Automated Resume Screening System

Project Description:

Automated Resume Screening System is an AI-powered machine learning solution designed to automatically analyses, scores, classifies, and filters resumes based on job requirements.

The system extract features such as skills, experience, education, and keywords from resumes, and then uses supervised learning algorithms to classify candidates as Suitable or Not Suitable for a job role.

This project aims to reduce manual HR effort, speed up recruitment, and improve accuracy by applying Natural Language Processing (NLP) and classification techniques.

Project Scenarios

Scenario 1: HR Recruitment Filtering

A company receives hundreds of resumes for a job role. Instead of manually checking each resume, the Automated Resume Screening System parses all resumes, extracts important features (skills, experience, degree, projects), and instantly classifies each candidate as Shortlisted or Rejected.

Scenario 2: Campus Placements

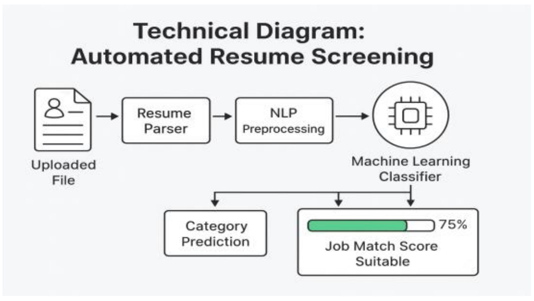
During mass placement drives, thousands of students submit their resumes.

The system quickly processes and categorizes them according to CGPA, skills, internships, certifications, and job role match score. The placement team can instantly shortlist top students.

Scenario 3: Job Portal Intelligent Matching

A job portal integrates the automated resume screening model.When a candidate uploads their resume, the system analyzes skills and experience and provides a Job Fit Score, recommending suitable job openings.

Technical Diagram:



Prerequisites:

Software Requirements:

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

Required Python Packages:

- numpy
- pandas
- scikit-learn
- nltk / spacy
- matplotlib
- seaborn
- pickle-mixin
- Flask (for deployment)

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 Flow:

- 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 Flow:

- 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 Collection & Preparation:

- Loaded manual resume dataset in Jupyter Notebook
- Removed duplicates and cleaned raw text
- Performed tokenization
- Applied lemmatization for normalizing words
- Removed stopwords and noise
- Converted the processed text into structured features (TF-IDF / BOW)

2. Exploratory Data Analysis:

- Generate descriptive statistics of resume text
- Visual analysis
- Perform univariate analysis (resume length, keyword frequency)
- Perform bivariate analysis (skills vs job category)
- Create word clouds for each job role

3. Model Building:

- Convert text into numerical vectors using TF-IDF
- Build models such as Logistic Regression
- Optimize hyperparameters for better results
- Validate model on training and test data

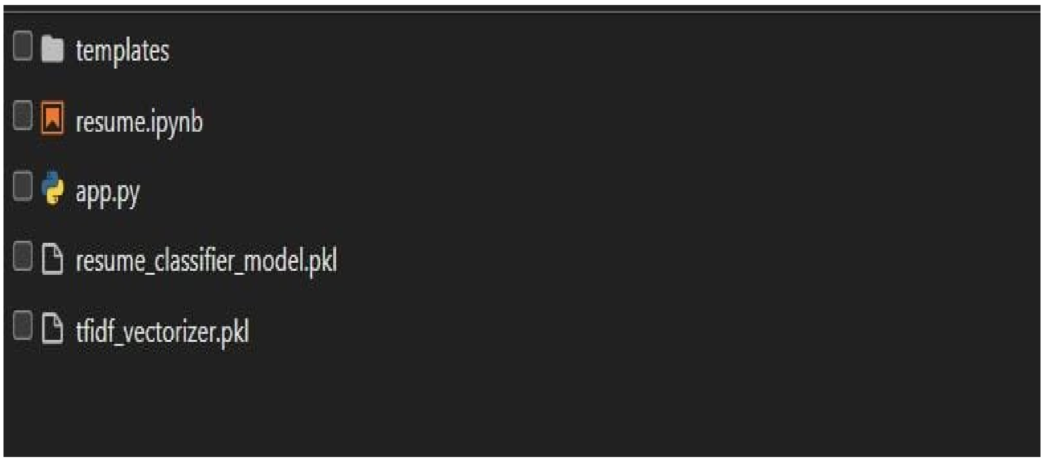
4. Performance Testing & Model Selection:

- Evaluate models using accuracy, precision, recall, F1-score
- Compare classifier performance across algorithms
- Analyze confusion matrices for each model
- Identify the best-performing model based on evaluation metrics
- Finalize the model for deployment

5. Model Deployment:

- Save the best model and TF-IDF vectorizer
- Integrate model with a web application (Flask)
- Create resume upload interface
- Implement real-time prediction (suitability & category)
- Display results with fit score and job match score

Project Structure:



```
templates
resume.ipynb
app.py
resume_classifier_model.pkl
tfidf_vectorizer.pkl
```

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:

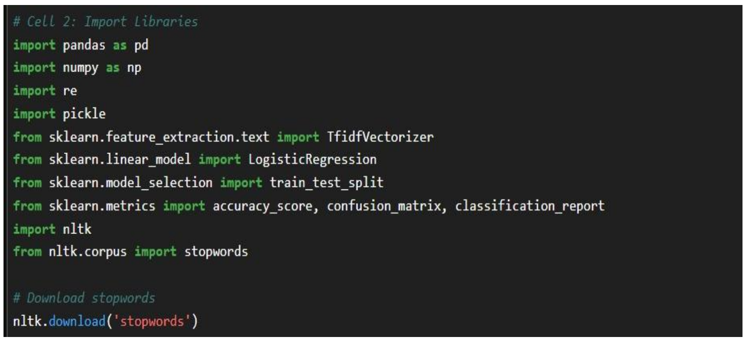
The dataset contains resumes categorized into classes (e.g., Data Science, Web Development, Android, HR, etc.).
Each resume is labeled, and text is extracted from PDF files.

Activity 1.1: Collect Dataset

The resume data was not taken from any external dataset. Instead, the resumes were collected manually. These files were then organized in the database so they could be used for further processing.

Activity 1.2: Import Libraries

Import the necessary libraries as shown below:



```
# Cell 2: Import Libraries
import pandas as pd
import numpy as np
import re
import pickle
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

# Download stopwords
nltk.download('stopwords')
```

These libraries serve the following purposes:

- **pickle** – Saves and loads Python objects (serialization/deserialization).
- **re** – Performs pattern matching and text processing using regular expressions.
- **PyPDF2** – Reads PDFs and extracts or manipulates their content.
- **io** – Handles in-memory file-like streams for reading/writing data.
- **flask** – A lightweight framework for building web applications and APIs.
- **render_template** – Renders and returns HTML templates in Flask.
- **request** – Retrieves incoming data (form, JSON, files) from client requests.
- **jsonify** – Converts Python data to JSON and sends it as a response.

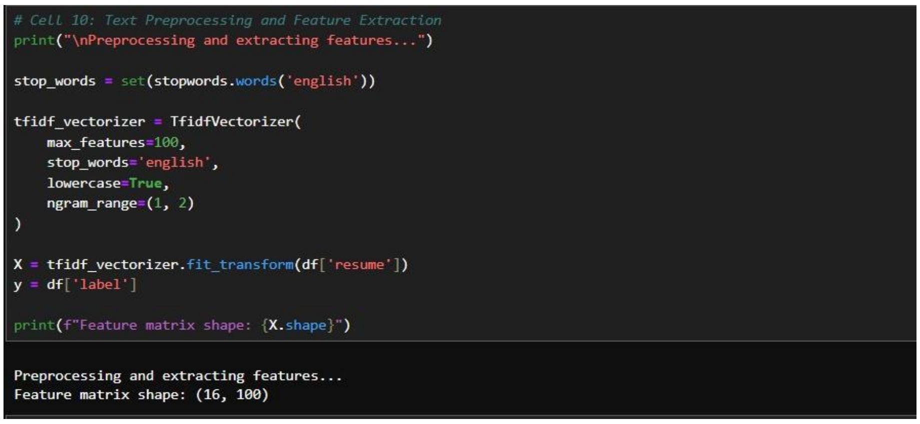
Activity 1.3: Read and Explore Data

- Load resume text
- Label encoded classes
- Check dataset size

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



```
# Cell 10: Text Preprocessing and Feature Extraction
print("\nPreprocessing and extracting features...")

stop_words = set(stopwords.words('english'))

tfidf_vectorizer = TfidfVectorizer(
    max_features=100,
    stop_words='english',
    lowercase=True,
    ngram_range=(1, 2)
)

X = tfidf_vectorizer.fit_transform(df['resume'])
y = df['label']

print(f"Feature matrix shape: {X.shape}")
```

```
Preprocessing and extracting features...
Feature matrix shape: (16, 100)
```

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

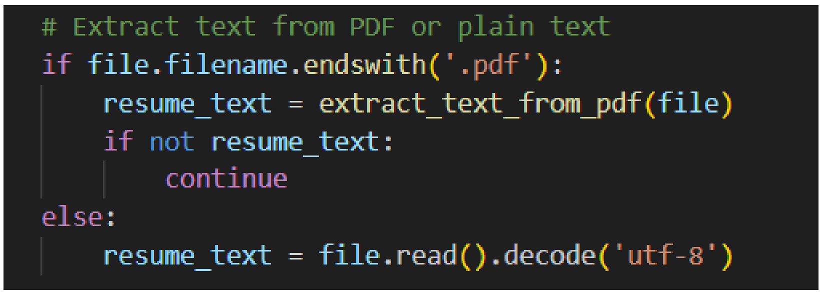
When .txt or .pdf resumes fail to decode or extract properly, no further processing is attempted, ensuring no null value reaches classification.

4. Internally normalize missing text fields

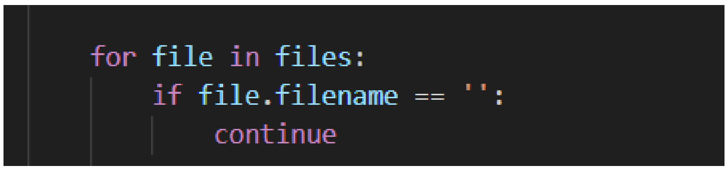
Even if parts of the resume such as skills, projects, or experience sections are absent, the keyword extractors return safe default values (e.g., empty lists or count zero), preventing null-related errors.

5. Prevent passing null text into ML model

The classification step only runs when valid, non-null text exists. Invalid entries are filtered before vectorization.



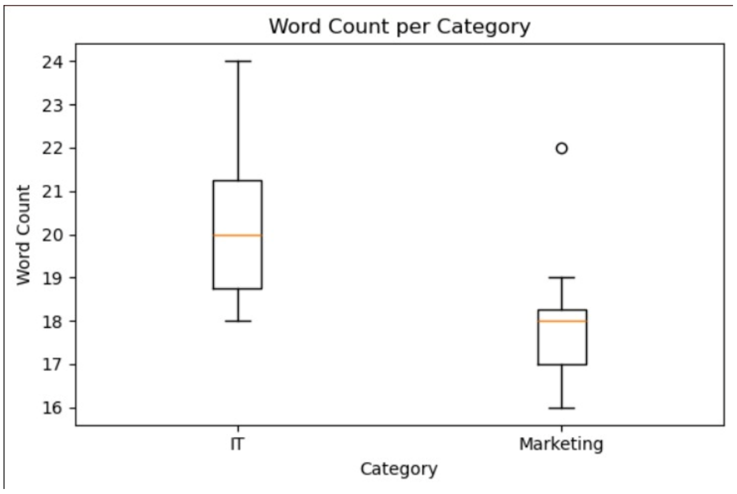
```python
# Extract text from PDF or plain text
if file.filename.endswith('.pdf'):
    resume_text = extract_text_from_pdf(file)
    if not resume_text:
        continue
else:
    resume_text = file.read().decode('utf-8')
```



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

ACTIVITY 1.6: Checking for outliers:

Create boxplots to visualize outliers in numerical features:



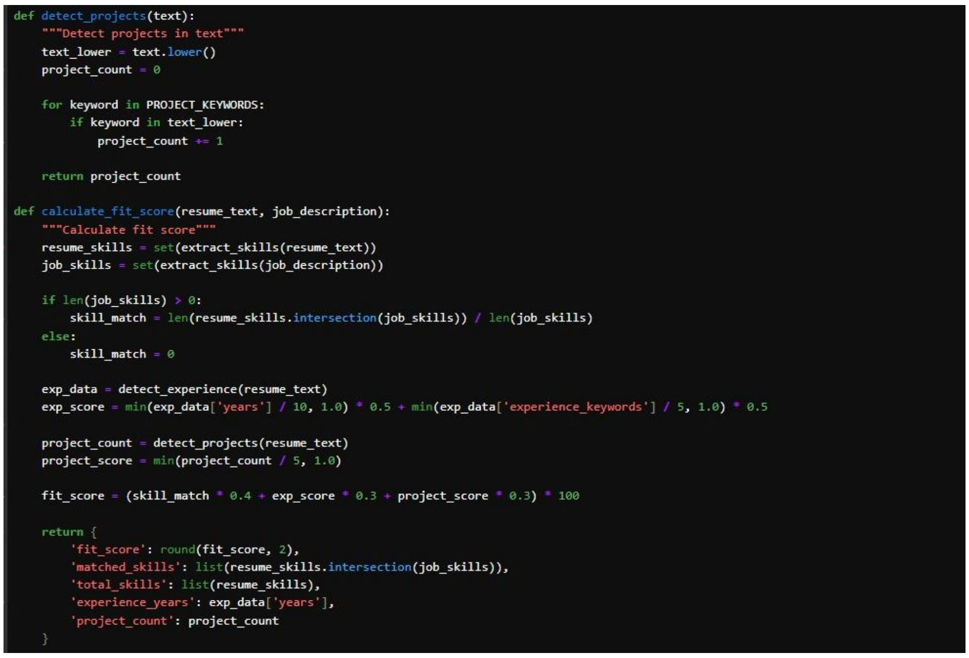
Boxplots help identify outliers, which are data points that fall outside the typical range. These can affect model performance if not handled properly.

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

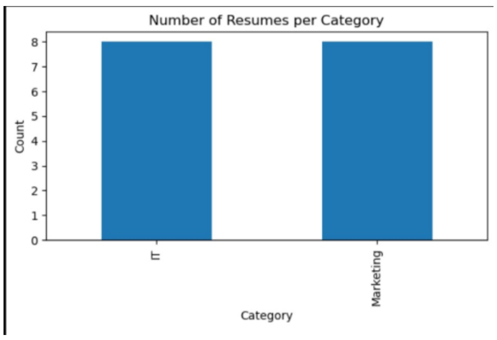
- Number of resume categories
- Count of resumes per class
- Length of text per resume



```
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
    }
```

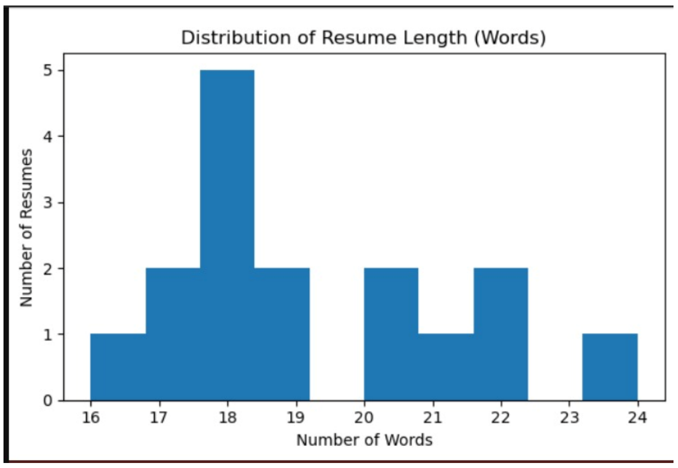
Activity 2.2: Visual Analysis

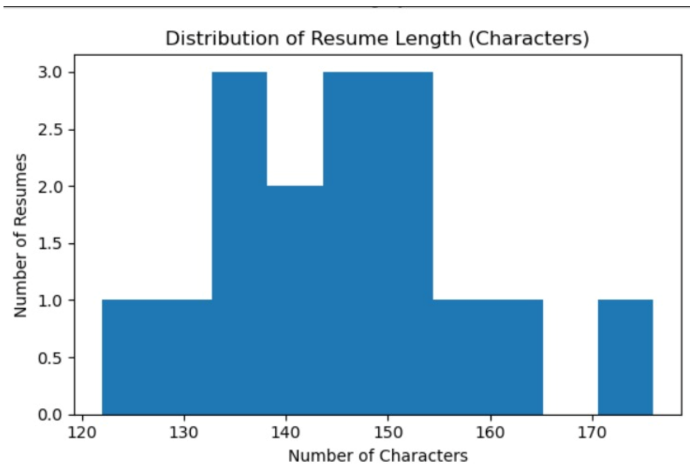
- Category distribution bar chart
- Word cloud for each resume class
- Skills frequency plot



Activity 2.3: Univariate Analysis

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





Activity 2.4: Bivariate Analysis

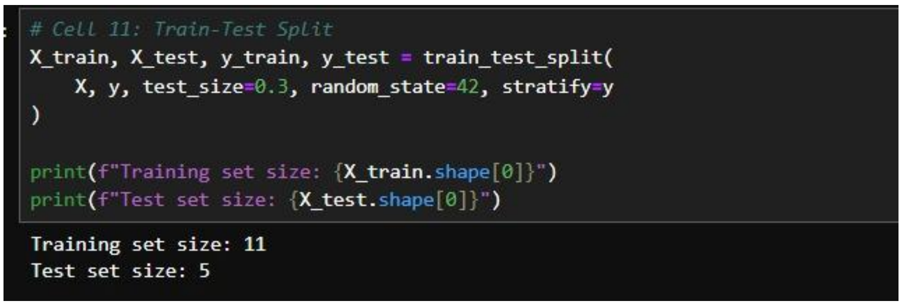
- Correlation between text length and classification
- Skill overlap between categories
-

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:



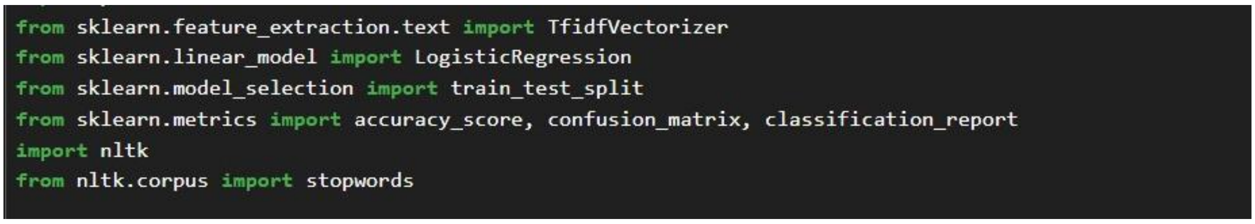
```
# Cell 11: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

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

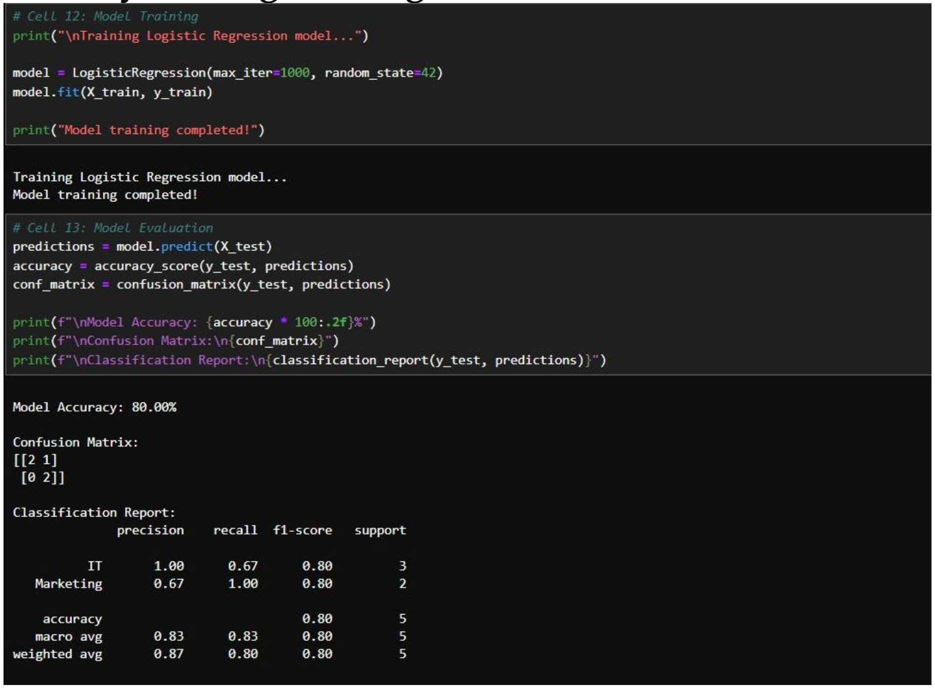
Milestone 3 : Feature Engineering & Model Building:

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)}")
```

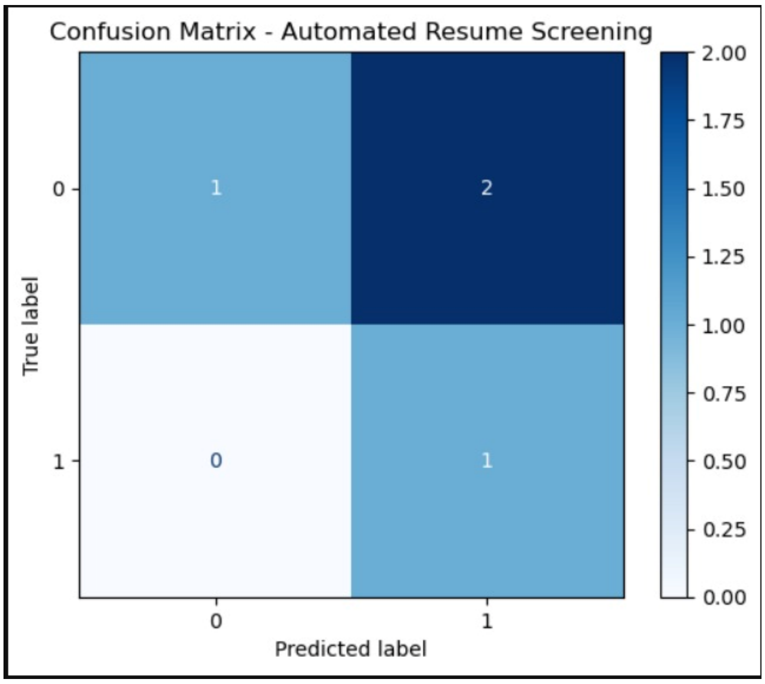
```
Model Accuracy: 80.00%

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

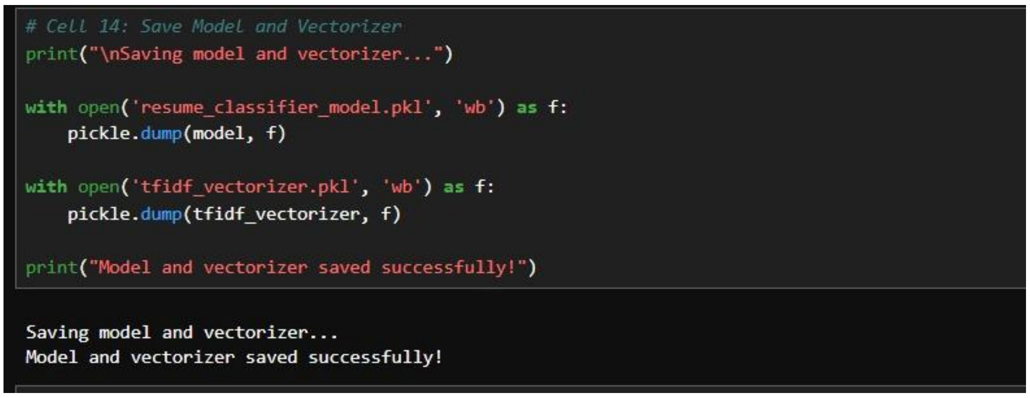
- Accuracy
- Precision
- Recall
- F1-Score
- Confusion Matrix



5. Model Deployment:

Activity 5.1: Save Model

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

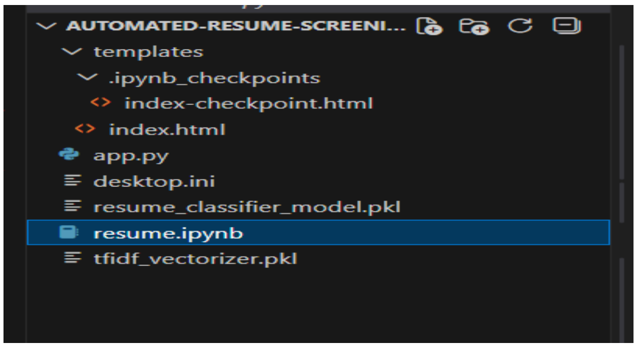


```
# Cell 14: Save Model and Vectorizer
print("\nSaving model and vectorizer...")

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

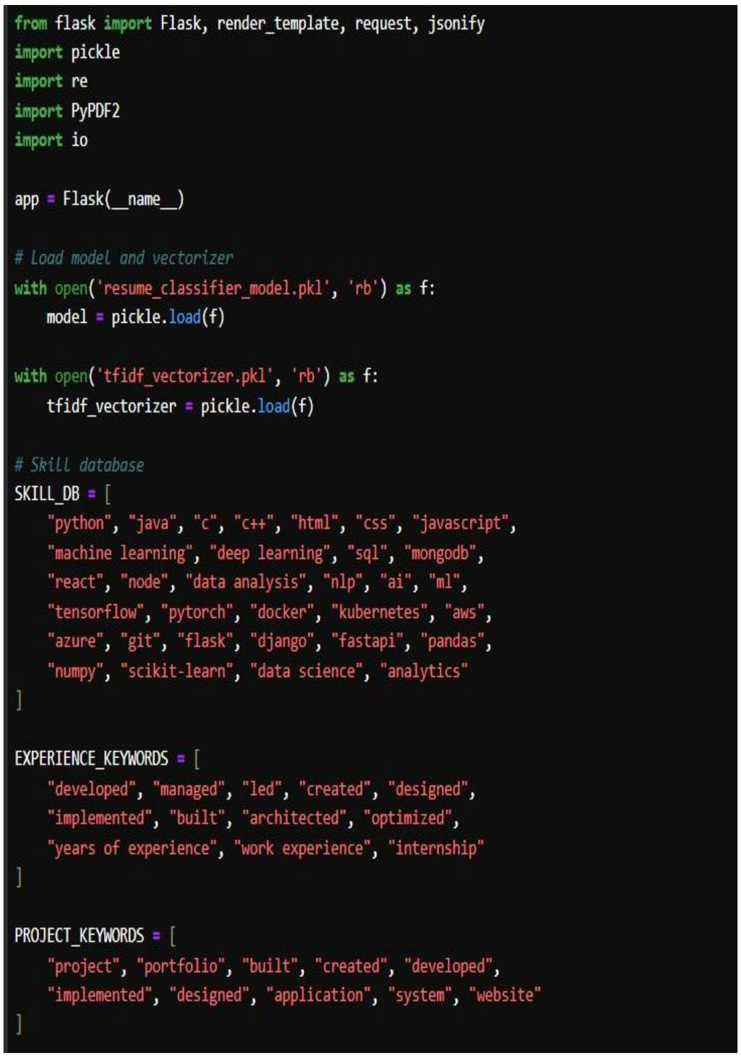
Application Architecture



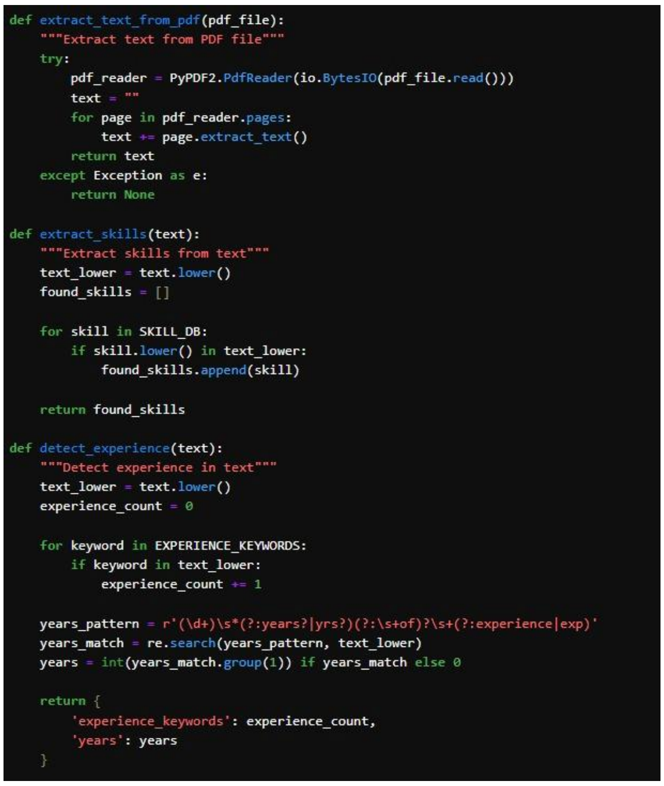
```
⌄ AUTOMATED-RESUME-SCREENI...
  ⌄ templates
    ⌄ .ipynb_checkpoints
        index-checkpoint.html
      index.html
    app.py
    desktop.ini
    resume_classifier_model.pkl
    resume.ipynb
    tfidf_vectorizer.pkl
```

Backend Implementation (app.py)

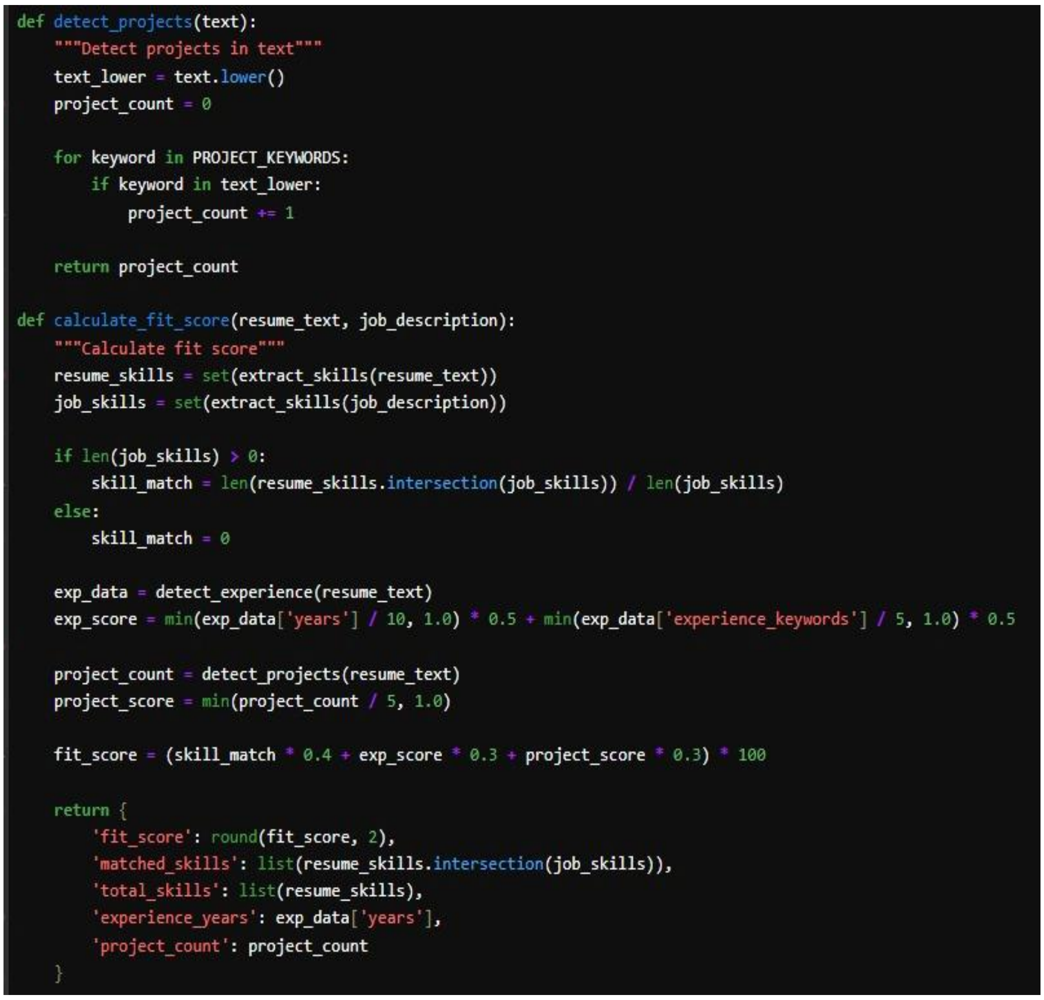
- **Model Loading:**
  Loads the pre-trained resume_classifier_model.pkl and tfidf_vectorizer.pkl during application startup, ensuring fast, real-time predictions for every uploaded resume.
- **File Handling:**
  Accepts multiple resume files via POST request and supports both PDF and TXT formats. Each 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.



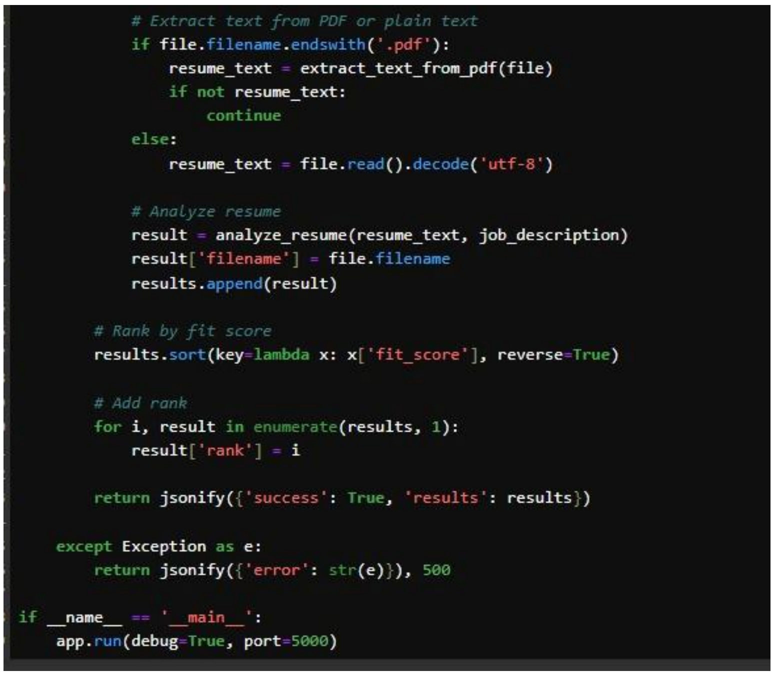
```python
from flask import Flask, render_template, request, jsonify
import pickle
import re
import PyPDF2
import io

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"
]
```



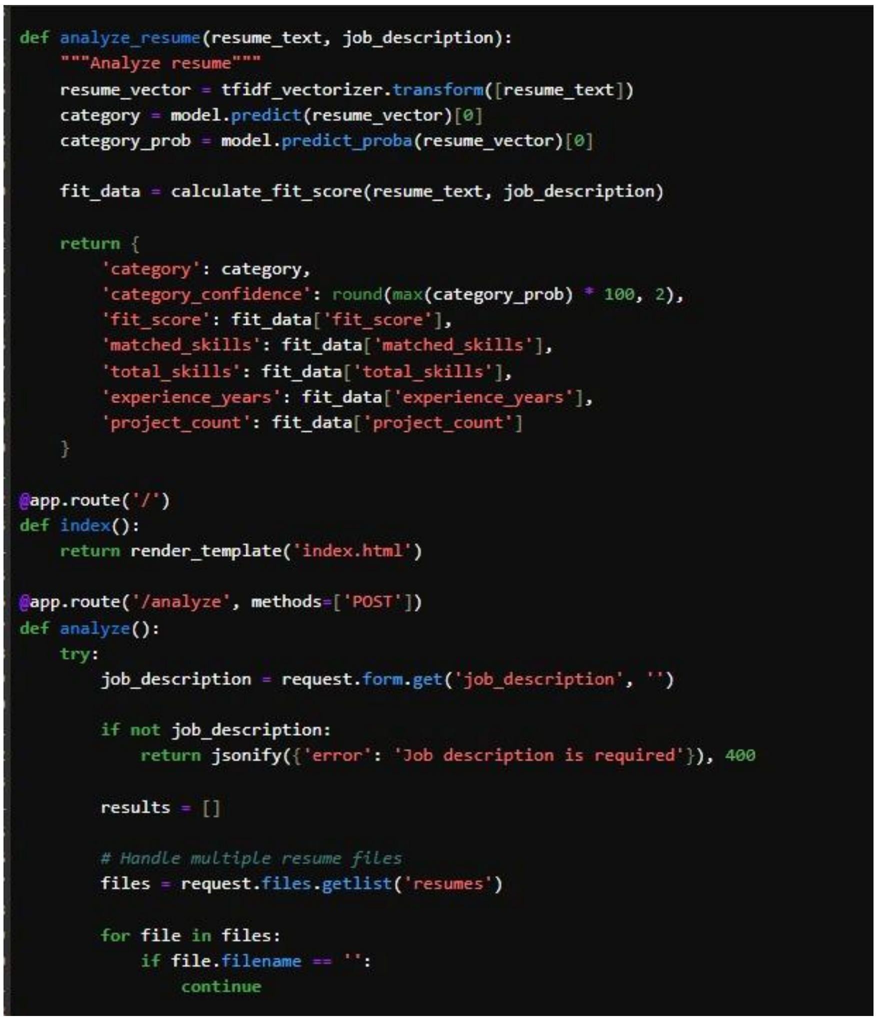
```python
def extract_text_from_pdf(pdf_file):
    """Extract text from PDF file"""
    try:
        pdf_reader = PyPDF2.PdfReader(io.BytesIO(pdf_file.read()))
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text()
        return text
    except Exception as e:
        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)

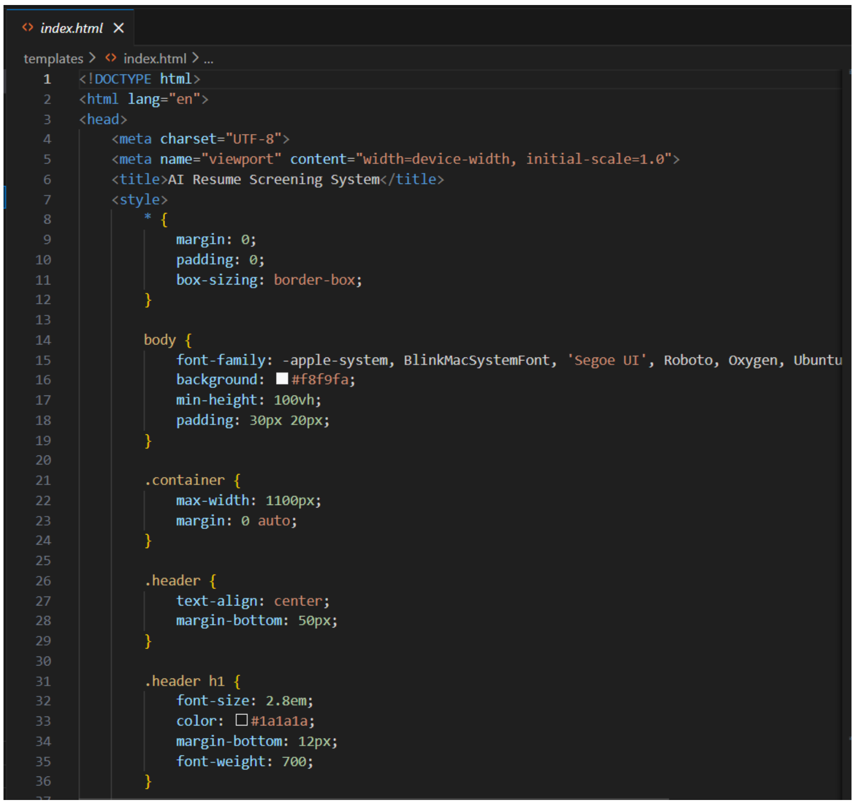
Key Features of the Web Application:

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:

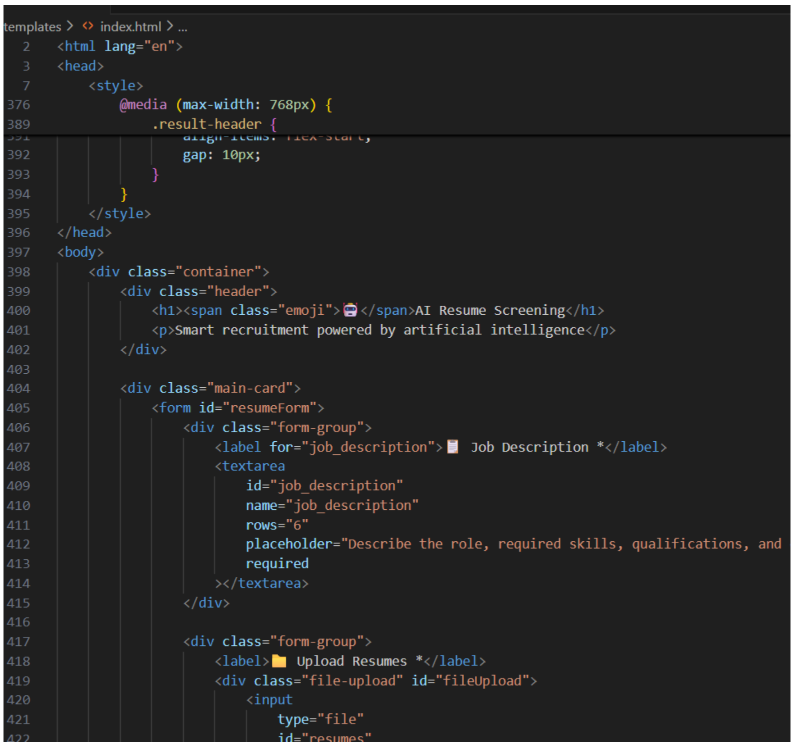
- File format validation (only PDF/DOCX/TXT allowed).
- File size validation to prevent large/invalid uploads.
- Error handling for unreadable or empty resumes.
- Secure communication with Flask backend for text extraction & prediction.
- Sanitized inputs to avoid injection or malicious file attacks.



index.html ×

templates > index.html > ...

```
<!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 > ...

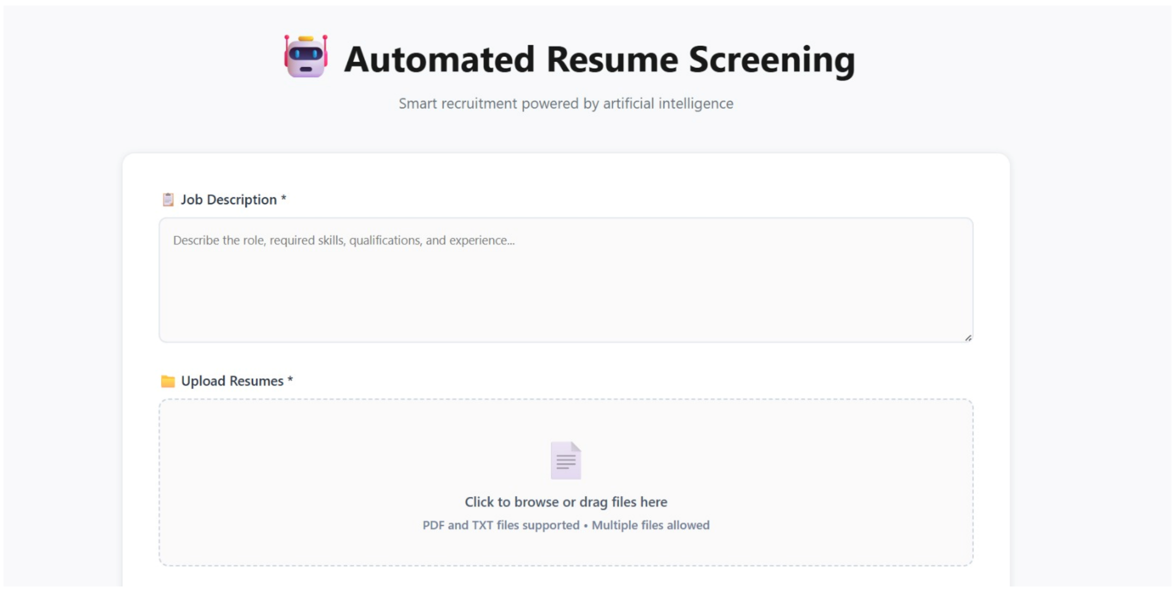
```
<html lang="en">
<head>
    <style>
        @media (max-width: 768px) {
            .result-header {
                gap: 10px;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1><span class="emoji">🤖</span>AI Resume Screening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

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.



Automated Resume Screening

Smart recruitment powered by artificial intelligence

Job Description *

Describe the role, required skills, qualifications, and experience...

Upload Resumes *

Click to browse or drag files here

PDF and TXT files supported • Multiple files allowed

2. Resume Upload Form

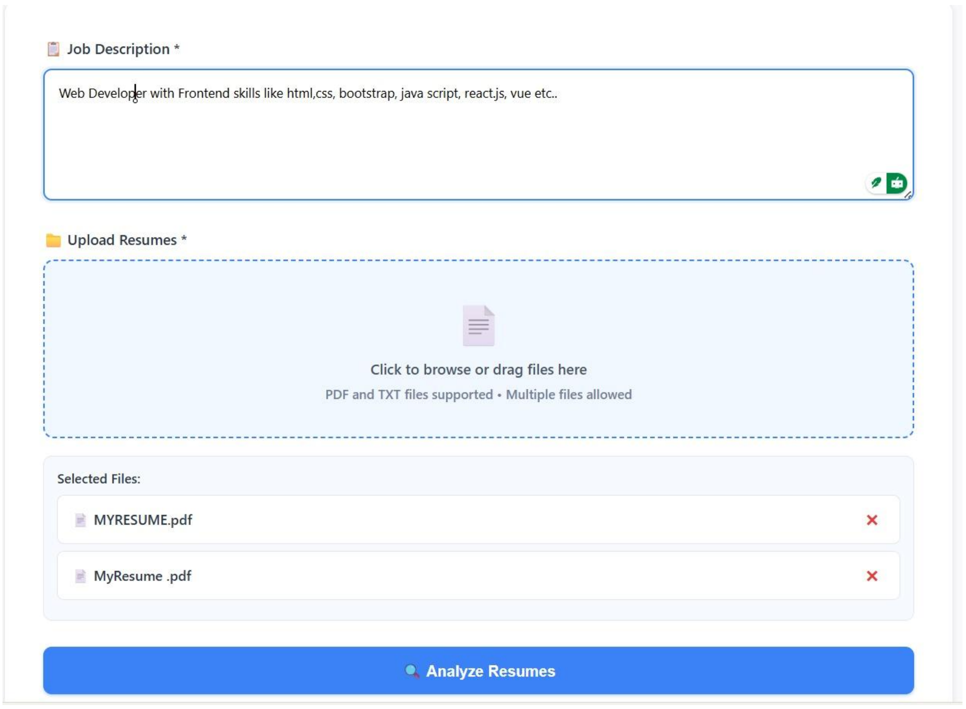
Form Components:

- 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.


📋 Job Description *

Web Developer with Frontend skills like html,css, bootstrap, java script, react.js, vue etc..

📁 Upload Resumes *

Click to browse or drag files here

PDF and TXT files supported • Multiple files allowed

Selected Files:

MYRESUME.pdf ✕

MyResume .pdf ✕

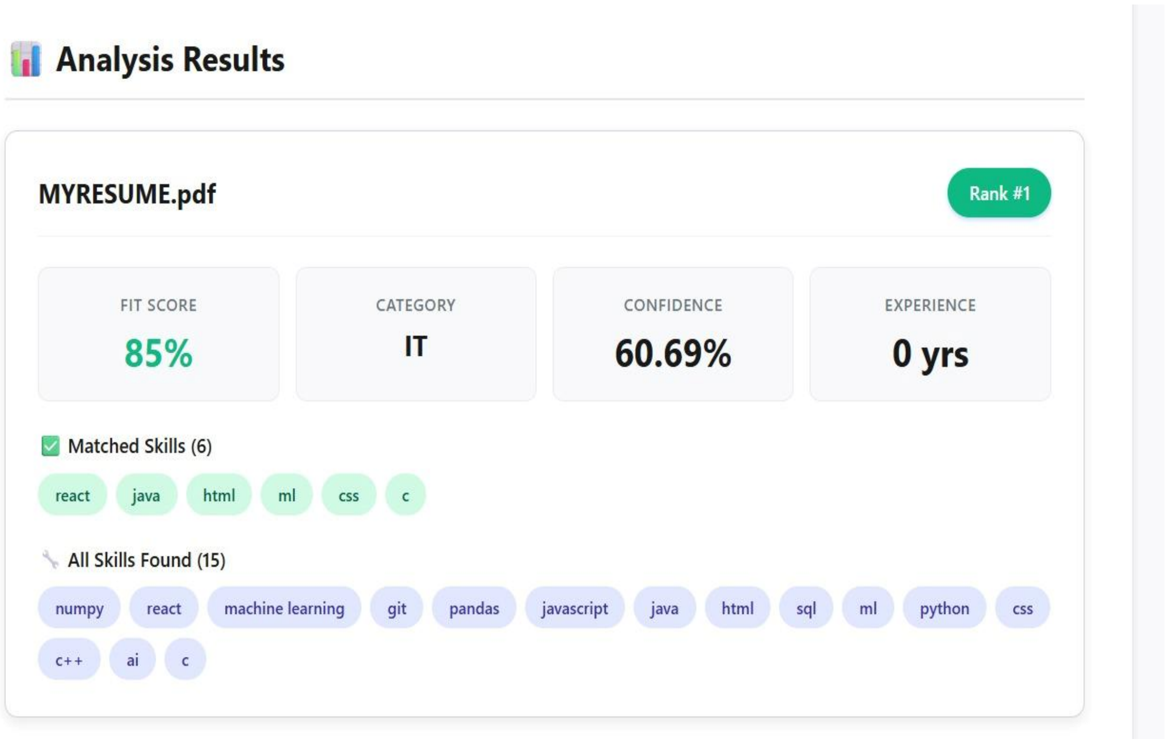
2. Screening Results Display

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



📊 Analysis Results

MYRESUME.pdf

Rank #1

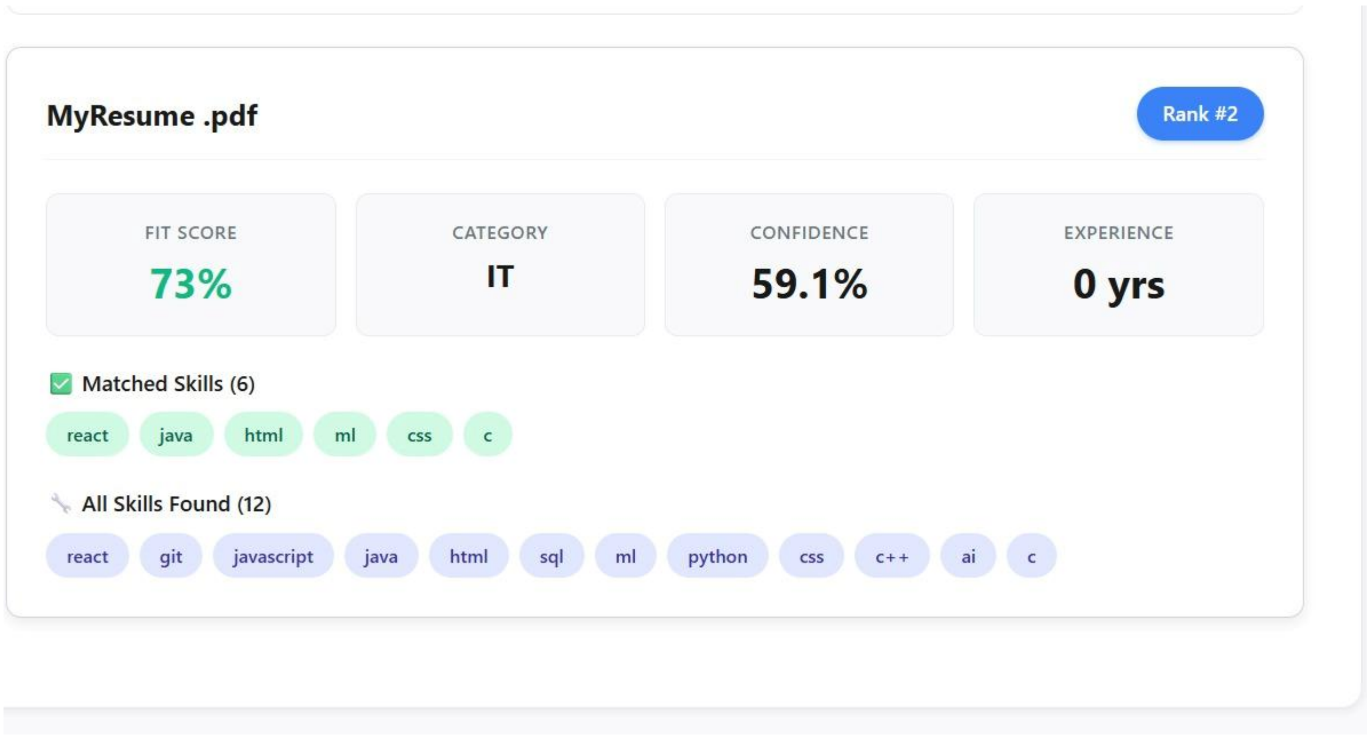
FIT SCORE	CATEGORY	CONFIDENCE	EXPERIENCE
85%	IT	60.69%	0 yrs

✅ Matched Skills (6)

react, java, html, ml, css, c

All Skills Found (15)

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



MyResume .pdf

Rank #2

FIT SCORE	CATEGORY	CONFIDENCE	EXPERIENCE
73%	IT	59.1%	0 yrs

✅ Matched Skills (6)

react, java, html, ml, css, c

All Skills Found (12)

react, git, javascript, java, html, sql, ml, python, css, c++, ai, c

Future Implementations

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

ATS & HRMS Integration

Develop API endpoints to integrate the screening model directly into Applicant Tracking Systems (ATS) and Human Resource Management Systems (HRMS).
This enables automatic resume classification, skill extraction, and candidate scoring the moment a new resume is uploaded in HR dashboards.

Multi-Model / Ensemble Deployment

Explore deploying ensemble models (e.g., Voting Classifier, Stacking Classifier) that combine SVM, Logistic Regression, and Random Forest.
This increases prediction accuracy, stability, and robustness across diverse resume formats and job categories.

Recruiter Dashboard / Web 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.

Candidate Insights & 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 Integration for Scanned Resumes

Add OCR (Optical Character Recognition) support to process scanned or image-based resumes, enabling end-to-end extraction from any document type.

Mobile Application 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

The Automated Resume Screening Project successfully developed a reliable, efficient, and scalable AI-powered recruitment support system. By leveraging Natural Language Processing (NLP) techniques and machine learning algorithms such as Logistic Regression and SVM, the model achieved accurate classification of resumes into relevant job categories and generated meaningful suitability insights.

The final product is a complete, end-to-end full-stack web application built using Flask, scikit-learn, and TF-IDF vectorization, integrating the trained model into a clean, responsive interface. It enables automated resume upload, text extraction, intelligent job-fit prediction, and skill-based analysis.

This solution provides a fast, unbiased, and consistent screening workflow, significantly reducing recruiter workload while improving the precision of candidate shortlisting. Overall, the project demonstrates strong success from data preprocessing and model development to practical deployment, establishing a foundation for modern, AI-driven hiring automation.