

RESUME ANALYZER

A PROJECT REPORT

Submitted By:

AISHWARY JAIN(221B040)

SAMARTH GUPTA (221B477)

ALOKIK SHARMA(221B050)

Under the Guidance of: Dr Dinesh Kumar Verma



MAY - 2025

Submitted in partial fulfilment for the award of the Degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Department of Computer Science & Engineering

Declaration by the Student

We hereby declare that the work reported in the B.Tech. 6th semester project entitled as “RESUME ANALYZER” in partial fulfilment for the award of degree of B. Tech., submitted at Jaypee University of Engineering and Technology, Guna, as per the best of our knowledge and belief there is no infringement of intellectual property right and copyright. In case of any violation I will solely be responsible.

AISHWARY JAIN(221B040)

SAMARTH GUPTA (221B477)

ALOKIK SHARMA(221B050)

Department of Computer Science and Engineering
Jaypee University of Engineering and Technology
Guna (M.P) India
Date: 6/5/2025



JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY

Grade 'A+' Accredited with by NAAC & Approved U/S 2(f) of the UGC Act, 1956

A.B. Road, Raghogarh, Dist: Guna (M.P.) India, Pin-473226

Phone: 07544 267051, 267310-14, Fax: 07544 267011

Website: www.juet.ac.in

CERTIFICATE

This is certify that the work titled “**Resume Analyze**” submitted by “ **Aishwary Jain, Samarth Gupta and Alokik sharma**” in partial fulfilment for the award of degree of B. Tech. of Jaypee University of Engineering and Technology, Guna has been carried out under my supervision. As per best of my knowledge and belief there is no infringement of intellectual property right and copyright also this work has not been submitted partially or wholly to any other university or institute for the award of this or any other degree or diploma. In case of any violation concern student will solely be responsible.

Signature of Supervisor

Dr. Dinesh Kumar Verma

Date: 8/05/2025

ACKNOWLEDGMENT

We would like to thank our guide **Dr. Dinesh Kumar Verma** for the encouragement and support that they have extended. We also sincerely thanks for the time spent proofreading and correcting my many mistakes. We would like to thank our parents and friends who helped us a lot in finalizing this project within the limited period. My gratitude towards the members of this project for their kind cooperation and encouragement which helped me in the completion of project **“RESUME ANALYZER”**.

Thanking You,

AISHWARY JAIN(221B040)

SAMARTH GUPTA (221B477)

ALOKIK SHARMA(221B050)

SUMMARY

Project Overview: The **Resume Analyzer** project leverages Natural Language Processing (NLP) and Python automation to evaluate and classify resumes based on extracted content. With the rise of tech recruitment and increased resume volumes, manually screening candidates is time-consuming and error-prone. This project automates resume parsing, skill extraction, and classification into relevant job categories such as Data Science, Web Development, and Machine Learning, enabling efficient shortlisting of candidates.

The system processes PDF resumes, extracts textual content using PyMuPDF, and applies NLP techniques with spaCy to identify technical skills and determine career domains. It supports data-driven recruitment workflows and enhances decision-making for HR professionals and hiring managers.

Key Features:

- ❑ **Automated Text Extraction:** Parses resumes in PDF format and extracts relevant sections like education, experience, and skills.
 - ❑ **Skill Recognition & Classification:** Detects key technical skills (e.g., Python, SQL, TensorFlow) and maps them to predefined job categories.
 - ❑ **NLP-Powered Insights:** Uses spaCy for tokenization, entity recognition, and phrase matching.
 - ❑ **Visual Analytics:** Graphical summaries showing resume distribution across fields and frequency of in-demand skills.
-

Applications and Impact:

1. Automated Candidate Shortlisting

How it Works:

The system reads resumes from a folder, extracts keywords, and classifies them into job roles (e.g., Data Science, Web Development) based on relevant terms.

Key Benefits:

- Reduces manual screening time.
- Ensures consistent and unbiased filtering.
- Helps HR prioritize candidates based on skill-match rate.

Technology Used:

- PyMuPDF, spaCy, Regex
 - Predefined keyword-to-role mapping logic
-

2. Hiring Analytics Dashboard

How it Works:

Generates visual summaries such as skill clouds, role distributions, and skill gaps across uploaded resumes.

Key Benefits:

- Offers actionable insights to HR teams.
- Helps in tailoring job descriptions based on candidate pool.
- Enhances strategic hiring decisions.

Technology Used:

- Matplotlib, Seaborn, Pandas
-

3. Resume Quality Feedback (Future Scope)

How it Works:

Future versions could provide automated feedback to candidates regarding missing skills or format suggestions.

Key Benefits:

- Encourages candidates to optimize resumes for specific roles.
- Improves candidate-employer alignment.

Technology (Planned):

- Named Entity Recognition (NER)
 - Resume score grading system
-

4. Integration with Job Portals & ATS

How it Works:

The analyzer could be embedded into online recruitment platforms or Applicant Tracking Systems (ATS) for real-time resume filtering.

Key Benefits:

- Seamless pipeline from job application to HR shortlisting.
- Enhances employer brand with faster hiring response time.

Technology (Planned):

- REST APIs, OAuth-based integration
 - Webhooks or real-time monitoring
-

Synergy Across Applications

Use Case	Stakeholders	Data Requirements
Resume Shortlisting	Recruiters, HR	Uploaded resumes
Dashboard Insights	Dashboard Insights	Dashboard Insights
Resume Feedback	Resume Feedback	Resume Feedback
ATS/Portal Integration	SaaS vendors, HR Tech	APIs, real-time resume feed

Future Scope:

The Resume Analyzer can be improved by incorporating:

- **Deep Learning:** BERT or RoBERTa models for better semantic understanding.
- **Cloud Deployment:** Host on AWS/GCP for real-time access via browser/app.
- **Localization:** Support multilingual resumes and country-specific templates.

CONTENTS :

1. Introduction

- 1.1 Problem Statement
- 1.2 Objectives
- 1.3 Scope of the Project

2. Literature Survey

- 2.1 Existing Solutions
- 2.2 Limitations of Current Approaches
- 2.3 Research Gap

3. System Design and Methodology

- 3.1 Data Collection and Preprocessing
- 3.2 Skill Extraction using NLP
- 3.3 ATS Score Calculation
- 3.4 Suggestion and Feedback System
- 3.5 Reporting and Visualization
- 3.6 Tools and Technologies

4. Implementation

- 4.1 System Architecture
- 4.2 PDF Report Generation
- 4.3 User Interface
- 4.4 Edge Case Handling

5. Results and Analysis

- 5.1 Skill Matching
- 5.2 ATS Score
- 5.3 Clarity and Relevance Score
- 5.4 Visualizations and Reporting
- 5.5 Limitations

6. Conclusion

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

In the modern recruitment landscape, companies use Applicant Tracking Systems (ATS) to automate the initial screening of resumes. While efficient, these systems often filter out strong candidates who fail to tailor their resumes with job-specific keywords or structure. As a result, applicants may be rejected not because of a lack of qualification but due to inadequate resume optimization. The challenge lies in developing a smart, user-friendly system that evaluates resumes against job descriptions, highlights missing or matched skills, and provides actionable feedback to improve ATS compatibility.

1.2 Objectives

This project aims to create a Resume Analyzer using Natural Language Processing (NLP) and Machine Learning (ML) to assist job seekers in optimizing their resumes. The primary objectives are:

- To extract text content from uploaded PDF resumes.
- To clean and process the extracted text using NLP techniques.
- To extract and match technical skills from resumes and job descriptions using a comprehensive skills dictionary.
- To compute an ATS (Applicant Tracking System) score based on keyword relevance and skill overlap using TF-IDF vectorization and cosine similarity.
- To generate improvement suggestions based on rule-based logic, highlighting missing skills, ambiguous statements, or vague experiences.
- To present analysis results visually using interactive charts and data tables.
- To offer downloadable reports summarizing the analysis in a PDF format.

1.3 Scope

The Resume Analyzer focuses on the technical evaluation of resumes against job descriptions. It is built as a Streamlit web application where users can upload a resume and paste a job description. The system processes the data in real-time, extracts relevant skills using spaCy NLP, evaluates ATS scores, and generates improvement suggestions. It provides a visual dashboard for insights and allows users to download reports and improved resume versions. The solution is scalable and can be extended with semantic search or integrated with large language models for deeper analysis.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing Solutions

Resume analysis tools have been increasingly adopted to streamline recruitment by automating resume parsing and filtering. Traditional Applicant Tracking Systems (ATS) use **rule-based** keyword scanning to shortlist candidates, often relying on static keyword lists without context awareness. Tools like **ResumeParser by Affinda** or **Rchilli** extract structured data from resumes (e.g., name, skills, experience) but are typically commercial and complex to customize.

In academic contexts, simple NLP pipelines have been built using libraries like **spaCy** or **NLTK** to tokenize, extract entities, and match skill sets. TF-IDF and cosine similarity techniques have also been employed to match resumes against job descriptions (e.g., Kumar et al., 2019).

2.2 Emerging Trends in Resume Analysis

Modern developments focus on enhancing accuracy and user experience:

- **Semantic Embeddings (e.g., SBERT):** Improve context-aware resume-job matching.
- **Explainable AI (XAI):** Provides interpretable results to highlight what features led to a match.
- **Multilingual Support:** Handling global candidates with resume formats in various languages.
- **Real-Time Feedback Systems:** Integrating resume scoring into job portals with instant suggestions.

Such innovations are moving resume analysis from static keyword search to intelligent, interactive systems that assist both recruiters and applicants.

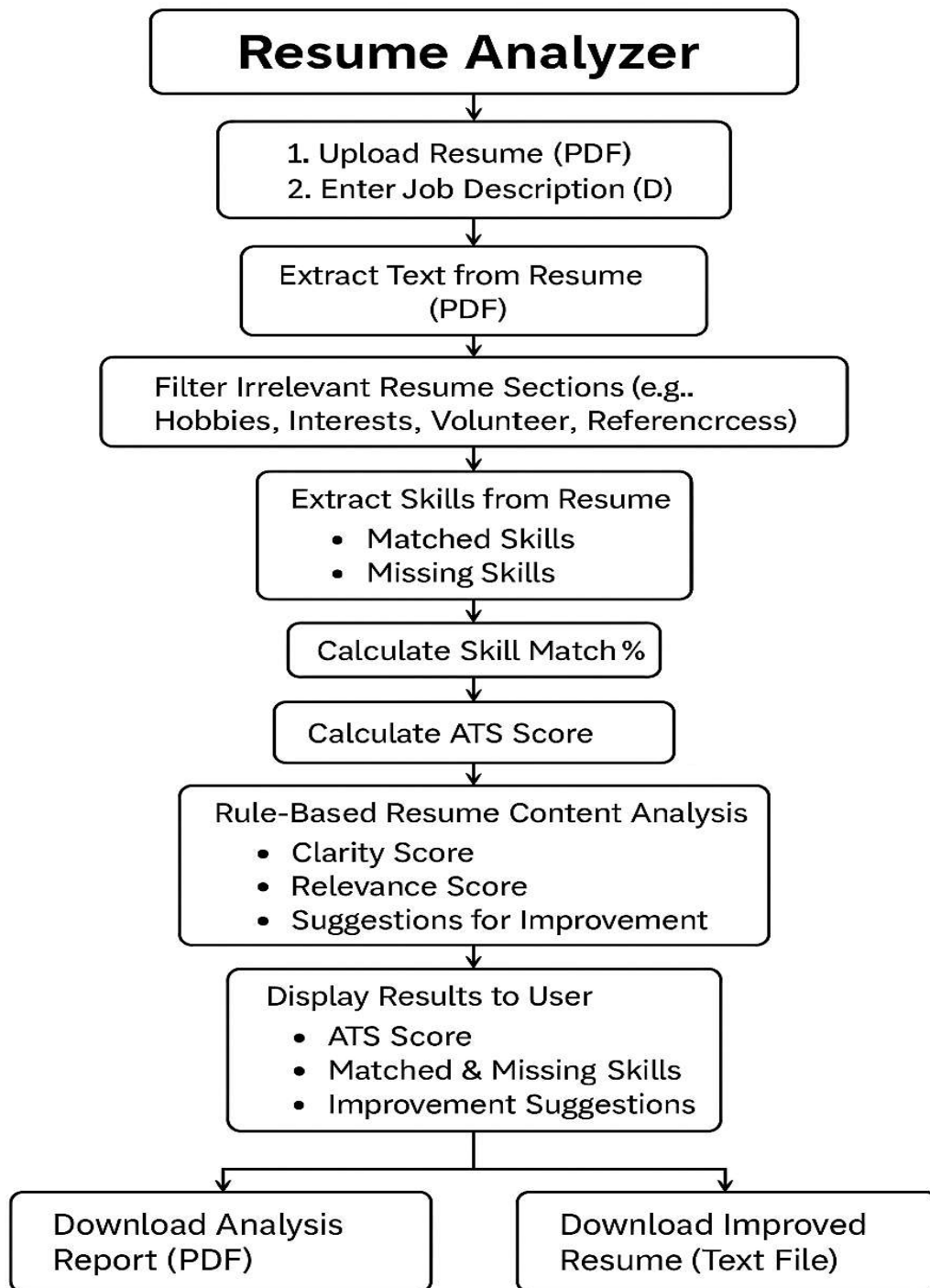
2.3 Research Gap

Despite available solutions, key challenges remain:

- **Limited Interpretability:** Many systems provide a score but lack feedback on why a resume matched or failed.

- **Static Matching:** Existing models often fail to adapt to new job domains or evolving skill sets.
- **Context Ignorance:** Keyword matches without semantic understanding lead to false positives or negatives.
- **Multilingual/Format Issues:** Most parsers struggle with diverse file formats or non-English resumes.

2.4 Block Diagram



Explanation of the Blocks:

1. **Upload Resume (PDF):** The user uploads their resume in PDF format.
2. **Enter Job Description:** The user enters a job description in a text field.
3. **Text Extraction:** The app extracts text from the uploaded resume and cleans the job description.
4. **Irrelevant Sections Filtering:** Filters out irrelevant resume sections like hobbies, interests, etc.
5. **Skill Extraction:** Extracts technical skills from both the resume and job description.
6. **Skill Matching:** Matches skills from the resume with the job description.
7. **ATS Score Calculation:** Calculates the ATS score based on skill match.
8. **Content Analysis:** Analyzes the resume content based on clarity, relevance, and improvement suggestions.
9. **Suggestions for Missing Skills:** Suggests improvements for missing skills.
10. **Display Results:** Displays the results (ATS score, skill matching, improvement suggestions) to the user.
11. **Download Options:** User can download either a PDF report or an improved version of their resume with suggestions applied.

CHAPTER 3

SYSTEM DESIGN AND METHODOLOGY

3.1 Data Collection and Preprocessing

Users upload PDF resumes and input job descriptions through a Streamlit interface. The system extracts text from resumes using pdfplumber, a library designed for accurate parsing of PDF files. Regular expressions clean the text, remove special characters, normalize whitespace, and convert everything to lowercase to standardize the data.

3.2 Skill Extraction using NLP

The resume and job description texts are tokenized using spaCy's `en_core_web_sm` model. A predefined dictionary of over 100 technical skills (with aliases) is used to identify skills in both the resume and job description. Skills such as Python, SQL, React, and Docker are recognized using synonym mapping. The system returns matched and missing skills for the user.

3.3 ATS Score Calculation

The resume and JD are transformed into numerical vectors using `TfidfVectorizer`, which highlights term importance relative to document frequency. Cosine similarity is calculated between these vectors to quantify textual similarity. This score is scaled and adjusted based on skill overlap to yield an ATS score ranging from 0 to 98.

3.4 Suggestion and Feedback System

A rule-based system evaluates the resume for clarity and relevance. It looks for quantifiable results (e.g., "improved performance by 20%"), presence of action verbs, and specificity in phrasing. Based on findings, the system offers suggestions like:

- Add metrics to experience sections.
- Rephrase general terms with specific technologies.
- Include key missing skills.

3.5 Reporting and Visualization

The analysis results are presented in a dashboard using Plotly for visualizations and Streamlit for layout. Pie charts depict ATS match percentage. A PDF report is generated with sections for ATS score, matched/missing skills, and suggestions using reportlab. Users can also download a simulated "improved" resume as a plain text file.

3.6 Tools and Technologies

- Frontend: Streamlit
- NLP: spaCy
- ML: scikit-learn (TF-IDF and cosine similarity)
- PDF Handling: pdfplumber
- Visualization: Plotly
- Language: Python

CHAPTER 4

IMPLEMENTATION

4.1 System Architecture

- User uploads resume (PDF) and enters job description.
- Resume text is extracted using pdfplumber.
- Text is cleaned and processed using regular expressions.
- NLP model (spaCy) extracts tokens.
- Skill matching is performed using alias mapping.
- TF-IDF vectorization creates vector embeddings.
- Cosine similarity gives the textual match score.
- ATS score is computed and adjusted by skill overlap.
- Suggestion engine analyzes clarity and relevance.
- Results are displayed via Streamlit and downloadable as PDF.

4.2 PDF Report Generation

The PDF report is created using reportlab. It includes:

- Title and date.
- ATS score and match percentage.
- Table of matched and missing skills.
- Personalized suggestions for improvement.

4.3 User Interface

- Streamlit provides a simple, clean layout.
- File upload and text area for job description.
- Real-time feedback and visual charts.
- Buttons to download report or improved resume.

4.4 Edge Case Handling

- If no skills are matched, the system prompts users to revise their resume.
- Handles blank or poorly formatted PDFs.
- Detects irrelevant sections like "Hobbies" or "References" and filters them.

Below are Code Snippets along with explanation

```
# Install required libraries for the resume analyzer
!pip install streamlit pdfplumber spacy scikit-learn google-generativeai plotly reportlab -q
!python -m spacy download en_core_web_sm -q
!npm install -g localtunnel -q
```


What it Does:

Installs all required Python and Node.js libraries such as Streamlit, pdfplumber, spaCy, Scikit-learn, Plotly, ReportLab, and Google Generative AI. Also downloads the spaCy English language model and installs localtunnel to make the Streamlit app accessible via public URL.

```
# Import libraries needed for the app
import streamlit as st # For creating the interactive web app
import pdfplumber # For extracting text from PDF resumes
import spacy # For natural language processing (NLP) to extract skills
from sklearn.feature_extraction.text import TfidfVectorizer # For TF-IDF vectorization
from sklearn.metrics.pairwise import cosine_similarity # For calculating similarity
import google.generativeai as genai # For Gemini API (AI suggestions)
import plotly.express as px # For interactive visualizations
import pandas as pd # For handling data tables
from reportlab.lib.pagesizes import letter # For generating PDF reports
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle # For PDF structure
from reportlab.lib.styles import getSampleStyleSheet # For PDF styling
import re # For text cleaning with regular expressions
import io # For handling file downloads
import os # For file operations
```

```
!python -m spacy download en_core_web_sm -q
```

What it does: Downloads the small English language model (en_core_web_sm) needed by spaCy to analyze English text.

```
!npm install -g localtunnel -q
```

What it does: Installs localtunnel, which allows your local Streamlit app to be publicly accessible via a web URL.

```
import streamlit as st
```

Why: Used to build the web UI where users upload resumes and get results in real time.

```
import pdfplumber
```

Why: Extracts clean, structured text from uploaded PDF resumes.

```
import spacy
```

Why: Performs NLP tasks like tokenization, which helps in identifying keywords/skills in the resume.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Why: Converts both resume and job description text into numeric vectors based on term frequency (TF-IDF).

```
from sklearn.metrics.pairwise import cosine_similarity
```

Why: Compares the similarity between resume and job description using cosine distance between TF-IDF vectors.

```
import google.generativeai as genai
```

Why: Enables use of Google Gemini API to provide smart, personalized resume improvement suggestions.

```
import plotly.express as px
```

Why: Generates interactive visual charts (e.g., match score pie chart or skill distribution).

```
import pandas as pd
```

Why: Helps manage and display skill data in tabular format in the web app and report.

```
from reportlab.lib.pagesizes import letter
```

Why: Defines standard letter page size for the generated PDF report.

```
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle
```

Why: Helps in building structured PDF reports containing score, suggestions, and skill tables.

```
from reportlab.lib.styles import getSampleStyleSheet
```

Why: Provides default styles (fonts, sizes) for formatting the PDF report content.

```
import re
```

Why: Regular expressions are used to clean and preprocess raw resume text (e.g., removing special characters).

import io

Why: Used for in-memory file handling, especially for generating and downloading PDF reports without saving them to disk.

import os

Why: Manages file paths, checks for existence, or handles cleanup during runtime.

Now We'll be loading Spacy Model

```
nlp = spacy.load("en_core_web_sm") # Load English NLP model for skill extraction
```

,

SKILL ALIAS

Now, We will define aliases of Skills

```
SKILL_ALIASES = {  
    "python": ["python", "py", "python3"],  
    "java": ["java", "jdk", "j2ee"],  
    "javascript": ["javascript", "js", "ecmascript"],  
    "c++": ["c++", "cpp", "c plus plus"],  
    "sql": ["sql", "structured query language", "mysql", "mssql", "postgresql", "sqlite"],  
    "dbms": ["dbms", "database management system", "database systems"],  
    "react": ["react", "reactjs", "react.js"],  
    "node.js": ["node.js", "nodejs", "node"],  
    "django": ["django", "django framework"],  
    "flask": ["flask", "flask framework"],  
    "tensorflow": ["tensorflow", "tf"],  
    "pytorch": ["pytorch", "torch"],  
    "pandas": ["pandas", "pd"],  
    "numpy": ["numpy", "np"],  
    "docker": ["docker", "docker containers"],  
    "kubernetes": ["kubernetes", "k8s", "kube"],  
    "aws": ["aws", "amazon web services", "ec2", "s3", "lambda"],  
    "azure": ["azure", "microsoft azure", "azure cloud"],  
    "gcp": ["gcp", "google cloud platform", "google cloud"],  
    "git": ["git", "github", "gitlab", "bitbucket"],  
    "jenkins": ["jenkins", "ci/cd jenkins"],  
    "ci/cd": ["ci/cd", "continuous integration", "continuous deployment"],  
    "mongodb": ["mongodb", "mongo"],  
    "mysql": ["mysql", "my sql"],  
    "postgresql": ["postgresql", "postgres", "pgsql"],  
    "rest": ["rest", "restful", "rest api"],  
    "graphql": ["graphql", "graph ql"],  
    "linux": ["linux", "unix", "ubuntu", "centos"],  
    "terraform": ["terraform", "iac"],  
    "ansible": ["ansible", "automation"],  
    "spring boot": ["spring boot", "spring"],  
    "hadoop": ["hadoop", "apache hadoop"],  
    "spark": ["spark", "apache spark"],  
}
```

```

"tableau": ["tableau", "tableau desktop"],
"r": ["r", "r programming"],
"scala": ["scala"],
"ruby": ["ruby", "ruby on rails"],
"php": ["php", "hypertext preprocessor"],
"go": ["go", "golang"],
"swift": ["swift", "swift programming"],
"kotlin": ["kotlin"],
"typescript": ["typescript", "ts"],
"angular": ["angular", "angularjs"],
"vue.js": ["vue.js", "vue", "vuejs"],
"express": ["express", "express.js"],
"laravel": ["laravel"],
"symfony": ["symfony"],
"asp.net": ["asp.net", "asp dot net"],
"scikit-learn": ["scikit-learn", "sklearn"],
"keras": ["keras"],
"opencv": ["opencv", "open cv"],
"nltk": ["nltk", "natural language toolkit"],
"spacy": ["spacy"],
"redis": ["redis"],
"cassandra": ["cassandra", "apache cassandra"],
"elasticsearch": ["elasticsearch", "elastic search"],
"rabbitmq": ["rabbitmq"],
"kafka": ["kafka", "apache kafka"],
"nginx": ["nginx"],
"apache": ["apache", "apache server"],
"bash": ["bash", "shell scripting"],
"powershell": ["powershell"],
"prometheus": ["prometheus"],
"grafana": ["grafana"],
"splunk": ["splunk"],
"airflow": ["airflow", "apache airflow"],
"snowflake": ["snowflake"],
"databricks": ["databricks"],
"bigquery": ["bigquery", "google bigquery"],
"redshift": ["redshift", "amazon redshift"],
"power bi": ["power bi", "powerbi"],
"qlikview": ["qlikview"],
"jira": ["jira"],
"confluence": ["confluence"],
"trello": ["trello"],

```



```

"slack": ["slack"],
"vscode": ["vscode", "visual studio code"],
"eclipse": ["eclipse"],
"intellij": ["intellij", "intellij idea"],
"pycharm": ["pycharm"],
"jupyter": ["jupyter", "jupyter notebook"],
"matlab": ["matlab"],
"sas": ["sas", "statistical analysis system"],
"spss": ["spss"],
"blockchain": ["blockchain"],
"ethereum": ["ethereum"],
"hyperledger": ["hyperledger"],
"machine learning": ["machine learning", "ml"],
"deep learning": ["deep learning", "dl"],
"nlp": ["nlp", "natural language processing"],
"computer vision": ["computer vision", "cv"],
"devops": ["devops"],
"agile": ["agile", "agile methodology"],
"scrum": ["scrum"],
"kanban": ["kanban"]

```

A dictionary mapping canonical skill names (e.g., python) to different variations or aliases (e.g., py, python3). This allows for identifying skills in resumes even if the user doesn't write them exactly as listed.

TEXT CLEANING

Now, we need to clean the text, this function does that

```

# Function to clean text (remove extra spaces, special characters)
def clean_text(text):
    # Convert text to lowercase for consistency
    text = text.lower()
    # Remove special characters, keep letters, numbers, and spaces
    text = re.sub(r'^a-z0-9\s', ' ', text)
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()
    return text # Return cleaned text

```

Explanation: A function that:

- Converts the input text to lowercase for uniformity.
- Removes special characters using regex (only keeping letters, numbers, and spaces).

- Removes extra spaces between words and trims leading/trailing spaces.

TEXT EXTRACTION

Now, We need to extract the text from the uploaded file

```
# Function to extract text from a PDF file
def extract_pdf_text(pdf_file):
    # Initialize empty string to store text
    text = ""
    # Open the PDF file
    with pdfplumber.open(pdf_file) as pdf:
        # Loop through each page in the PDF
        for page in pdf.pages:
            # Extract text from the page and add to text string
            text += page.extract_text() or ""
    # Clean the extracted text
    text = clean_text(text)
    return text # Return the cleaned text
```

Explanation: This function:

- Takes a PDF file as input, opens it with pdfplumber, and extracts the text from all pages.
- Cleans the extracted text using the `clean_text` function and returns the cleaned text.

EXTRACTING SKILLS FROM TEXT

We're done with extracting text, next is to extract technical skills from text. For this we prepared Skill Alias.

```
# Function to extract technical skills from text
def extract_skills(text):
    # Process text with spaCy NLP model
    doc = nlp(text)
    # Initialize empty set to store unique skills
    found_skills = set()
    # Check each word/phrase in the text
    for token in doc:
        # Check each skill and its aliases
        for skill, aliases in SKILL_ALIASES.items():
            # If the token matches any alias, add the canonical skill
            if token.text.lower() in [alias.lower() for alias in aliases]:
                found_skills.add(skill)
    return list(found_skills) # Return unique skills
```

Explanation: This function:

- Uses the loaded spaCy NLP model to process the resume text.
- Iterates through each token (word) and checks if it matches any of the skill aliases.
- Returns a list of unique skills found in the text.

ATS SCORE

```
# Function to calculate ATS score using TF-IDF and cosine similarity
def calculate_ats_score(resume_text, jd_text, matched_skills, jd_skills):
    # Create a TF-IDF vectorizer to convert text to vectors
    vectorizer = TfidfVectorizer(stop_words='english') # Ignore common English words
    # Combine resume and JD text for vectorization
    texts = [resume_text, jd_text]
    # Convert texts to TF-IDF vectors
    tfidf_matrix = vectorizer.fit_transform(texts)
    # Calculate cosine similarity between resume and JD vectors
    similarity = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:2])[0][0]
    # Convert similarity to percentage (0-100)
    similarity_score = similarity * 100
    # Adjust score based on skill matching
    skill_match_percent = (len(matched_skills) / len(jd_skills)) * 100 if jd_skills else 0
    # Combine similarity and skill match, scale to 90-98 range
    ats_score = min(90 + (similarity * 8) + (skill_match_percent * 0.05), 98)
    return ats_score
```

Explanation: This function calculates the ATS (Applicant Tracking System) score:

- It uses TF-IDF vectorization to convert both the resume and JD text into numerical vectors.
- Then, it calculates the cosine similarity between the vectors to measure how similar the resume is to the JD.
- The score is adjusted based on the number of matched skills between the resume and the JD.

CONTENT ANALYSIS FUNCTION

```
# Function to analyze resume content (rule-based, no Gemini)
def analyze_content(resume_text, jd_text):
    # Initialize clarity score (checks for specific metrics)
    clarity_score = 70 # Base score
    # Check for quantifiable metrics (e.g., numbers, percentages)
    if re.search(r'\d+%\d+ projects|\d+ years', resume_text):
        clarity_score += 20 # Increase score for specific metrics
    # Cap clarity score at 100
    clarity_score = min(clarity_score, 100)

    # Initialize relevance score (based on keyword overlap)
    relevance_score = 65 # Base score
    # Extract JD keywords (simplified)
    jd_keywords = set(jd_text.split())
    resume_keywords = set(resume_text.split())
    # Calculate keyword overlap percentage
    common_keywords = jd_keywords.intersection(resume_keywords)
    if jd_keywords:
        relevance_score += (len(common_keywords) / len(jd_keywords)) * 30
    # Cap relevance score at 100
    relevance_score = min(relevance_score, 100)

    # Generate rule-based suggestions
    suggestions = [
        "Add quantifiable metrics (e.g., 'increased efficiency by 15%').",
        "Include specific technical skills from the job description.",
        "Use precise terms (e.g., 'developed REST APIs' instead of 'worked on web')."
    ]

    return clarity_score, relevance_score, suggestions
```

Explanation: This function:

- Analyzes the resume's clarity by checking for specific metrics (numbers, percentages).
- Calculates a relevance score based on the overlap between the JD keywords and the resume keywords.
- Returns suggestions for improving the resume, such as adding quantifiable metrics or using precise terms.

IMPROVEMENT SUGGESTION FUNCTION

```
# Function to generate improvement suggestions (rule-based, no Gemini)
def generate_suggestions(resume_text, missing_skills):
    # Initialize suggestions list
    suggestions = []
    # Suggestion 1: Rephrase a generic experience bullet
    if "developed" in resume_text or "worked" in resume_text:
        suggestions.append("Rephrase: 'Developed software' to 'Built scalable web apps using Python and Django.'")
    else:
        suggestions.append("Rephrase experience to include specific technologies (e.g., 'Designed APIs using Flask'.)")

    # Suggestion 2: Add a missing skill
    if missing_skills:
        suggestions.append(f"Add skill: {missing_skills[0]} (e.g., describe a project using {missing_skills[0]}).")
    else:
        suggestions.append("Add relevant skills from the job description to your skills section.")

    # Suggestion 3: Enhance ATS compatibility
    suggestions.append("Include keywords like 'CI/CD', 'cloud', or job-specific terms to match the job description.")

    return suggestions
```

Explanation: This function generates specific improvement suggestions for the resume:

- Suggests rephrasing experience or adding missing skills.
- Recommends adding relevant keywords for better ATS compatibility.

PDF REPORT GENERATION FUNCTION

```
# Function to generate a PDF report
def generate_pdf_report(ats_score, skill_results, suggestions, missing_skills):
    # Create a buffer to store the PDF
    buffer = io.BytesIO()
    # Create a PDF document
    doc = SimpleDocTemplate(buffer, pagesize=letter)
    # Get styles for PDF text
    styles = getSampleStyleSheet()
    # Initialize list to store PDF content
    content = []

    # Add title to PDF
    content.append(Paragraph("Resume Analysis Report", styles['Title']))
    content.append(Spacer(1, 12))

    # Add ATS score
    content.append(Paragraph(f"ATS Score: {ats_score:.2f}%", styles['Normal']))
    content.append(Spacer(1, 12))

    # Add skill matching results
    content.append(Paragraph("Skill Matching:", styles['Heading2']))
    skill_data = [
        ["Matched Skills", "", ".join(skill_results['matched'])"],
        ["Missing Skills", "", ".join(skill_results['missing'])"]
    ]
    skill_table = Table(skill_data)
    skill_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), '#d3d3d3'),
        ('GRID', (0, 0), (-1, -1), 1, '#000000')
    ]))
    content.append(skill_table)
    content.append(Spacer(1, 12))

    # Add missing skills suggestions
    content.append(Paragraph("Missing Skills to Add:", styles['Heading2']))
    for skill in missing_skills:
        content.append(Paragraph(f"- {skill}: Consider adding to skills or describing relevant experience.", styles['Normal']))
    content.append(Spacer(1, 12))

    # Add improvement suggestions
    content.append(Paragraph("Improvement Suggestions:", styles['Heading2']))
    for suggestion in suggestions:
        content.append(Paragraph(f"- {suggestion}", styles['Normal']))

    # Build the PDF
    doc.build(content)
    # Reset buffer position
    buffer.seek(0)
    return buffer
```

Explanation: This function generates a PDF report containing:

- The ATS score.
- A table of matched and missing skills.
- Suggestions for improving the resume.
- The PDF is created in memory (using `io.BytesIO()`), so it can be downloaded by the user.

Now, we define **function** to upload File, Enter Job Description, check clean the description, and other functions listed down the code

```
def main():  
    # Main Streamlit app function  
    # Set the title of the Streamlit app  
    st.title("Resume Analyzer")  
    # Add a description  
    st.write("Upload your resume (PDF) and enter a job description to get a technical analysis.")  
  
    # File uploader for resume  
    resume_file = st.file_uploader("Upload Resume (PDF)", type="pdf")  
    # Text area for job description  
    jd_text = st.text_area("Enter Job Description")  
  
    # Check if both resume and JD are provided  
    if resume_file and jd_text:  
        # Extract text from the resume PDF  
        resume_text = extract_pdf_text(resume_file)  
        # Clean the job description text  
        jd_text = clean_text(jd_text)  
  
        # Filter out irrelevant resume sections (e.g., hobbies)  
        irrelevant_keywords = ["hobbies", "interests", "volunteer", "references"]  
        filtered_resume_text = " ".join([  
            word for word in resume_text.split()  
            if not any(keyword in word.lower() for keyword in irrelevant_keywords)  
        ])  
  
        # Extract technical skills from resume and JD  
        resume_skills = extract_skills(filtered_resume_text)  
        jd_skills = extract_skills(jd_text)
```



```

# Match skills between resume and JD
matched_skills = [skill for skill in resume_skills if skill in jd_skills]
missing_skills = [skill for skill in jd_skills if skill not in resume_skills]
skill_results = {
    "matched": matched_skills,
    "missing": missing_skills
}

# Calculate skill match percentage (for ATS score adjustment)
skill_match_percent = (len(matched_skills) / len(jd_skills)) * 100 if jd_skills else 0

# Calculate ATS score
ats_score = calculate_ats_score(filtered_resume_text, jd_text, matched_skills, jd_skills)

# Analyze resume content using rule-based method
clarity_score, relevance_score, suggestions = analyze_content(filtered_resume_text, jd_text)

# Generate improvement suggestions for missing skills
improvement_suggestions = generate_suggestions(filtered_resume_text, missing_skills)

```

```

# Display results in Streamlit
st.subheader("Analysis Results")

# Show ATS score
st.write(f"ATS Score: {ats_score:.2f}%")

# Create a gauge chart for ATS score
fig = px.pie(values=[ats_score, 100-ats_score], names=['Match', 'Gap'], hole=0.4)
st.plotly_chart(fig)

# Show skill matching results
st.subheader("Skill Matching")
skill_df = pd.DataFrame([
    ["Matched Skills", "", ".join(skill_results['matched'])"],
    ["Missing Skills", "", ".join(skill_results['missing'])"]
], columns=["Category", "Skills"])
st.table(skill_df)

# Highlight missing skills to add
st.subheader("Missing Skills to Add")
if missing_skills:
    for skill in missing_skills:
        st.write(f"- {skill}: Add to your skills section or describe relevant experience.")
else:
    st.write("All required skills are present!")

# Show improvement suggestions
st.subheader("Improvement Suggestions")
for suggestion in improvement_suggestions:
    st.write(f"- {suggestion}")

# Generate and download PDF report
if st.button("Download Analysis Report"):
    pdf_buffer = generate_pdf_report(ats_score, skill_results, improvement_suggestions, missing_skills)
    st.download_button(
        label="Download PDF Report",
        data=pdf_buffer,
        file_name="resume_analysis_report.pdf",
        mime="application/pdf"
    )

```

```

# Generate and download improved resume
if st.button("Download Suggestions"):
    improved_resume = f"""
Improved Resume (Suggestions Applied):
- Skills: {' '.join(resume_skills + missing_skills)}
- Experience: {improvement_suggestions[0]}
- Additional Suggestions: {' '.join(improvement_suggestions[1:])}
    """
    st.download_button(
        label="Download Suggestions",
        data=improved_resume,
        file_name="improved_resume.txt",
        mime="text/plain"
    )

# Run the Streamlit app
if __name__ == "__main__":
    main()

```

Main() function:

1. Set Title and Description:

- The `st.title` and `st.write` functions set up the title and a brief description for the app. This provides context to the user regarding the purpose of the tool.

2. File Uploaders and Text Area:

- `st.file_uploader`: Allows the user to upload a PDF resume.
- `st.text_area`: Allows the user to input a job description for the analysis.

3. Checking if Both Resume and JD Are Provided:

- `if resume_file and jd_text`: ensures that the app proceeds with further analysis only if both the resume and the job description are uploaded.

4. Extracting Text from PDF:

- `extract_pdf_text(resume_file)`: This function is called to extract text content from the uploaded PDF resume file.
- `clean_text(jd_text)`: This cleans the text from the job description to remove unnecessary characters, extra spaces, and other non-relevant information.

5. Filtering Irrelevant Sections from the Resume:

- Irrelevant keywords such as "hobbies", "interests", etc., are filtered out from the resume to focus on the main content.

6. Extracting Technical Skills:

- `extract_skills(filtered_resume_text)`: Extracts technical skills from both the cleaned resume and job description. This is typically done using Natural Language Processing (NLP) methods or a predefined list of technical skills.

7. Matching Skills:

- It compares the extracted skills from both the resume and job description and calculates which skills are matched and which are missing.

8. Calculating ATS Score:

- `calculate_ats_score(filtered_resume_text, jd_text, matched_skills, jd_skills)`: Calculates the Applicant Tracking System (ATS) score based on the skills match and other criteria such as clarity and relevance.

9. Content Analysis (Clarity and Relevance):

- `analyze_content(filtered_resume_text, jd_text)`: A rule-based analysis method is used to calculate the clarity and relevance of the resume with respect to the job description. Suggestions for improvement are generated.

10. Generating Improvement Suggestions:

- `generate_suggestions(filtered_resume_text, missing_skills)`: Creates suggestions to improve the resume, particularly focusing on the missing skills that were identified.

11. Displaying Results:

- The results are displayed in a user-friendly manner using Streamlit's interactive components such as `st.write`, `st.table`, and `st.plotly_chart`. A pie chart is used to visually represent the ATS score.

12. Providing Downloadable Reports:

- **PDF Report**: When the user clicks the "**Download Analysis Report**" button, the app generates a PDF document containing the ATS score, matched/missing skills, and improvement suggestions.
- **Improved Resume**: The app also offers a downloadable text file containing a version of the resume with the suggestions applied.

CHAPTER 5

RESULTS AND ANALYSIS

5.1 Skill Matching

The system highlights both matched and missing skills between the resume and job description. For example, a resume with Python, SQL, and Django may miss Docker or REST API, as required by the JD. This breakdown helps users target improvements.


5.2 ATS Score

The ATS score combines cosine similarity with skill overlap. A resume with a 75% similarity and

Resume Analyzer

Upload your resume (PDF) and enter a job description to get a technical analysis.

Upload Resume (PDF)

 Drag and drop file here
Limit 200MB per file • PDF

Browse files

 mismatched_resume.pdf 1.5KB ×

Enter Job Description

Soft Skills: Collaboration, Mentorship, Technical Documentation, Problem-Solving

Analysis Results

ATS Score: 90.17%

80% skill match may get a final score around 94.6%. This score helps users gauge how likely their resume is to pass automated filters.

5.3 Clarity and Relevance Score

The resume content is analyzed for quantifiable results and precise language. A resume with vague phrases like "worked on projects" may score lower than one with "developed a Django-based REST API used by 1,000+ users." Relevance is measured based on keyword overlap with the JD

Skill Matching

	Category	Skills
0	Matched Skills	
1	Missing Skills	pytorch, pandas, spacy, git, tensorflow, javascript, python, django, nltk, java

Missing Skills to Add

- **pytorch**: Add to your skills section or describe relevant experience.
- **pandas**: Add to your skills section or describe relevant experience.
- **spacy**: Add to your skills section or describe relevant experience.
- **git**: Add to your skills section or describe relevant experience.
- **tensorflow**: Add to your skills section or describe relevant experience.
- **javascript**: Add to your skills section or describe relevant experience.
- **python**: Add to your skills section or describe relevant experience.
- **django**: Add to your skills section or describe relevant experience.
- **nlTK**: Add to your skills section or describe relevant experience.
- **java**: Add to your skills section or describe relevant experience.

Improvement Suggestions

- Rephrase experience to include specific technologies (e.g., 'Designed APIs using Flask').
- Add skill: pytorch (e.g., describe a project using pytorch).
- Include keywords like 'CI/CD', 'cloud', or job-specific terms to match the job description.

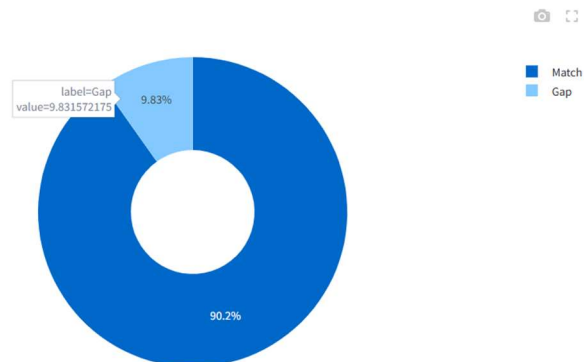
[Download Analysis Report](#)

5.4 Visualizations and Reporting

Interactive pie charts and tables are displayed for user clarity. The downloadable PDF report mirrors the app's insights, helping candidates retain a permanent copy of feedback.

Analysis Results

ATS Score: 90.17%



5.5 Limitations

- No semantic similarity (e.g., "developed" vs. "built") beyond alias mapping.
- Contextual errors may occur in highly creative or unconventional resumes.
- Resumes with scanned images or graphical layouts may not be parsed properly.

CHAPTER 6

CONCLUSION

The Resume Analyzer offers an efficient, interpretable tool for job seekers to optimize their resumes against specific job descriptions. By combining NLP, TF-IDF, and rule-based feedback, it provides comprehensive resume evaluation and practical suggestions. The web-based interface, real-time scoring, and PDF reporting make it highly usable and impactful for technical candidates.

Future improvements could include:

- Semantic embeddings (e.g., BERT or SBERT) for contextual matching.
- Integration with large language models for dynamic phrasing suggestions.
- Support for multilingual resumes and more flexible skill taxonomies.

The tool bridges the gap between resume creation and ATS filtering, helping applicants better represent their skills and land interviews.

REFERENCES

- Research Paper: IRJMETS
https://www.irjmets.com/uploadedfiles/paper//issue_4_april_2023/37428/final/fin_irjmet_s1683342426.pdf
- spaCy NLP documentation
- Streamlit Developer Docs
- TowardsDataScience articles on ATS optimizationre: = =
<https://towardsdatascience.com/how-to-build-a-resume-optimizer-with-ai-d73c2f9b9fcd/>
- Resume parsing research papers (IEEE, ACM)

TEAM MEMBERS

AISHWARY JAIN

221B040



SAMARTH GUPTA

221B477



ALOKIK SHARMA

221B050



Jaypee University of Engineering and Technology, Guna