Resumate Miner Documentation

Link to deployed site: http://aryans.tech/

Table of contents

About Resumate Miner

- Overview of the application
- Purpose of the project
- Key Features

Technologies Used

- Streamlit library
- Python

Functionality

- How the parser works
- Integration of parsel into streamlit

Pages

- Resume parsel

- Analytics
- Ranking
- Test Score

About Resumate Miner

Overview of the tool -

Equipped with the best filtering and searching techniques, ResumateMiner helps you browse a high volume of resumes quickly.

Purpose of the Project -

This tool helps our company to run through a high volume of resumes and select the best of the best candidates for job opportunities in their respective skills.

Key Features -

ResumateMiner's searching and filtering technique is what sets it apart.

We have implemented a tool that helps parse the PDF resumes of candidates and display them in the form of

various analytics, which helps recruiters hire worthy candidates based on their skills. The searching and filtering technique in our project also helps to shortlist various candidates based on particular queries.

Technologies used -

Streamlit library -

Streamlit is an open-source Python library that allows you to create interactive web applications for data science and machine learning projects with minimal effort. It's designed to enable data scientists and developers to quickly build user interfaces for their data-driven projects without requiring a deep understanding of web development. Streamlit has gained popularity due to its simplicity and rapid prototyping capabilities.

Here's a detailed breakdown of the Streamlit library

1. Creating Apps:

Streamlit apps are typically structured as simple Python scripts. You write the code in a linear fashion, and Streamlit automatically converts the code into a web app.

2. App Structure:

A basic Streamlit app follows this structure:

```
import streamlit as st

# Streamlit code goes here

if __name__ == '__main__':
    main()
```

The `main()` function is where you'll put your Streamlit code.

3. Widgets:

Streamlit provides a wide range of widgets that you can use to interact with your data. These widgets include sliders, buttons, text inputs, checkboxes, and more. You can use these widgets to take user input and display data dynamically.

4. Data Display:

You can display data using Streamlit's various built-in functions. For example:

```
st.write("Hello, Streamlit!")  # Display text
st.dataframe(dataframe)  # Display a Pandas DataFrame
st.pyplot(figure)  # Display a Matplotlib figure
```

5. Interactive Elements:

Streamlit allows you to make your apps interactive. For example, you can create a slider and use its value to update a plot dynamically:

```
python
x = st.slider("Select a value", 0, 100, 50)
st.line_chart(data[x:x+10])
```

6. Caching:

Streamlit provides caching capabilities to optimize performance. If a computation is time-consuming and doesn't change frequently, you can use the `@st.cache` decorator to cache its result.

7. Layout and Design:

Streamlit automatically arranges widgets in a single-column layout by default. However, you can

customize the layout using columns, sidebars, and other functions to organize your app's components.

8. Deployment:

Once you've created your Streamlit app, you can deploy it easily using various platforms such as Streamlit Sharing, Heroku, AWS, or any web server that supports Python applications.

9. Integration:
Streamlit can be integrated with other popular libraries and frameworks such as Pandas, Matplotlib, Plotly, and sci-kit-learn, making it an excellent tool for showcasing data analysis and machine learning models.

Python-

Python is a high-level, interpreted, and dynamically typed programming language emphasizing simplicity, readability, and ease of use. Here are a few features of Python -

1. Interpreted Language
2. Dynamically Typed
3. High-Level Constructs
4. Extensive Standard Library
5. Dynamic Memory Management

6. Cross-Platform
7. Indentation and Whitespace
8. Scripting and Automation
9. Object Oriented

Functionality-

How does the parser work?

The parser in the code follows a multi-step process to extract and parse information from resumes:

1. Text Extraction from PDF:

The first step is to extract the text from the uploaded resume PDF file. This is done using the `PDFMiner` library, which is a tool for extracting text and metadata from PDF files.

2. Text Parsing with spaCy:

After extracting the text from the PDF, the code uses spaCy with the `en_core_web_sm` model to perform natural language processing (NLP) on the text. SpaCy is a powerful NLP library that can parse and analyze text data.

3. Regex Searching for Specific Details:

The code utilizes regular expressions (regex) to search for specific details within the parsed text. These details can include items like degrees, names, mobile numbers, CPI (Cumulative Performance Index), and more.

4. Text Segmentation:

The parsed text is divided into different sections, such as projects, courses, positions of responsibility (PORs), etc. This segmentation is typically based on the resume's structure and formatting.

5. Skill Extraction using a Large Dataset:

The code likely uses a predefined dataset of skills or keywords to extract skills mentioned in the resume. It checks the parsed text against this dataset to identify and extract the skills mentioned by the applicant.

6. Storing Details in JSON Object:

Finally, the extracted information is stored in a JSON object. This JSON object likely contains structured data with keys representing different sections or types of information (e.g., "name," "mobile number," "education," "skills," etc.), and the corresponding values are the extracted details.

The key steps involve text extraction, NLP-based parsing, regex pattern matching, and skill extraction, all of which work together to extract and structure information from the uploaded resumes. The resulting JSON object contains organized resume data that can be used for further analysis or display in the application.

Integration of parser into streamlit -

Here's how the integration of the parser into streamlit works -

1. Using Python Libraries:

The resume parser and Streamlit are Python libraries, making it straightforward to integrate them.

2. Calling the Parser Component:

In the "Resume Parser" section of the Streamlit application, the code allows users to upload their resumes using `st.file_uploader`. Users can drag and drop a PDF file into the application.

3. Parsing with ResumeParser Function:

When a resume is uploaded, the `ResumeParser` function from the `pyresparser` library is called to parse the content of the resume.
The `ResumeParser` function takes the uploaded file as input and extracts structured information such as name, contact details, education, skills, and more from the resume.

4. Storing Parsed Data:
The parsed data is stored in the `data1` variable as a JSON object.
This data is then used to display the resume details in the right column of the "Resume Parser" section, including sections for name, contact details, education, skills, etc.

5. Data Storage for Analytics:

The parsed resume data is also stored in a JSON object or file (possibly `applicants_data.json`), which can be later used for analytics and ranking purposes in other application sections.

6. User Interaction:

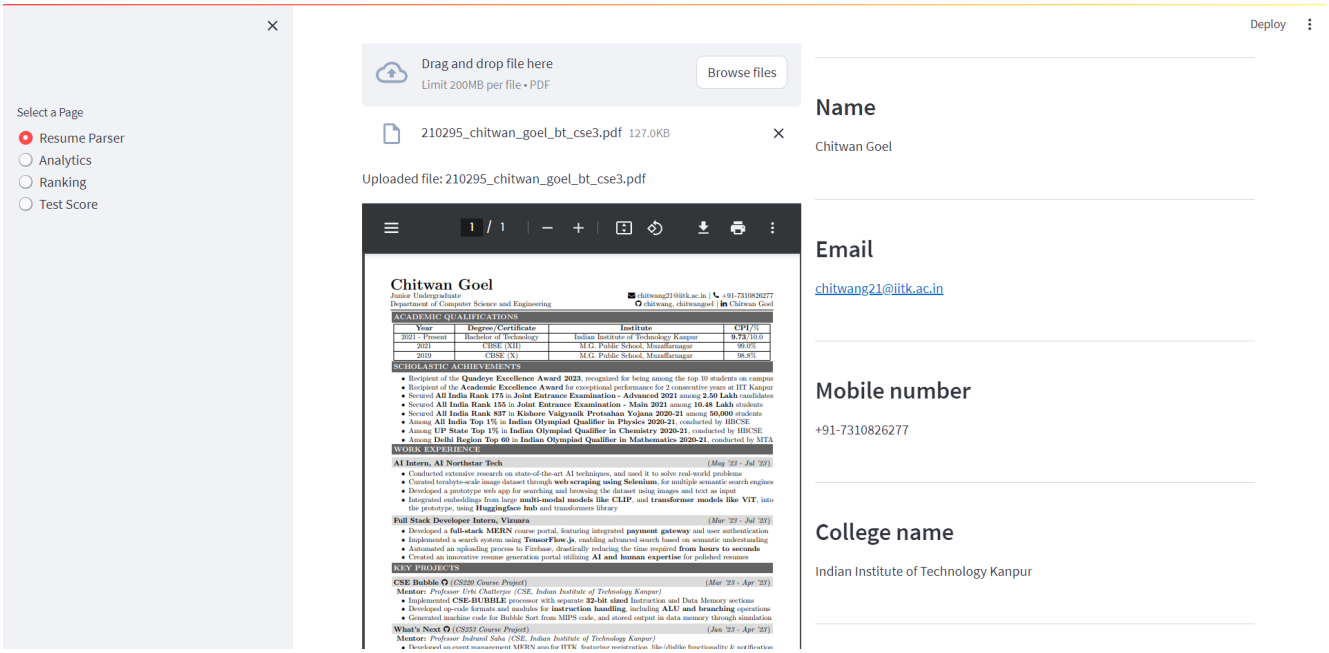
Users can easily interact with the Streamlit application by uploading their resumes and instantly seeing the parsed information.
The application provides a user-friendly interface for visualizing the parsed resume content.

In summary, the integration of the parser into Streamlit involves calling the `ResumeParser` function to extract structured data from uploaded resumes. This parsed data is then displayed to the user and stored for further analytics, making it a seamless and user-friendly experience within the Streamlit application.

Pages-

Resume Parser-

On this page, we upload the resume of the candidate in PDF format, which then displays the overview of the candidate's resume, which includes their personal information, education, experience, and skills.



Analytics Page-

The page implements a provided code in Python script that uses the Streamlit library to create an interactive web-based analytics dashboard. The dashboard allows users to analyze data related to job applicants,

specifically comparing their Cumulative Performance Index (CPI) and identifying the top-mentioned skills. Let's break down the code step by step:

1. Import Libraries:

The script starts by importing necessary libraries: `streamlit`, `json`, `pandas`, and `numpy`. These libraries are used for creating the web application, loading JSON data, data manipulation, and calculations.

2. CPI Comparison Option:

The code begins with a conditional block that executes if the selected option is "Analytics."
Inside this block, the title "Analytics Page" is displayed using `st.subheader`.

- **Sidebar and Analytics Options:**

The analytics options are defined in the `analytics_options` list, which includes "CPI Comparison," "Top Mentioned Skills," and "Most Projects or Experience."
The user can choose an analytics option from a sidebar using `st.sidebar.radio`. The selected option is stored in the `selected_analytics_option` variable.

- **Load Data and Create DataFrame:**

The code loads data from a JSON file named 'applicants_data.json' using the `json` library.

The data is then converted into a Pandas DataFrame named `applicants_df`, which holds information about applicants.

- **CPI Comparison Chart:**

If the selected analytics option is "CPI Comparison," the code proceeds to create a CPI comparison chart.

Users can select a range of CPI values using a slider (`st.slider`).

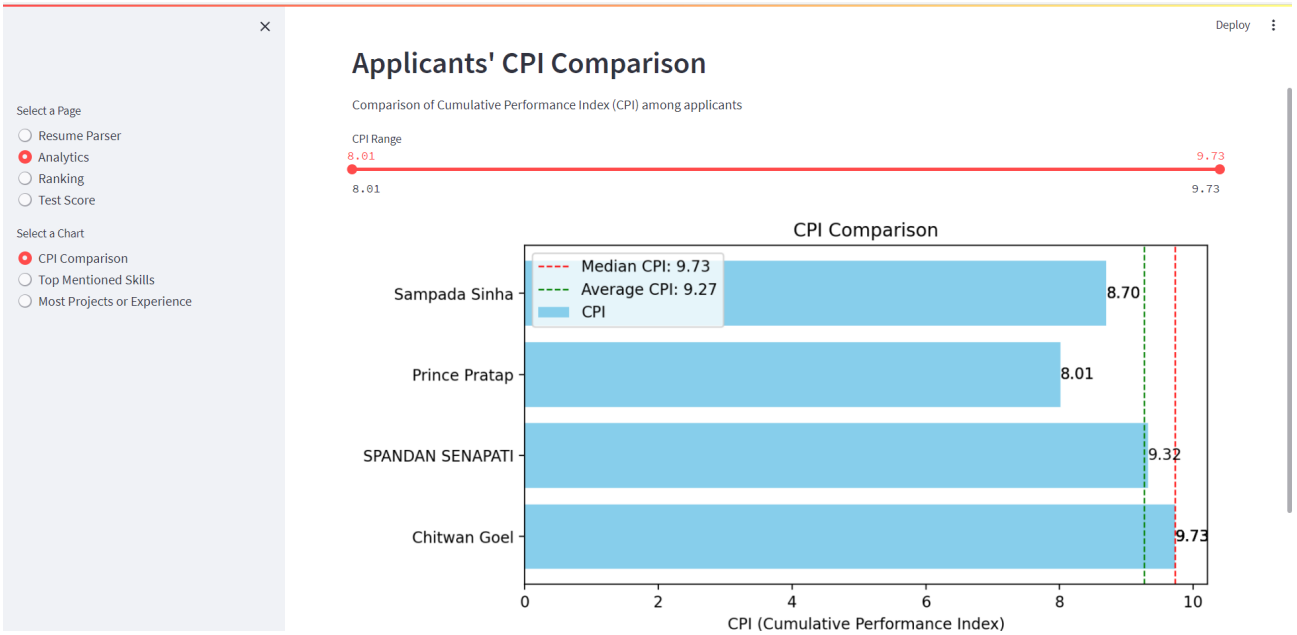
The DataFrame is filtered to include only applicants within the selected CPI range.

Median and average CPI values are calculated using NumPy functions.

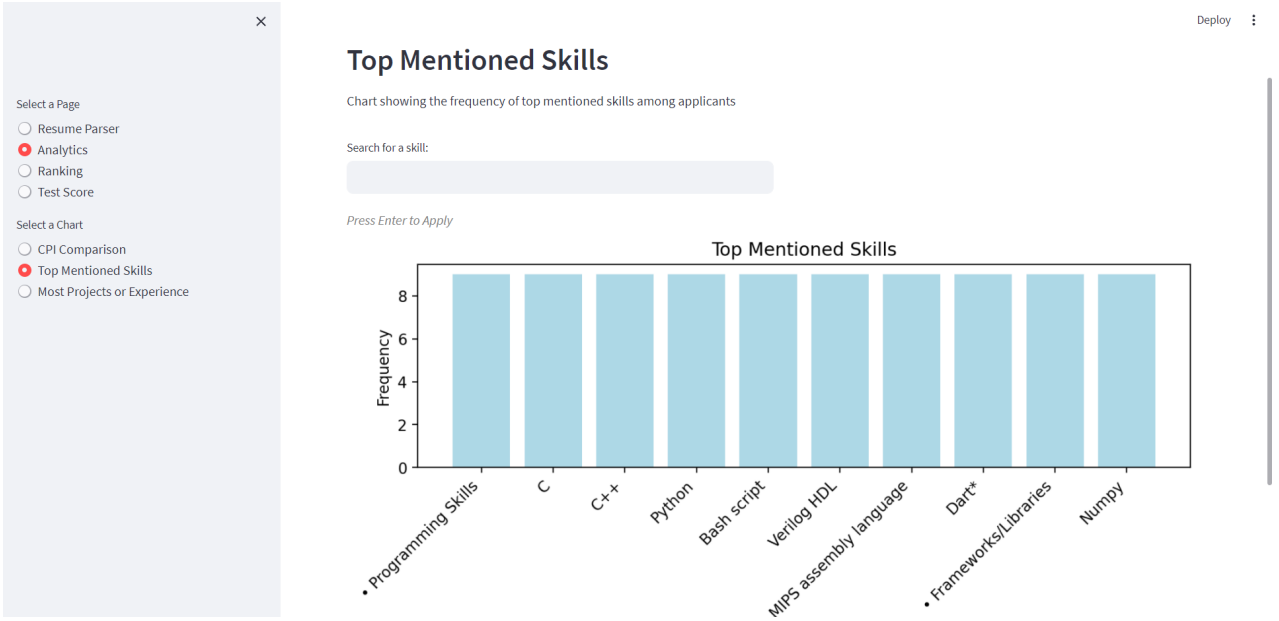
A horizontal bar chart is created using Matplotlib, displaying CPI values for different applicants.

Median and average CPI lines are added to the chart, and labels are provided.

The chart is displayed using `st.pyplot`.



3. Top Mentioned Skills Option:



If the selected analytics option is "Top Mentioned Skills," the code proceeds to create a visualization related to skills.

Users can search for a specific skill using a text input field (`st.text_input`).
The matching applicants who possess the searched skill are filtered from the DataFrame.
If matching applicants are found, their names and skills are displayed in a DataFrame using `st.dataframe`.
If no matching skills are found, a message indicating so is displayed using `st.subheader`.

The code effectively uses Streamlit to create an interactive dashboard that allows users to analyze data related to job applicants. It provides options to visualize and compare applicants' CPI values and to search for applicants based on specific skills. The dashboard is user-friendly and can be accessed through a web browser, providing insights into the applicant data.

This continuation of the code snippet provides the implementation for the remaining two analytics options: "Top Mentioned Skills" and "Most Projects or Experience."
Inside the conditional block for the "Top Mentioned Skills" option, the code counts the frequency of each skill mentioned by all applicants using the `Counter` class from the `collections` module.
The most common skills are extracted using the `most_common` method, and you can adjust the number of top skills displayed by changing the argument inside `most_common`.

The skill names and their frequencies are separated into lists for creating the bar chart.
A bar chart is created using Matplotlib, displaying the frequencies of the top-mentioned skills.
The x-axis labels are rotated, and spacing adjustments are made for better readability.
The smaller bar chart for skill frequencies is displayed using `st.pyplot`.

4. Most Projects or Experience Option:

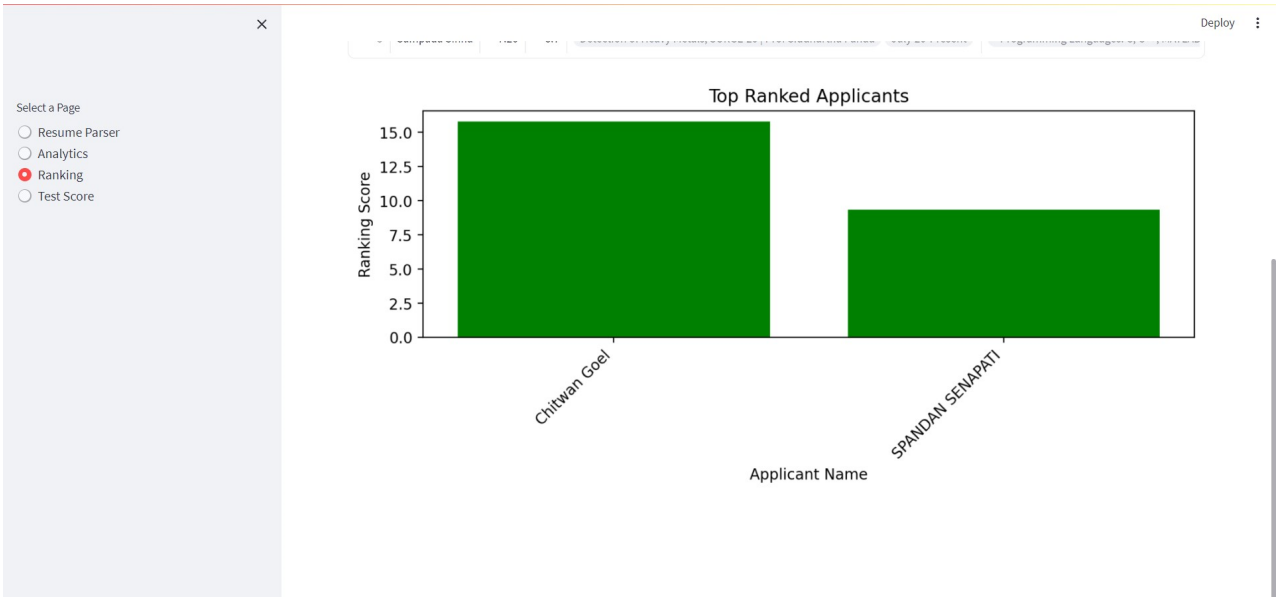
If the selected analytics option is "Most Projects or Experience," the code displays applicants with the most number of projects or experience.
Users can search for specific keywords within applicants' project or experience descriptions using a text input field.

Matching applicants are filtered based on the keyword search using a lambda function and the `apply` method. If matching applicants are found, their names and experience information are displayed using `st.dataframe`.

If no matching applicants are found based on the keyword search, a message indicating so is displayed. If no keyword is provided (i.e., the keyword search input is empty), the code counts the number of projects or experiences for each applicant, sorts the DataFrame by experience in descending order, and creates a bar chart to visualize the applicants with the most projects or experience. This chart is similar to the one created in the "CPI Comparison" option.

Overall, the code handles the "Top Mentioned Skills" and "Most Projects or Experience" options in the analytics dashboard. It provides visualizations and filtering capabilities to help users gain insights into the skills and experience levels of job applicants. The combination of Streamlit, Pandas, and Matplotlib makes it possible to create an interactive and informative user interface for data analysis.

Ranking Page-



The page is implemented with a code that handles the "Ranking of Applicants" section in the analytics dashboard. It calculates a ranking score for each applicant based on certain criteria and then displays the ranked list of applicants along with a bar chart visualization of the top-ranked applicants.

Here's what this section of the code does step by step:

1. Load Data and Create DataFrame:

Similar to previous sections, the code starts by loading the JSON data and creating a DataFrame named `applicants_df` to store applicant information.

2. Calculate Ranking Score:

The ranking score for each applicant is calculated using a combination of different criteria.
- The formula used to calculate the score is:

score = (cpi * 0.5) + (experience_count * 0.1) + (skills_count * 0.2)

where `cpi` is the Cumulative Performance Index, `experience_count` is the count of experiences, and `skills_count` is the count of skills.

3. Add Points Based on Designation Matching:

Additional points are added to the ranking score if the applicant's designation matches any of their skills or experiences.
If the applicant's designation is found in their skills or experiences, the score is increased by 0.3.

4. Ranking Page Display:

The code displays a subheader and a description indicating that the page is for ranking applicants based on the provided criteria.

5. Sort and Display Ranked Applicants:

The applicants are sorted based on their calculated ranking score in descending order.
The ranked applicants' DataFrame is displayed using `st.dataframe` to show columns like name, score, CPI, experience, skills, and designation.

6. Top Ranked Applicants Bar Chart:

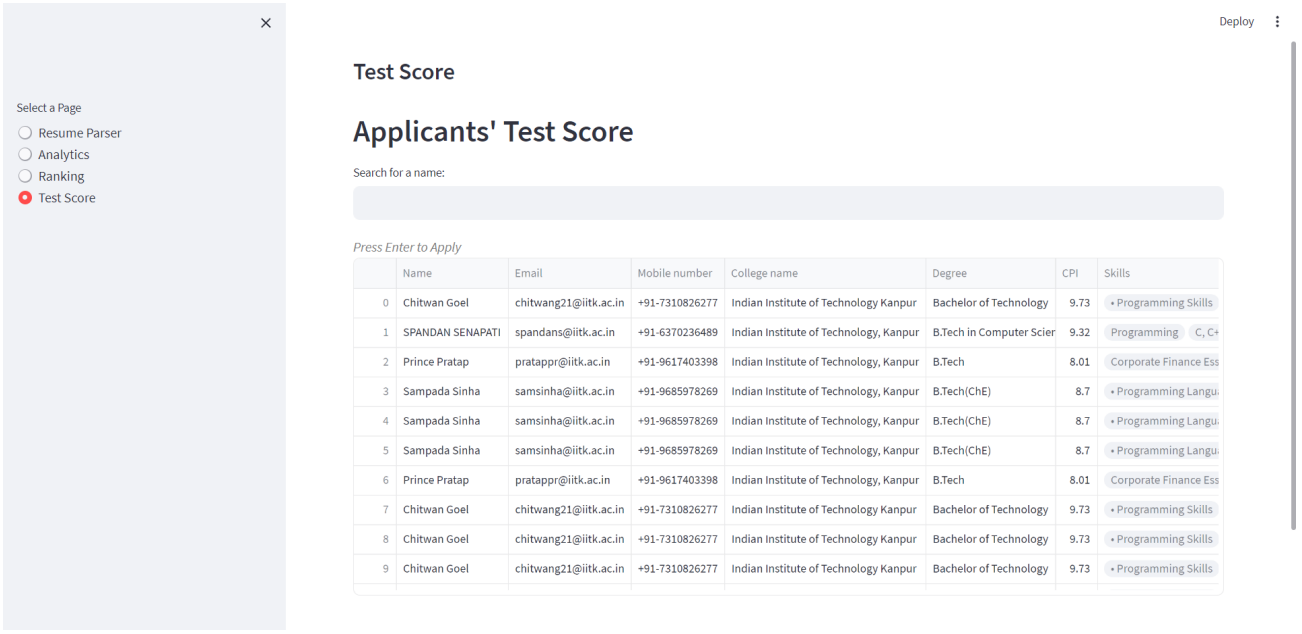
The top-ranked applicants (e.g., top 10) are selected from the ranked DataFrame.
A bar chart is created using Matplotlib to visualize the ranking scores of these top applicants.
Similar to previous charts, the x-axis labels are rotated for better readability.

7. Display Bar Chart:

The bar chart of top-ranked applicants is displayed using `st.pyplot`.

This section of the code adds a ranking functionality to the analytics dashboard. It calculates a composite score for each applicant based on their CPI, experience, skills, and designation and then displays the ranked list of applicants along with a bar chart of the top-ranked applicants. This feature can help recruiters or evaluators quickly identify the most suitable candidates based on the provided criteria.

Test Score Page-



There is an input field that takes the student's name as input and shows the test score.
If a test score is not added, there is an option to add it, and if added, you can also modify it very easily.

Team Members:

- **Chitwan Goel (210295)**
- **Jenil Dobariya (220385)**
- **Yash Chauhan (221217)**
- **Shivansh Mangal (221021)**
- **Aryan Kadam (230221)**
- **Soham Panchal (231014)**
- **Narayan Somaiya (231019)**