume match score.

Project Activities:

1. Data Preparation:

- Manual resume dataset collected and labelled
- Duplicate resume text removed
- Text cleaned
- Stop words removed
- Lemmatization applied
- Converted resume text into TF-IDF vectors

2. Exploratory Data Analysis:

Performed:

- resume text statistics
- category-wise distribution
- keywords frequency
- visual plots
- sample word clouds

3. Model Building:

- TF-IDF representation applied to resize text
- Logistic Regression used for classification
- Hyperparameters tuned
- Performance tested using metrics

4. Performance Testing & Model Selection:

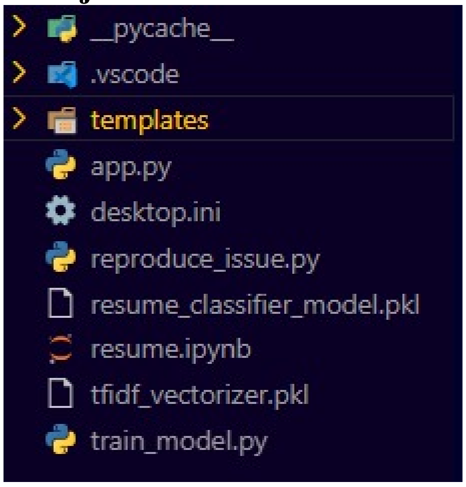
Performance tested using:

- Accuracy
- Precision
- Recall
- F-score
- Confusion matrix observations

5. Deployment:

- Model saved using pickle
- TF-IDF saved
- Backend created in Flask
- Web interface implemented
- Job suitability and match score displayed

Project Structure:



```
> __pycache__
> .vscode
> templates
  app.py
  desktop.ini
  reproduce_issue.py
  resume_classifier_model.pkl
  resume.ipynb
  tfidf_vectorizer.pkl
  train_model.py
```

Project Structure Explanation:

- templates

  Contains the HTML templates used by the application (mainly in Flask or Streamlit with HTML support).

- index.html

  Defines the structure of the web pages, user interface, input fields, and result display components.

- app.py

  This is the main backend file that runs the resume classification

- system.resume_classifier_model.pkl

  This file contains the trained machine learning model saved using Pickle.

- tfidf_vectorizer.pkl

  This file stores the trained TF-IDF vectorizer used to convert the resume text into numerical feature vectors.

- resume.ipynb

  This Jupyter notebook contains the complete model development process

- Dataset Folder.

  Contains the dataset used for model training.

Milestone 1: Data Collection & Preparation:

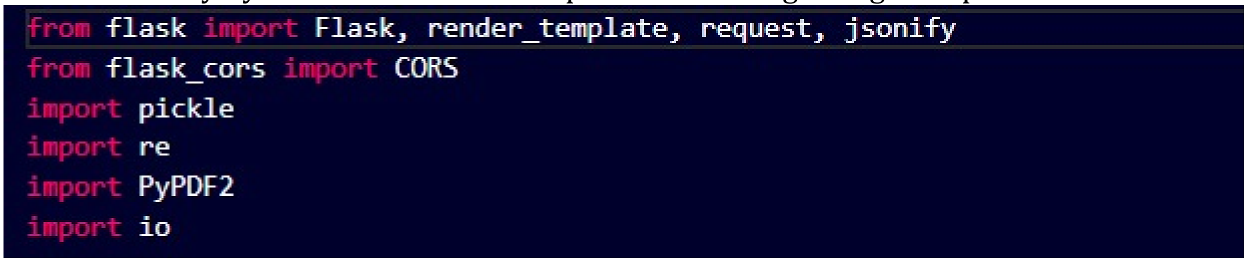
The initial phase of this project deals with preparing the dataset used for model learning. For this experiment, the resume files were manually collected and grouped into different professional categories such as IT, HR, Data Science, Web Development, etc. Each resume was tagged with a corresponding label and later processed to extract meaningful textual information from PDF documents.

Activity 1.1: Collect Dataset

Instead of using pre-built datasets from public sources, the resume files were gathered manually. All collected resumes were arranged into a structured folder so that they could be easily utilized during preprocessing and model building.

Activity 1.2: Import Libraries

The necessary Python modules were imported at the beginning of implementation.



```
from flask import Flask, render_template, request, jsonify
from flask_cors import CORS
import pickle
import re
import PyPDF2
import io
```

Role of each library:

- **pickle** – storing and retrieving model objects
- **re** – regular expression-based text cleaning
- **PyPDF2** – reading and extracting PDF content
- **io** – handling byte-stream data
- **flask** – web framework for backend development
- **render_template** – displaying HTML pages
- **request** – receiving input and file uploads
- **jsonify** – converting response data into JSON format

Activity 1.3: Read and Explore Data

In this task:

- raw resume text was loaded
- category labels were assigned
- overall dataset samples were inspected

Activity 1.4: Data Preparation

Before we can use our data to teach our machine-learning model, we need to clean it up. This includes:

- Remove HTML tags, punctuation, numbers
- Convert to lowercase
- Remove stopwords
- Apply lemmatization

Activity 1.5: Handling Missing Values

1. Skip unreadable or null PDF text

If extract_text_from_pdf() returns None due to an unreadable or corrupted file, that resume is ignored to prevent system crashes.

2. Ignore resumes with empty extracted content

If the extracted text is empty (e.g., scanned PDFs with no text), the system skips processing to avoid passing null text into the TF-IDF model.

3. Default handling inside text-based resumes

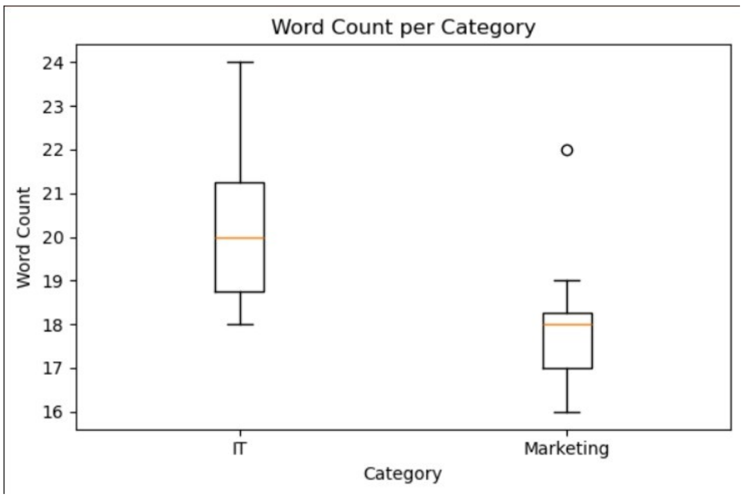
When .txt or .pdf resumes fail to decode or extract properly, no further processing is attempted, ensuring no null value reaches classification.

4. Internally normalize missing text fields

Even if parts of the resume such as skills, projects, or experience sections are absent, the keyword extractors return safe default values (e.g., empty lists or count zero), preventing null-related errors.

ACTIVITY 1.6: Checking for outliers:

Create boxplots to visualize outliers in numerical features:



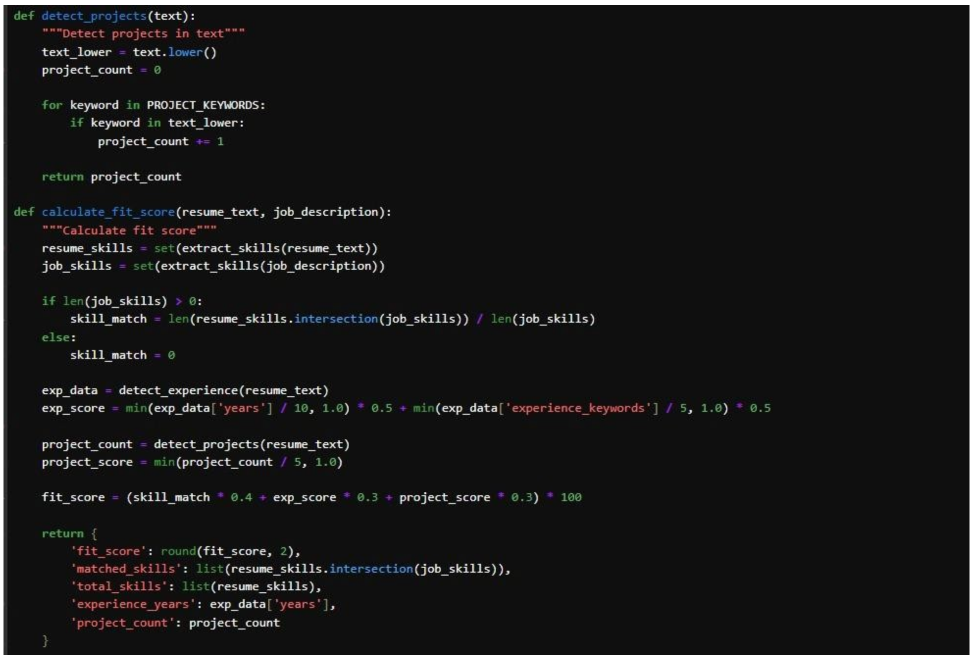
Boxplots help identify outliers, which are data points that fall outside the typical range. These can affect model performance if not handled properly.

Milestone 2: Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial step in understanding the characteristics, patterns, and relationships within the dataset. It helps in making informed decisions about feature engineering and model selection.

Activity 2.1: Descriptive Statistics

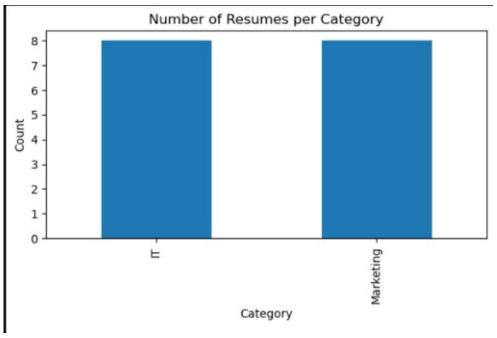
- Number of resume categories
- Count of resumes per class
- Length of text per resume



```python
def detect_projects(text):
    """Detect projects in text"""
    text_lower = text.lower()
    project_count = 0

    for keyword in PROJECT_KEYWORDS:
        if keyword in text_lower:
            project_count += 1

    return project_count

def calculate_fit_score(resume_text, job_description):
    """Calculate fit score"""
    resume_skills = set(extract_skills(resume_text))
    job_skills = set(extract_skills(job_description))

    if len(job_skills) > 0:
        skill_match = len(resume_skills.intersection(job_skills)) / len(job_skills)
    else:
        skill_match = 0

    exp_data = detect_experience(resume_text)
    exp_score = min(exp_data['years'] / 10, 1.0) * 0.5 + min(exp_data['experience_keywords'] / 5, 1.0) * 0.5

    project_count = detect_projects(resume_text)
    project_score = min(project_count / 5, 1.0)

    fit_score = (skill_match * 0.4 + exp_score * 0.3 + project_score * 0.3) * 100

    return {
        'fit_score': round(fit_score, 2),
        'matched_skills': list(resume_skills.intersection(job_skills)),
        'total_skills': list(resume_skills),
        'experience_years': exp_data['years'],
        'project_count': project_count
    }
```

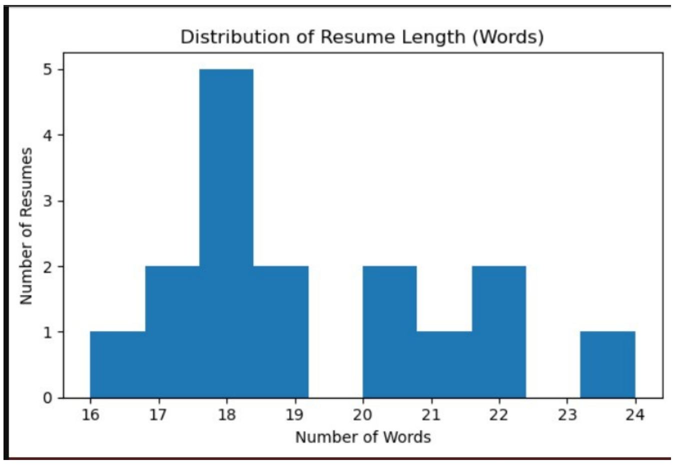
Activity 2.2: Visual Analysis

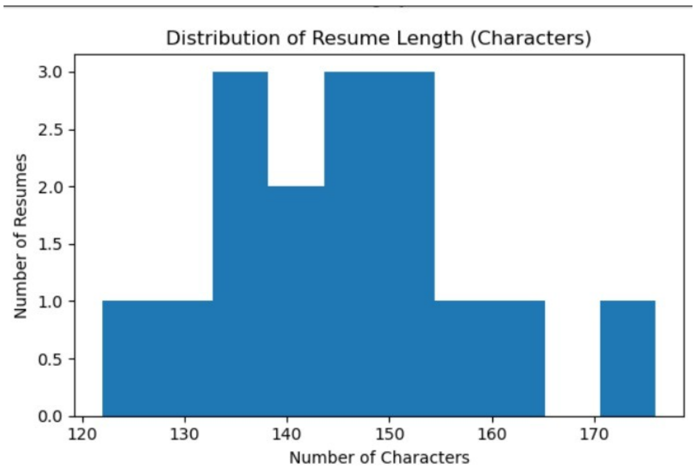
- Category distribution bar chart
- Word cloud for each resume class
- Skills frequency plot



Activity 2.3: Univariate Analysis

- Distribution of resume text length
- Category-wise sample count





Activity 2.4: Bivariate Analysis

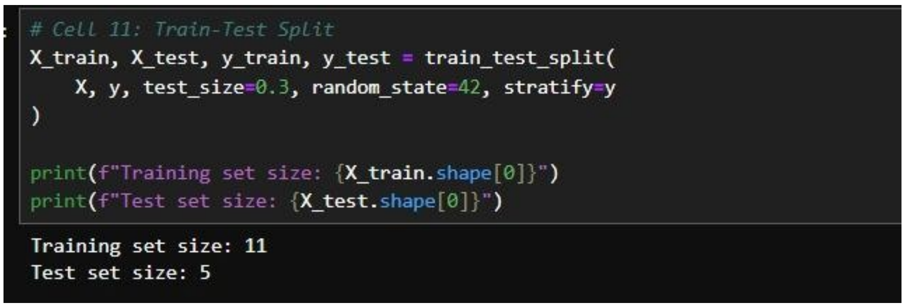
- Correlation between text length and classification
- Skill overlap between categories
-

Activity 2.5: Correlation Heatmap

Not numerical features, but TF-IDF vectors can show cosine similarity

Activity 2.8: Train-Test Split

Split the dataset into training and testing sets:



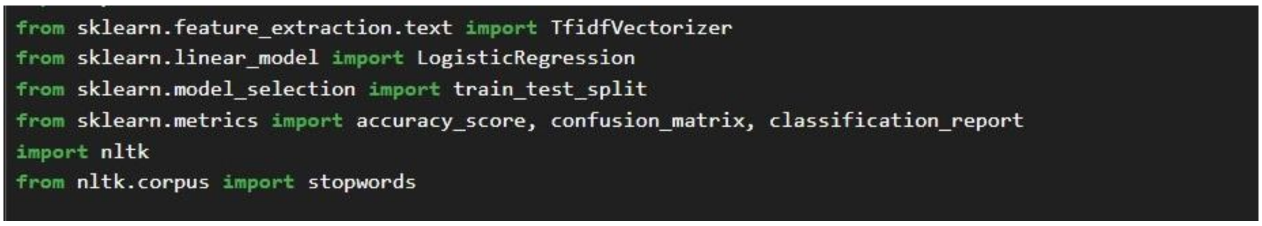
```
# Cell 11: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"Training set size: {X_train.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

```
Training set size: 11
Test set size: 5
```

Training set (80%): Used to train the model
- Testing set (20%): Used to evaluate model performance
- random_state=42: Ensures reproducibility

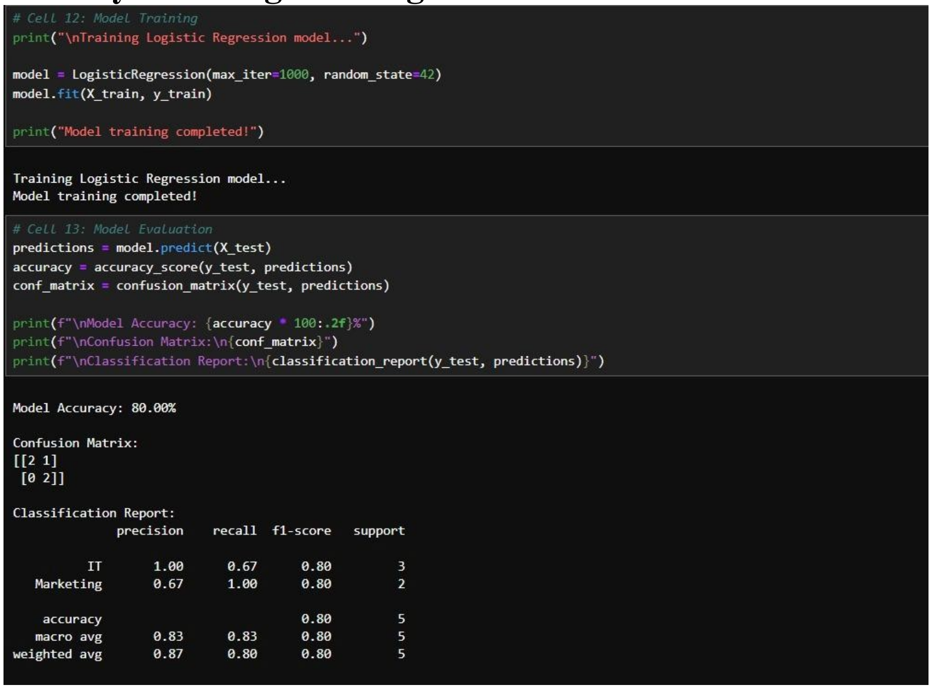
Milestone 3 : Feature Engineering & Model Building:

Activity 3.1: Import required libraries



```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
```

Activity 3.2: Logistic Regression Model



```
# Cell 12: Model Training
print("\nTraining Logistic Regression model...")

model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train, y_train)

print("Model training completed!")
```

```
Training Logistic Regression model...
Model training completed!
```

```
# Cell 13: Model Evaluation
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
conf_matrix = confusion_matrix(y_test, predictions)

print(f"\nModel Accuracy: {accuracy * 100:.2f}%")
print(f"\nConfusion Matrix:\n{conf_matrix}")
print(f"\nClassification Report:\n{classification_report(y_test, predictions)}")
```

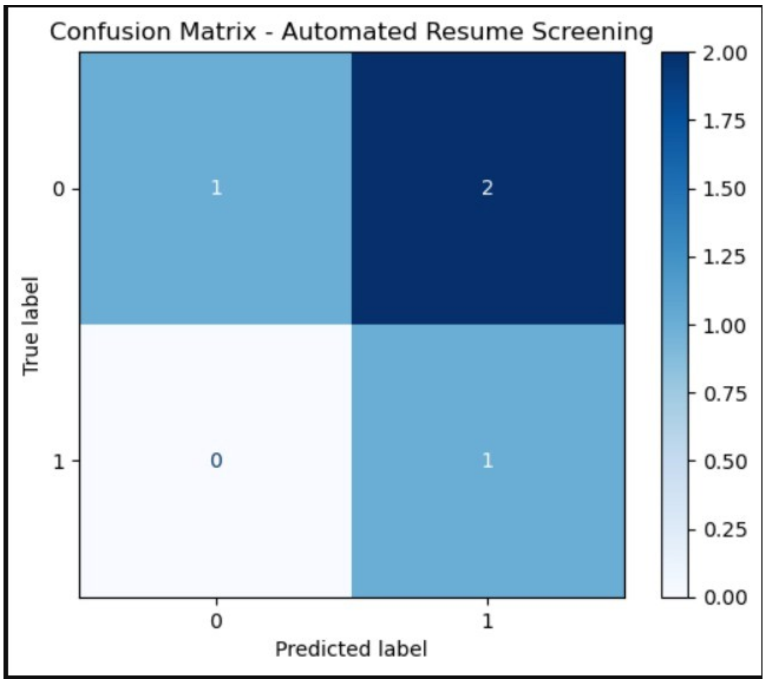
```
Model Accuracy: 80.00%

Confusion Matrix:
[[2 1]
 [0 2]]

Classification Report:
              precision    recall  f1-score   support

          IT       1.00      0.67      0.80         3
   Marketing       0.67      1.00      0.80         2

    accuracy                           0.80         5
   macro avg       0.83      0.83      0.80         5
weighted avg       0.87      0.80      0.80         5
```

Logistic Regression is a linear model for binary classification that predicts the probability of a sample belonging to a particular class.

4. Performance Testing & Model Selection

Activity 4.1: Evaluation Metrics

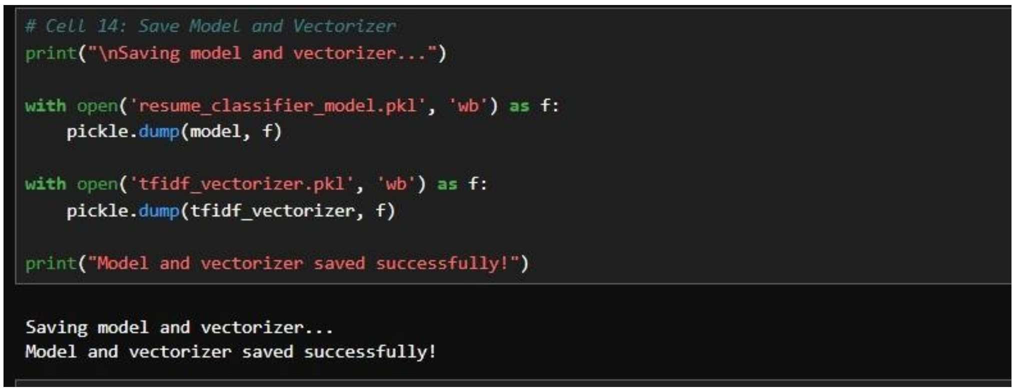
- Accuracy
- Precision
- Recall
- F1-Score
- Confusion Matrix



5. Model Deployment:

Activity 5.1: Save Model

- Save trained model using pickle
- Save TF-IDF vectorizer

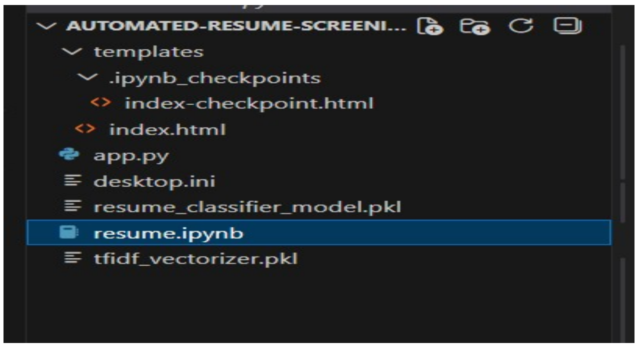


```python
# Cell 14: Save Model and Vectorizer
print("\nSaving model and vectorizer...")

with open('resume_classifier_model.pkl', 'wb') as f:
    pickle.dump(model, f)

with open('tfidf_vectorizer.pkl', 'wb') as f:
    pickle.dump(tfidf_vectorizer, f)

print("Model and vectorizer saved successfully!")
```

```
Saving model and vectorizer...
Model and vectorizer saved successfully!
```

Activity 5.2: Flask Web Application Development

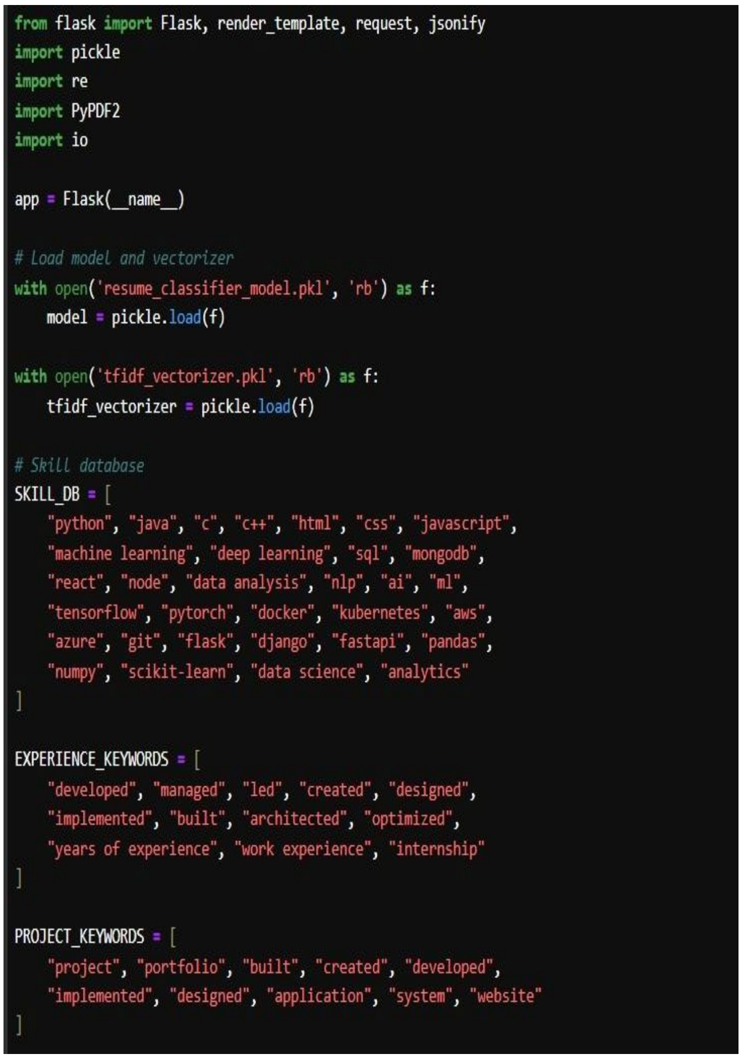
Application Architecture



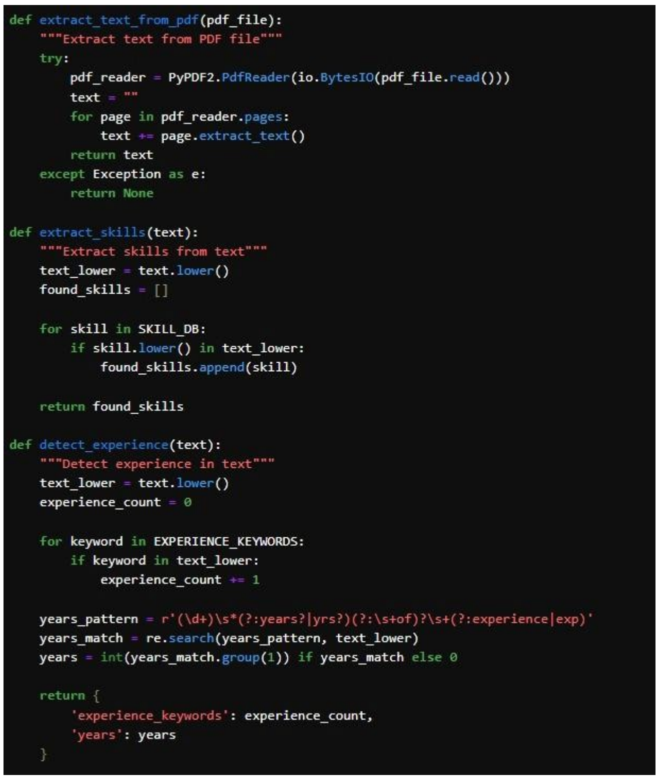
```
AUTOMATED-RESUME-SCREENI...
  templates
    .ipynb_checkpoints
      index-checkpoint.html
    index.html
  app.py
  desktop.ini
  resume_classifier_model.pkl
  resume.ipynb
  tfidf_vectorizer.pkl
```

Backend Implementation (app.py)

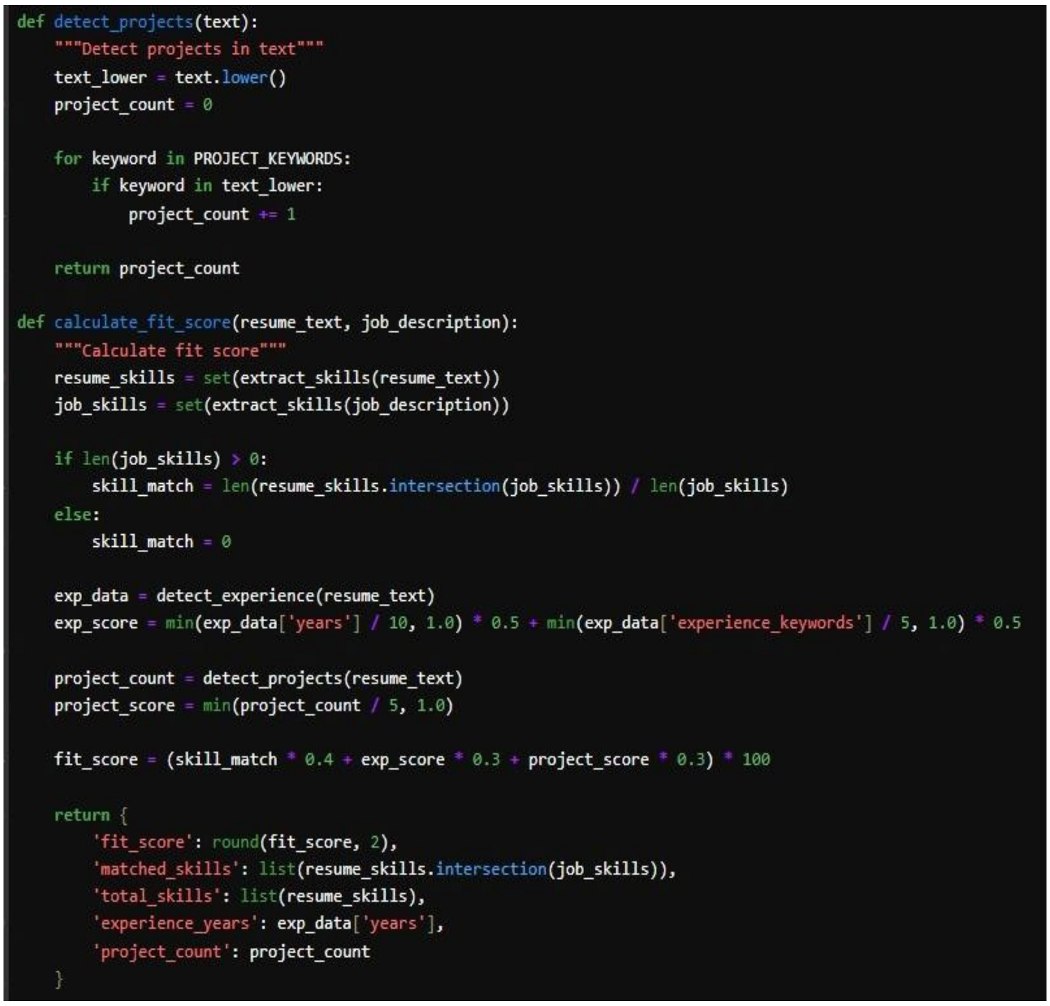
- **Model Loading:**
  Loads the pre-trained resume_classifier_model.pkl and tfidf_vectorizer.pkl during application startup, ensuring fast, real-time predictions for every uploaded resume.
- **File Handling:**
  Accepts multiple resume files via POST request and supports both PDF and TXT formats. Each file is read, and its text is extracted using PyPDF2 or UTF-8 decoding.
- **Text Extraction:**
  Implements extract_text_from_pdf() to extract clean text from PDFs. If a file is unreadable or returns empty text, it is safely skipped to avoid processing errors.
- **Skill, Experience & Project Detection:**
  Uses predefined keyword databases (SKILL_DB, EXPERIENCE_KEYWORDS, PROJECT_KEYWORDS) to automatically identify skills, experience terms, years of experience, and project indicators from the extracted text.
- **Preprocessing & Vectorization:**
  Converts the extracted resume text into numerical form using the saved TF-IDF vectorizer, ensuring consistent processing with the model's training pipeline.
- **Prediction:**
  Passes the TF-IDF vector into the loaded ML model to predict the resume category/domain (e.g., Data Science, Web Development). The backend also retrieves the probability scores for confidence calculation.
- **Fit Score Calculation:**
  Combines skill match percentage, project count, and experience signals to compute a composite Fit Score (0–100). This helps rank resumes with respect to the job description provided.



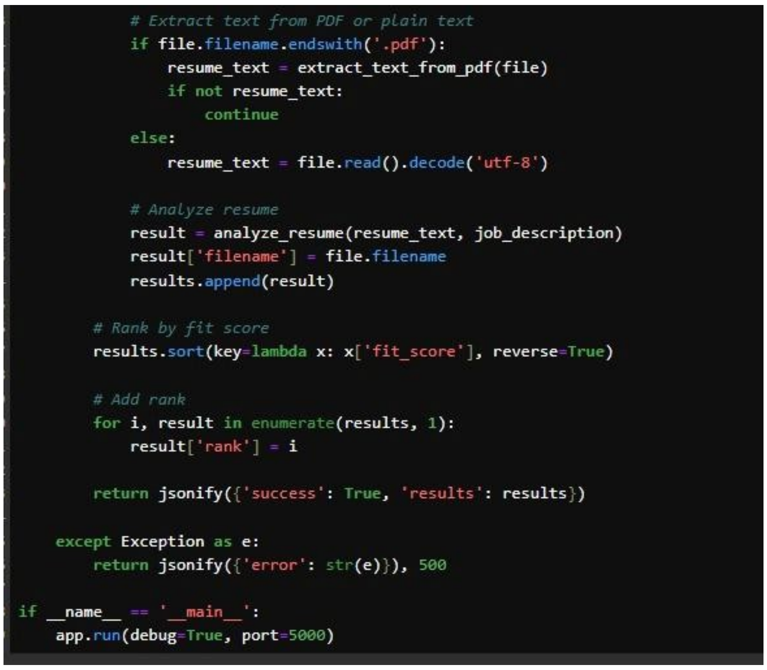
```python
from flask import Flask, render_template, request, jsonify
import pickle
import re
import PyPDF2
import io

app = Flask(__name__)

# Load model and vectorizer
with open('resume_classifier_model.pkl', 'rb') as f:
    model = pickle.load(f)

with open('tfidf_vectorizer.pkl', 'rb') as f:
    tfidf_vectorizer = pickle.load(f)

# Skill database
SKILL_DB = [
    "python", "java", "c", "c++", "html", "css", "javascript",
    "machine learning", "deep learning", "sql", "mongodb",
    "react", "node", "data analysis", "nlp", "ai", "ml",
    "tensorflow", "pytorch", "docker", "kubernetes", "aws",
    "azure", "git", "flask", "django", "fastapi", "pandas",
    "numpy", "scikit-learn", "data science", "analytics"
]

EXPERIENCE_KEYWORDS = [
    "developed", "managed", "led", "created", "designed",
    "implemented", "built", "architected", "optimized",
    "years of experience", "work experience", "internship"
]

PROJECT_KEYWORDS = [
    "project", "portfolio", "built", "created", "developed",
    "implemented", "designed", "application", "system", "website"
]
```



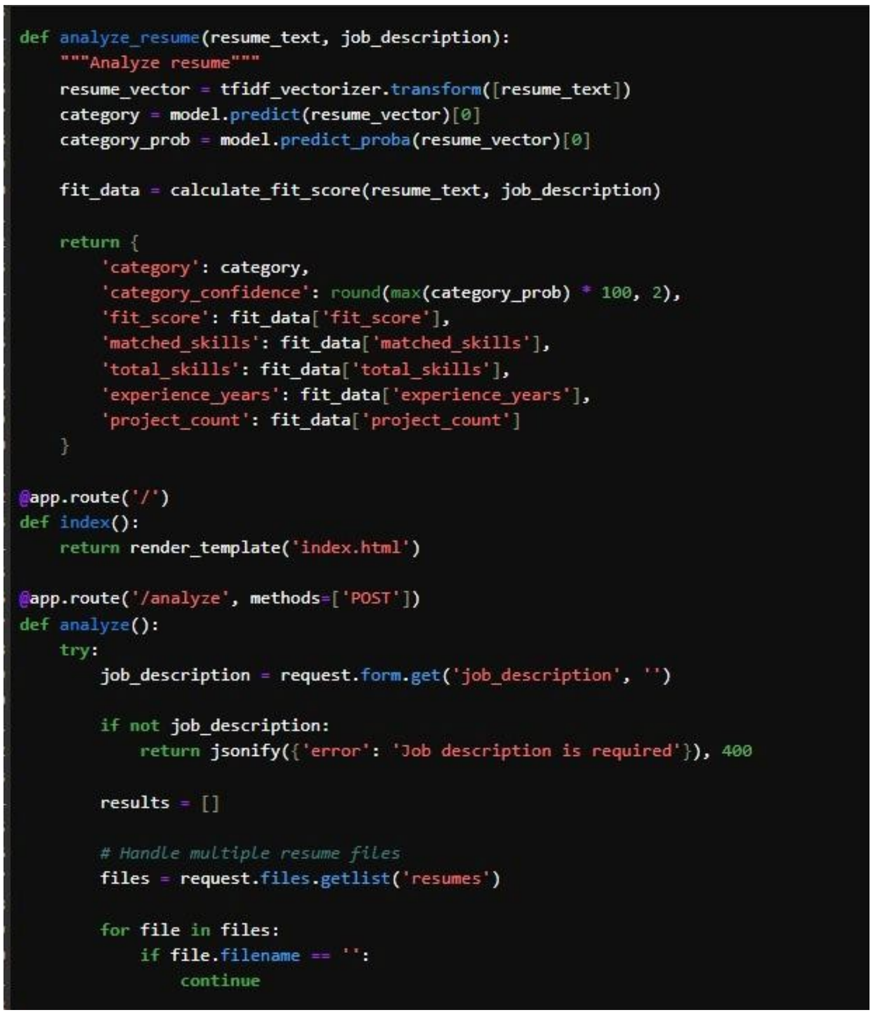
```python
def extract_text_from_pdf(pdf_file):
    """Extract text from PDF file"""
    try:
        pdf_reader = PyPDF2.PdfReader(io.BytesIO(pdf_file.read()))
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text()
        return text
    except Exception as e:
        return None

def extract_skills(text):
    """Extract skills from text"""
    text_lower = text.lower()
    found_skills = []

    for skill in SKILL_DB:
        if skill.lower() in text_lower:
            found_skills.append(skill)

    return found_skills

def detect_experience(text):
    """Detect experience in text"""
    text_lower = text.lower()
    experience_count = 0

    for keyword in EXPERIENCE_KEYWORDS:
        if keyword in text_lower:
            experience_count += 1

    years_pattern = r'(\d+)\s*(?:years?|yrs?)(?:\s+of)?\s+(?:experience|exp)'
    years_match = re.search(years_pattern, text_lower)
    years = int(years_match.group(1)) if years_match else 0

    return {
        'experience_keywords': experience_count,
        'years': years
    }
```



```python
def detect_projects(text):
    """Detect projects in text"""
    text_lower = text.lower()
    project_count = 0

    for keyword in PROJECT_KEYWORDS:
        if keyword in text_lower:
            project_count += 1

    return project_count

def calculate_fit_score(resume_text, job_description):
    """Calculate fit score"""
    resume_skills = set(extract_skills(resume_text))
    job_skills = set(extract_skills(job_description))

    if len(job_skills) > 0:
        skill_match = len(resume_skills.intersection(job_skills)) / len(job_skills)
    else:
        skill_match = 0

    exp_data = detect_experience(resume_text)
    exp_score = min(exp_data['years'] / 10, 1.0) * 0.5 + min(exp_data['experience_keywords'] / 5, 1.0) * 0.5

    project_count = detect_projects(resume_text)
    project_score = min(project_count / 5, 1.0)

    fit_score = (skill_match * 0.4 + exp_score * 0.3 + project_score * 0.3) * 100

    return {
        'fit_score': round(fit_score, 2),
        'matched_skills': list(resume_skills.intersection(job_skills)),
        'total_skills': list(resume_skills),
        'experience_years': exp_data['years'],
        'project_count': project_count
    }
```



```python
            # Extract text from PDF or plain text
            if file.filename.endswith('.pdf'):
                resume_text = extract_text_from_pdf(file)
                if not resume_text:
                    continue
            else:
                resume_text = file.read().decode('utf-8')

            # Analyze resume
            result = analyze_resume(resume_text, job_description)
            result['filename'] = file.filename
            results.append(result)

        # Rank by fit score
        results.sort(key=lambda x: x['fit_score'], reverse=True)

        # Add rank
        for i, result in enumerate(results, 1):
            result['rank'] = i

        return jsonify({'success': True, 'results': results})

    except Exception as e:
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```



```python
def analyze_resume(resume_text, job_description):
    """Analyze resume"""
    resume_vector = tfidf_vectorizer.transform([resume_text])
    category = model.predict(resume_vector)[0]
    category_prob = model.predict_proba(resume_vector)[0]

    fit_data = calculate_fit_score(resume_text, job_description)

    return {
        'category': category,
        'category_confidence': round(max(category_prob) * 100, 2),
        'fit_score': fit_data['fit_score'],
        'matched_skills': fit_data['matched_skills'],
        'total_skills': fit_data['total_skills'],
        'experience_years': fit_data['experience_years'],
        'project_count': fit_data['project_count']
    }

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/analyze', methods=['POST'])
def analyze():
    try:
        job_description = request.form.get('job_description', '')

        if not job_description:
            return jsonify({'error': 'Job description is required'}), 400

        results = []

        # Handle multiple resume files
        files = request.files.getlist('resumes')

        for file in files:
            if file.filename == '':
                continue
```

Frontend Implementation (HTML Templates)

Key Features of the Web Application:

1. User Interface Components:

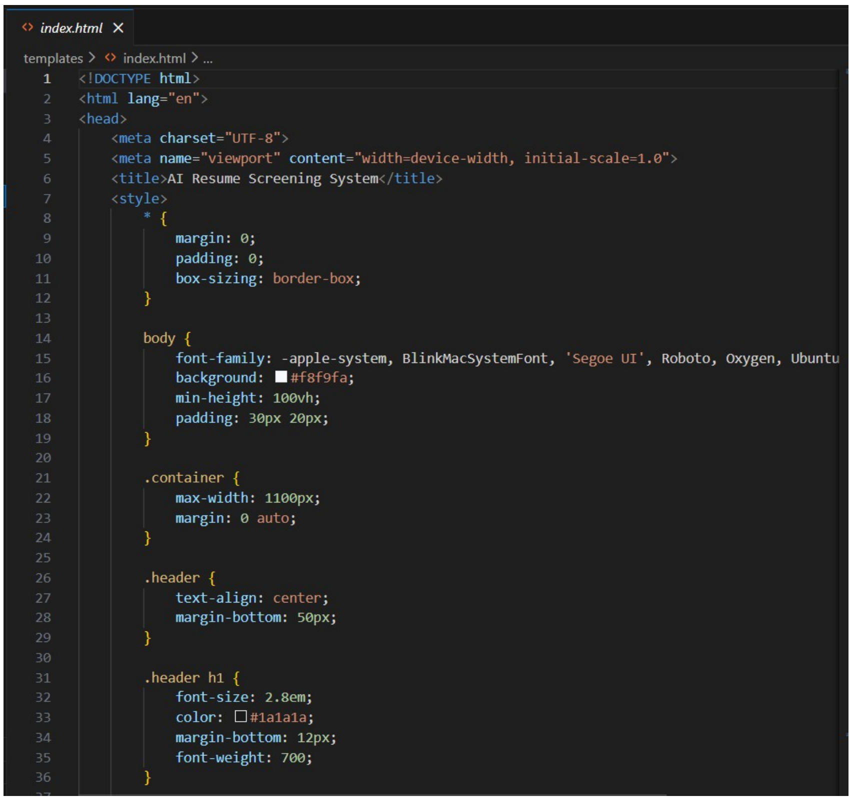
■ Clean and professional ATS-style header for a recruitment-focused interface.
■ Simple and intuitive resume upload section (PDF/DOCX).
■ Drag-and-drop upload box for user convenience.
■ Display area showing extracted text/keywords from the uploaded resume.
■ Responsive HTML + CSS layout for seamless desktop and mobile use.
■ Progress indicators for file upload and prediction processing.

2. Intelligent Screening & Decision Support:

■ AI-powered resume analysis: Predicts the job category or suitability.
■ Job Match Score: Displays percentage match (e.g., “78% Fit for Data Science”).
■ Highlights key extracted skills and missing skills for transparency.
■ Shows prediction confidence generated by the ML model.
■ Provides category classification such as *“Suitable / Not Suitable”*.

3. Safety & Validation Features:

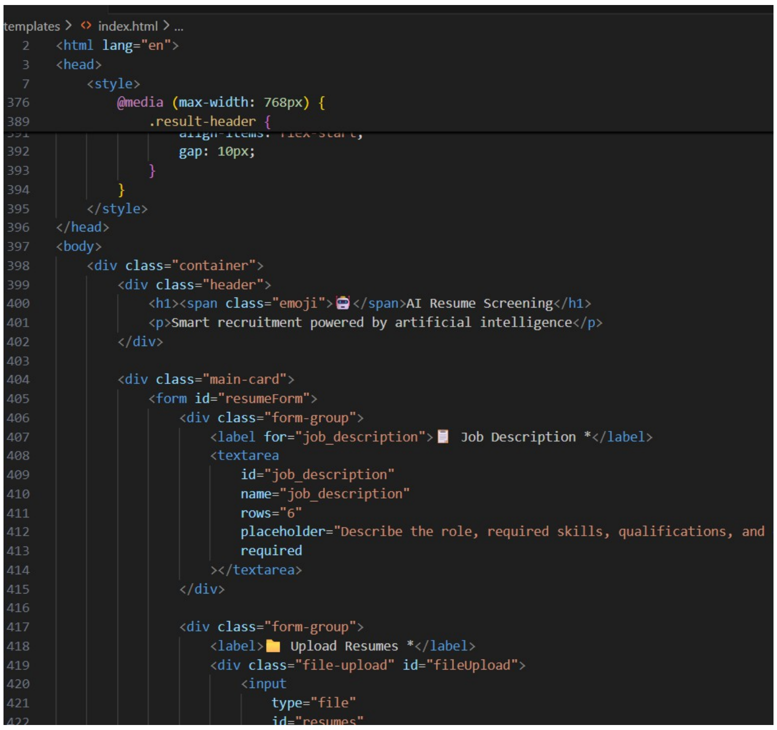
- File format validation (only PDF/DOCX/TXT allowed).
- File size validation to prevent large/invalid uploads.
- Error handling for unreadable or empty resumes.
- Secure communication with Flask backend for text extraction & prediction.
- Sanitized inputs to avoid injection or malicious file attacks.



index.html ×

templates > index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AI Resume Screening System</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen, Ubuntu
            background: #f8f9fa;
            min-height: 100vh;
            padding: 30px 20px;
        }

        .container {
            max-width: 1100px;
            margin: 0 auto;
        }

        .header {
            text-align: center;
            margin-bottom: 50px;
        }

        .header h1 {
            font-size: 2.8em;
            color: #1a1a1a;
            margin-bottom: 12px;
            font-weight: 700;
        }
```



templates > index.html > ...

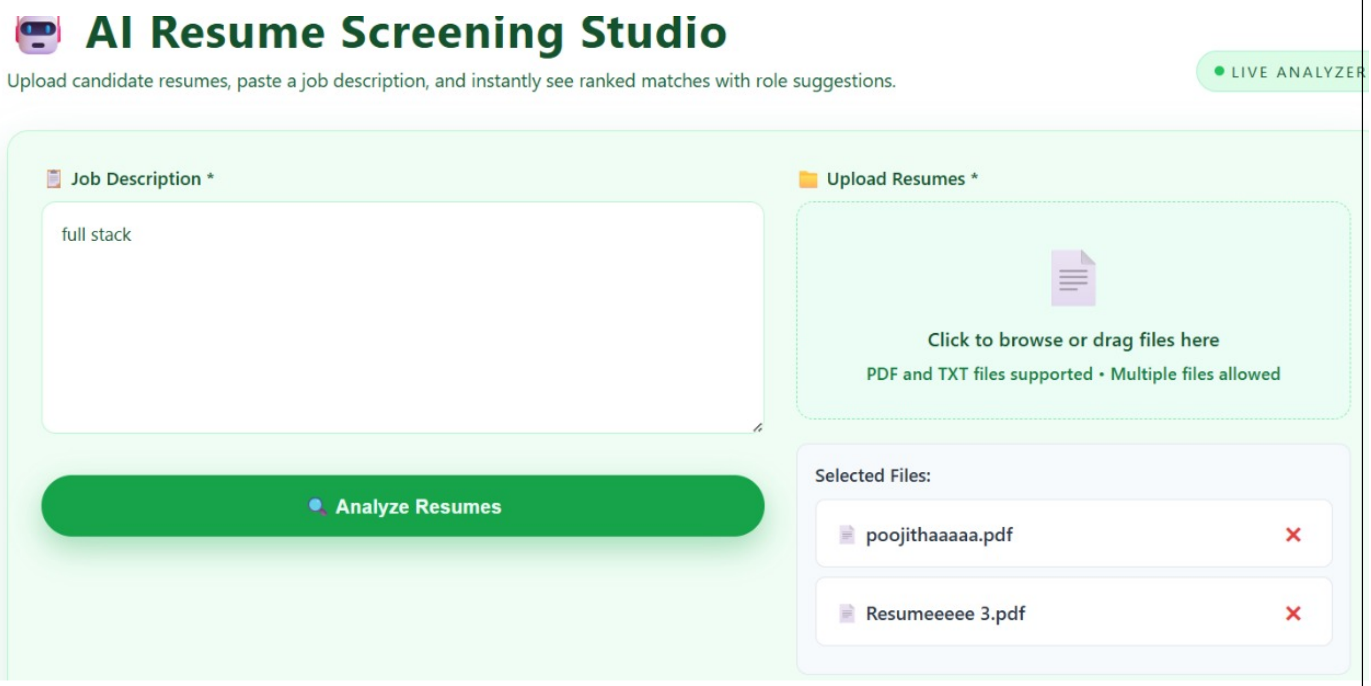
```html
<html lang="en">
<head>
    <style>
        @media (max-width: 768px) {
            .result-header {
                gap: 10px;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1><span class="emoji">🤖</span>AI Resume Screening</h1>
            <p>Smart recruitment powered by artificial intelligence</p>
        </div>

        <div class="main-card">
            <form id="resumeForm">
                <div class="form-group">
                    <label for="job_description">📋 Job Description *</label>
                    <textarea
                        id="job_description"
                        name="job_description"
                        rows="6"
                        placeholder="Describe the role, required skills, qualifications, and
                        required
                    ></textarea>
                </div>

                <div class="form-group">
                    <label>📁 Upload Resumes *</label>
                    <div class="file-upload" id="fileUpload">
                        <input
                            type="file"
                            id="resumes"
```

Application Screenshots and Workflow

The workflow moves sequentially from data input to clinical risk assessment.

1. Home Page Interface (index.html)

○ Features: Clean, professional interface with clear headings and branding. All necessary inputs are available on a single screen for efficient data entry.



AI Resume Screening Studio

Upload candidate resumes, paste a job description, and instantly see ranked matches with role suggestions.

● LIVE ANALYZER
Job Description *
Upload Resumes *
Click to browse or drag files here
PDF and TXT files supported • Multiple files allowed
Analyze Resumes
Selected Files:

full stack

poojithaaaaa.pdf

Resumeeeee 3.pdf

2. Resume Upload Form

Form Components:

- File Upload Input: Allows users to upload resumes in PDF, DOCX, or TXT format.
- Drag-and-drop upload option for improved accessibility.
- Instruction text to guide users on accepted formats and size limits.
- Upload button triggering the server-side parsing workflow.

Focus:

The form is intentionally kept minimal, containing only what the ML model requires—Resume File → Text Extraction → NLP → Classification.

This ensures simplicity, faster processing, and better data integrity.

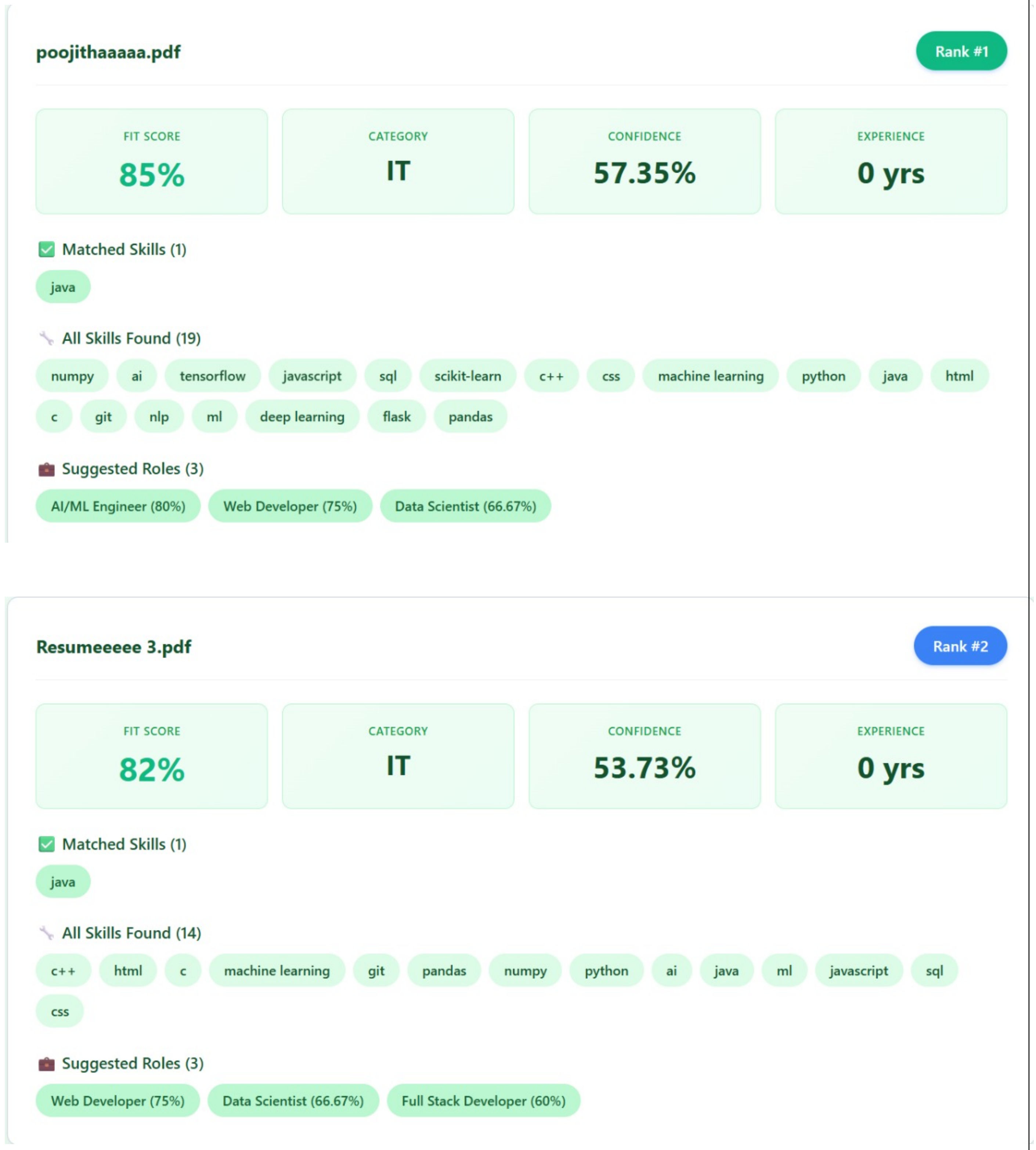
2. Screening Results Display

Results Include:

- Category Classification (e.g., “Predicted Role: Data Science / Web Developer / HR”).
- Suitability Status (e.g., “Suitable” or “Not Suitable”).
- Job Match Score percentage
  → e.g., “Match Score: 82.47%”.
- Confidence Level of prediction (e.g., “Confidence Score: 93.12%”).
- Extracted Skills Summary highlighting key skills detected in the resume.
- Missing Skills Section (optional future enhancement) indicating what skills the andidate lacks.

Color-coded Results Box:

- Green box → Suitable / High Match Score
- Yellow box → Moderate Suitability
- Red box → Not Suitable / Low Match Score



poojithaaaaa.pdf Rank #1

FIT SCORE	CATEGORY	CONFIDENCE	EXPERIENCE
85%	IT	57.35%	0 yrs

✓ Matched Skills (1)

java

All Skills Found (19)

numpy · ai · tensorflow · javascript · sql · scikit-learn · c++ · css · machine learning · python · java · html · c · git · nlp · ml · deep learning · flask · pandas

Suggested Roles (3)

AI/ML Engineer (80%) · Web Developer (75%) · Data Scientist (66.67%)

Resumeeeee 3.pdf Rank #2

FIT SCORE	CATEGORY	CONFIDENCE	EXPERIENCE
82%	IT	53.73%	0 yrs

✓ Matched Skills (1)

java

All Skills Found (14)

c++ · html · c · machine learning · git · pandas · numpy · python · ai · java · ml · javascript · sql · css

Suggested Roles (3)

Web Developer (75%) · Data Scientist (66.67%) · Full Stack Developer (60%)

Future Enhancements

Future plans for the Automated Resume Screening System focus on improved deployment, enhanced HR integration, stronger model performance, and real-world hiring support:

Integration with ATS and HRMS Platforms

A major future extension of this system is to create REST API endpoints that can seamlessly integrate with Applicant Tracking Systems (ATS) and Human Resource Management Systems (HRMS).
This integration will enable automatic classification and evaluation the moment a resume gets uploaded into the HR portal. As a result, recruiters will receive immediate job-fit scores, extracted skills, and candidate suitability insights directly inside their existing dashboards without manually running the tool separately. This would transform the model into a background automation service embedded inside recruitment workflows.

Ensemble Based Learning

Another promising enhancement is to extend the current logistic regression model into a more robust ensemble framework. Techniques such as Voting Classifiers, Bagging, and Stacking can combine multiple learning algorithms like SVM, Random Forest, Naive Bayes, and Logistic Regression. Using ensemble modelling helps improve generalization and accuracy especially when the resume dataset becomes more diverse in terms of job domains and writing styles. This multi-model approach ensures better prediction stability and less sensitivity to classification noise.

End-to-End Recruiter Portal

Build a full recruiter portal with:

- Bulk resume upload
- Candidate ranking list

- Job match score comparison
- Filter & search by skills, experience, category
- Export shortlisted candidates

  This would transform the model into a complete recruitment assistance tool.

Automatic Skill Gap Analysis

Add advanced features that highlight:

- Missing skills compared to job description
- Suggested courses for skill improvement
- Resume optimization tips

  This improves transparency for job seekers and supports professional development.

Continuous Model Improvement

Expand the dataset to include:

- More job categories (Cloud, Cybersecurity, DevOps, Product Management)
- Resumes from different countries and formats
- Diverse experience levels (fresher, mid-level, senior)

  This enhances generalization, reduces bias, and improves performance across industries.

OCR for Scanned or Image-Based Resume Files

At present, the system works best with readable PDF or text formats. In real hiring situations, many resumes are scanned images or picture-based PDFs which cannot be processed directly. By integrating Optical Character Recognition (OCR) technology, the system will be able to convert scanned images into digital text before applying NLP. This makes the solution more universal and capable of processing any type of resume format.

Mobile App for Job Seekers

Develop a mobile app where candidates can:

- Upload their resume
- Get a Job Match Score
- View skills extracted and missing skills
- Receive job recommendations

  This makes the system accessible to a wider audience.

Conclusion

This project successfully demonstrates resume classification and evaluation using Machine Learning and NLP. It automatically extracts and analyzes resume content, assigns suitability scores, and categorizes resumes through a deployed Flask application.

The system minimizes manual effort, offers unbiased evaluation, and accelerates hiring workflows by supporting instant resume processing and filtering. It provides a foundation for fully automated and intelligent hiring systems based on artificial intelligence.