Capstone Project Report on

IRINS Researchers Database Analysis and Ranking

submitted in partial fulfillment (8 Course Credits) of the

requirements for the award of the degree of

Masters of Technology

in

Computer Science and Engineering

By:

Dwaipayan Mondal (MT23035)

Parnita Bokade (MT23055)

Pragati Agrawal (MT23059)

Prolay Shankar Mazumder (MT23065)

under the guidance of

Dr.N. Arul Murugan

(Associate Professor, CB)

Department of Computer Science & Engineering

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

Table of Contents

1. ABSTRACT

The primary objective of this project is to develop a comprehensive methodology for gathering and analyzing university ranking data through web scraping techniques. This report outlines the challenges faced and solutions implemented in the process of scraping data from the IRINS database to rank universities, departments, and professors based on various metrics such as H-index, total citations, and total publications. Utilizing Python programming and web scraping tools like BeautifulSoup and Selenium, the project successfully extracted and analyzed data from numerous institutions.

The initial phase addressed issues such as identical links for different metrics, pagination complexities, and dynamic content loading. Solutions involved detailed observational analysis, integration of Selenium for dynamic content handling, and ongoing efforts to manage extensive pagination. Subsequent phases focused on refining data extraction, eliminating extraneous information, and overcoming obstacles related to large-scale data scraping and multi-threading.

The project culminated in the development of ranking systems for departments and colleges, showcasing the potential of web scraping and data analysis in the field of higher education. This work provides valuable insights into the trends and performance metrics of academic institutions, contributing significantly to the domain of educational data analysis.

2. MOTIVATION

a) **Importance of University Rankings**: Address the critical role university rankings play in shaping perceptions, guiding student choices, and influencing institutional policies.

b) Limitations of Traditional Methods: Overcome the limitations of manual data collection and static datasets in traditional ranking methodologies.

c) Advanced Web Scraping Techniques: Utilize Python programming and web scraping tools like BeautifulSoup and Selenium to create a dynamic, comprehensive, and scalable approach to data collection.

d) Efficiency and Accuracy: Achieve more accurate, up-to-date, and detailed insights into academic performance by automating data extraction processes.

e) Challenge Resolution: Tackle specific web scraping challenges such as handling pagination, managing dynamic content loading, and overcoming restrictions on large-scale data extraction.

f) Knowledge Contribution: Enhance the understanding and implementation of robust web scraping methodologies through the development of effective solutions.

g) Informing Decision-Making: Provide valuable insights to inform decision-making processes for students, educators, and policymakers in the field of higher education.

h) Advancement of Higher Education: Demonstrate the transformative potential of web scraping and data analysis in generating and interpreting university rankings, contributing to the advancement of higher education.

i) Leveraging Internet Data: Exploit the growing availability and accessibility of online data for gathering and analyzing university rankings.

3. RELEVANT TOOLS

a) **Python**:

A versatile and powerful programming language, widely used in data science and web development due to its readability, extensive libraries, and active community.

Python serves as the backbone of this project, enabling seamless integration of various libraries and tools necessary for web scraping and data analysis.

b) Cloudscraper

A library designed to bypass anti-bot measures on websites by mimicking a real web browser with randomized user-agent strings.

Cloudscraper is used to create scraper instances capable of accessing web pages that implement anti-bot technologies, ensuring uninterrupted data extraction.

c) BeautifulSoup

A Python library for parsing HTML and XML documents, allowing for easy navigation, searching, and modification of the parse tree.

BeautifulSoup is employed to parse the HTML content fetched from the web pages, extracting specific data points such as names, Vidwan IDs, departments, colleges, and academic metrics.

Installation Command:

```
pip install beautifulsoup4
```

Dependencies:

requests: Often used alongside BeautifulSoup for making HTTP requests.

```
pip install requests
```

d) Selenium WebDriver:

A tool for automating web browser interaction, enabling the simulation of user actions such as clicking, form submission, and navigation through web pages.

Selenium WebDriver is crucial for handling dynamic web content, navigating through pages, interacting with elements like pagination links and forms, and ensuring that the content is fully loaded before extraction.

e) Chromedriver_autoinstaller:

A utility that automatically downloads and installs the correct version of ChromeDriver, which is necessary for Selenium WebDriver to control Chrome. This tool ensures compatibility between the Chrome browser and Selenium WebDriver, streamlining the setup process and avoiding version mismatches that could disrupt automation.

f) Pandas:

A powerful data manipulation and analysis library for Python, providing data structures like DataFrames to manage and analyze data efficiently.

Pandas is used to store the scraped data in a structured format, facilitating easy manipulation, analysis, and exportation to CSV files for further use and reporting.

4. WEB SCRAPING

Definition: Web scraping involves extracting specific data from web pages. It targets particular information within a website and collects it for further processing or analysis. Web scraping is a powerful technique used to extract information from websites. This project employs web scraping to gather detailed data about scientists and their academic metrics from the IRINS database.

Purpose: The main goal is to gather data from a particular website or web pages, often to convert it into a structured format like a spreadsheet or database.

Scope: Typically limited to a single website or a set of predefined web pages.

Example: The provided code scrapes detailed information (like names, Vidwan IDs, departments, colleges, total citations, h-index, and journal articles) from specific pages of the IRINS website.

4.1 Web Scrapper Operation:

The process begins by loading the target web page using Selenium, which automates browser interactions, allowing for navigation through dynamic content and handling elements like pagination and form submissions. Cloudscraper and BeautifulSoup work together to fetch the page content and parse it efficiently, identifying and extracting relevant data points such as names, Vidwan IDs, departments, colleges, total citations, h-index, and journal articles. Randomized delays and user-agent strings are implemented to mimic human browsing behavior and avoid detection by anti-scraping measures. This methodical approach ensures comprehensive and accurate data collection, which is then organized into a structured format using Pandas for subsequent analysis and reporting. Web scraping, as demonstrated in this project, is a versatile and essential tool for collecting large-scale data from web sources, enabling detailed insights and analysis in various fields.

Setup and Configuration:

Python Libraries: Import necessary libraries including `cloudscraper`, `BeautifulSoup`, `pandas`, `time`, `chromedriver_autoinstaller`, `selenium`, and `random`.

Cloudscraper Instance: Create a cloudscraper instance with a randomized user-agent to bypass anti-bot measures.

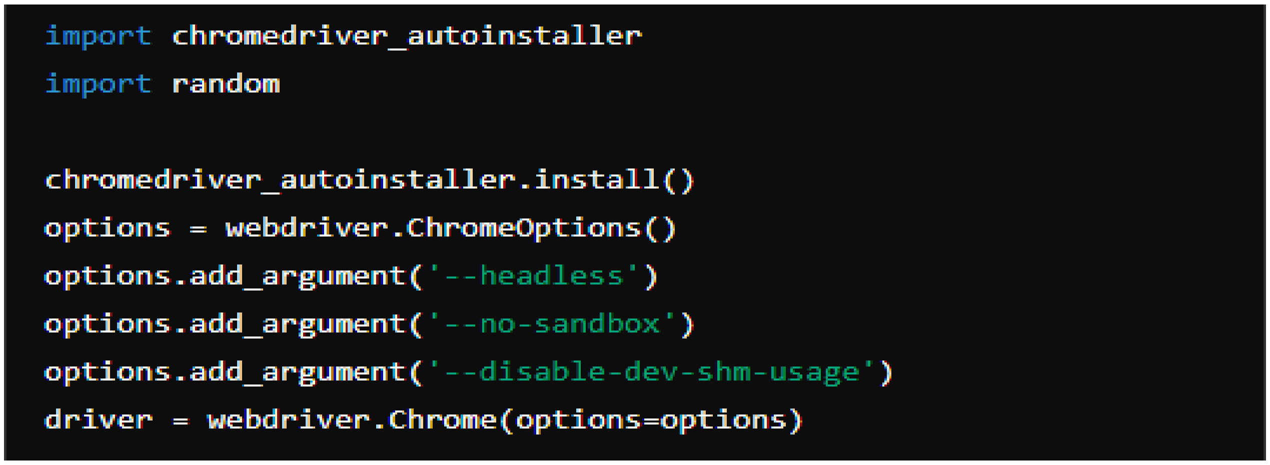
Chromedriver Installation: Automatically install the correct version of ChromeDriver using `chromedriver_autoinstaller`.

```
import cloudscraper
scraper = cloudscraper.create_scraper(browser='chrome')
```

WebDriver Initialization:

Selenium WebDriver: Configure Selenium WebDriver with ChromeOptions to run in headless mode and use a randomized user-agent to mimic real user behavior.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
```

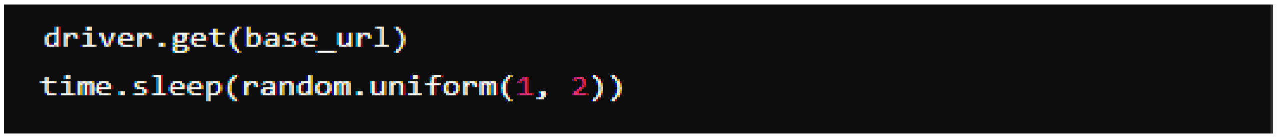


```python
import chromedriver_autoinstaller
import random

chromedriver_autoinstaller.install()
options = webdriver.ChromeOptions()
options.add_argument('--headless')
options.add_argument('--no-sandbox')
options.add_argument('--disable-dev-shm-usage')
driver = webdriver.Chrome(options=options)
```

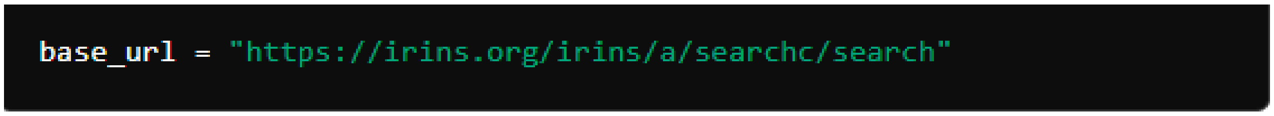
Navigating to the Target Page:

Load Initial URL: We open the base URL of the IRINS search page using Selenium.

Wait for Page Load: Implemented a random delay to ensure the page is fully loaded.



```python
driver.get(base_url)
time.sleep(random.uniform(1, 2))
```



```python
base_url = "https://irins.org/irins/a/searchc/search"
```

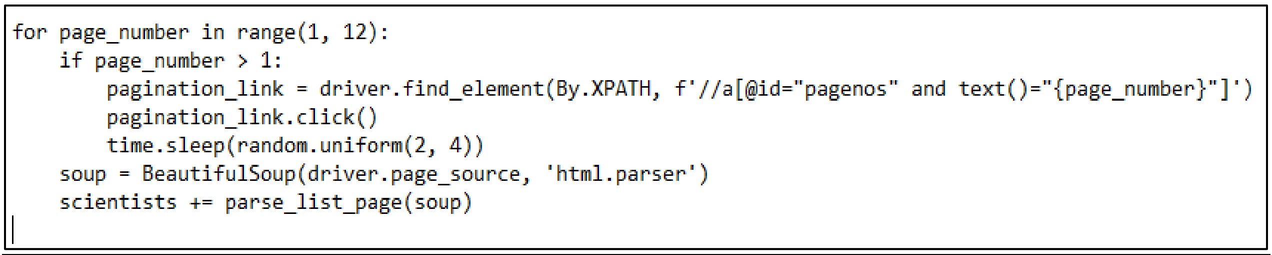
Performing Search:

Select Search Type: We choose the "Exact Search" radio button.

Enter Search Query: Input the search query (e.g., "central university") into the search field and submit the search.

Handling Pagination and Applying Filters:

JavaScript Execution: Use Selenium to execute JavaScript for applying specific filters (e.g., filtering by college name).



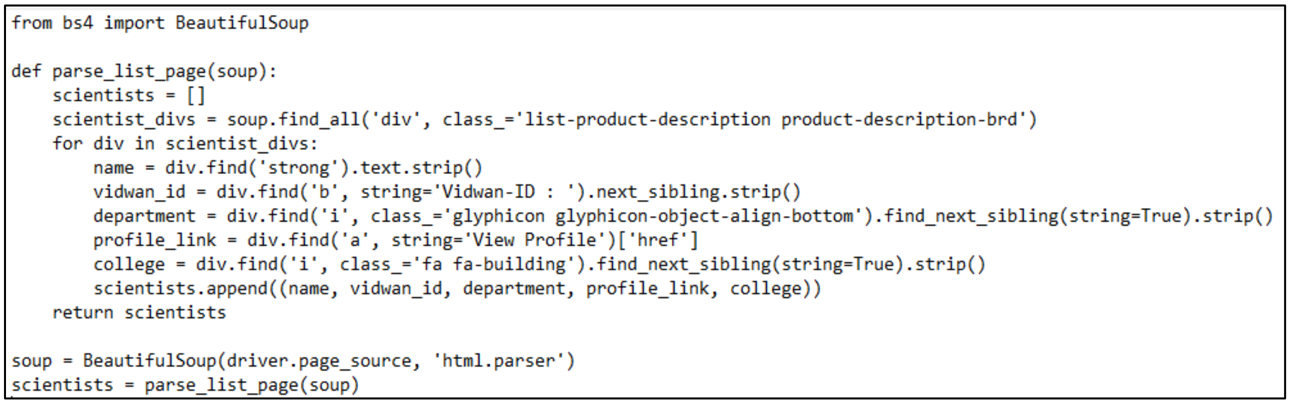
```
for page_number in range(1, 12):
    if page_number > 1:
        pagination_link = driver.find_element(By.XPATH, f'//a[@id="pagenos" and text()="{page_number}"]')
        pagination_link.click()
        time.sleep(random.uniform(2, 4))
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    scientists += parse_list_page(soup)
```

Data Extraction:

Pagination Handling: Loop through multiple pages of search results to scrape data from each page.

Parse List Page: Use BeautifulSoup to parse the HTML content of the current page and extract basic information about scientists (e.g., name, Vidwan ID, department, profile link, college).

Profile Page Parsing: For each scientist, navigate to their profile page and extract additional details like total citations, h-index, and journal articles.

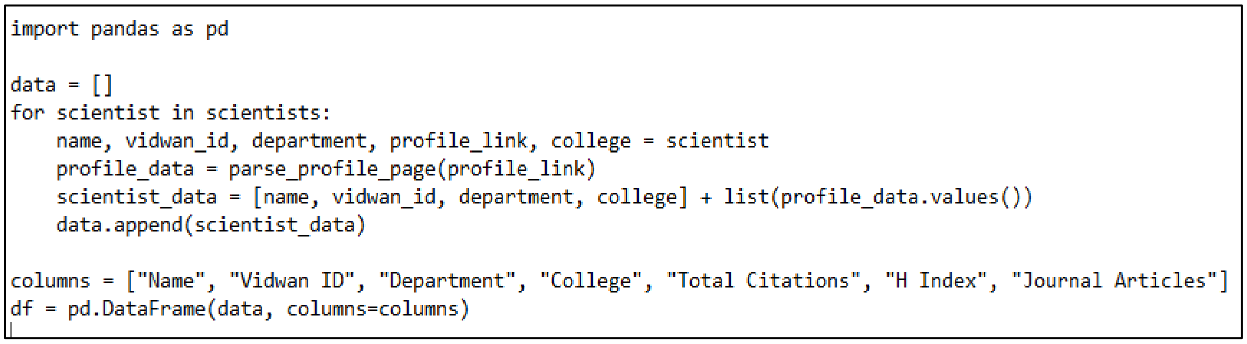


```
from bs4 import BeautifulSoup

def parse_list_page(soup):
    scientists = []
    scientist_divs = soup.find_all('div', class_='list-product-description product-description-brd')
    for div in scientist_divs:
        name = div.find('strong').text.strip()
        vidwan_id = div.find('b', string='Vidwan-ID : ').next_sibling.strip()
        department = div.find('i', class_='glyphicon glyphicon-object-align-bottom').find_next_sibling(string=True).strip()
        profile_link = div.find('a', string='View Profile')['href']
        college = div.find('i', class_='fa fa-building').find_next_sibling(string=True).strip()
        scientists.append((name, vidwan_id, department, profile_link, college))
    return scientists

soup = BeautifulSoup(driver.page_source, 'html.parser')
scientists = parse_list_page(soup)
```

Data Storage:

Data Aggregation: Append the extracted data for each scientist to a list.

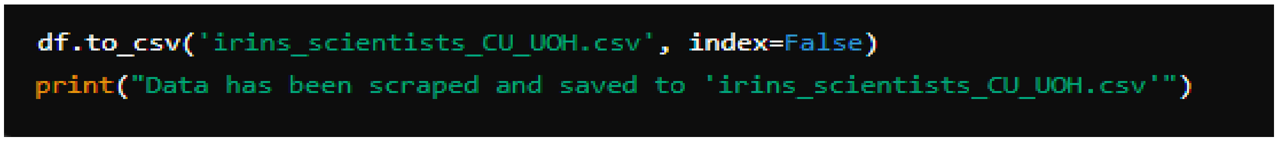
DataFrame Creation: Convert the list of data into a pandas DataFrame for structured data storage.



```
import pandas as pd

data = []
for scientist in scientists:
    name, vidwan_id, department, profile_link, college = scientist
    profile_data = parse_profile_page(profile_link)
    scientist_data = [name, vidwan_id, department, college] + list(profile_data.values())
    data.append(scientist_data)

columns = ["Name", "Vidwan ID", "Department", "College", "Total Citations", "H Index", "Journal Articles"]
df = pd.DataFrame(data, columns=columns)
```

Saving the Data:

CSV Export: Save the DataFrame to a CSV file (`irins_scientists_CU_UOH.csv`) for further analysis and reporting.



```
df.to_csv('irins_scientists_CU_UOH.csv', index=False)
print("Data has been scraped and saved to 'irins_scientists_CU_UOH.csv'")
```

We introduced random delays and user-agent strings to mimic human behavior and avoid detection by anti-scraping mechanisms. We also used Selenium to interact with dynamically loaded content and JavaScript-driven elements along with organization of extracted data in a pandas DataFrame and export it to a CSV file for easy analysis.

5. DATA ANALYSIS:

Loading and Cleaning the Data

a) Importing the Required Library: The script uses the `pandas` library for data manipulation and analysis.

b) Loading the Excel File: The script loads a csv file named `IRINS_professor_Database.csv` located at `/content/`.

c) Removing Extra Spaces in the 'Name' Column: The script removes any extra spaces from the 'Name' column using regex to replace multiple spaces with a single space and then trims any leading or trailing spaces.

d) Saving the Cleaned Data to a CSV File: The cleaned DataFrame with the total scores is saved to a new CSV file named IRINS_professor_Database_cleaned.csv.

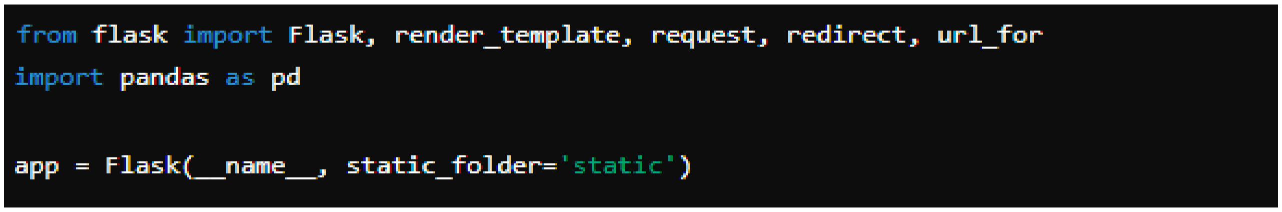
6. WEB APPLICATION FOR UNIVERSITY AND PROFESSOR RANKING

Here we have used Flask web application which serves as a platform to display and rank professors, colleges, and departments based on various academic criteria. The data is loaded from CSV files, processed, and then displayed through HTML templates. Users can interact with the application to view rankings and search for specific department ranking.

Components of the Application

a) Importing Libraries and Initializing the Flask App

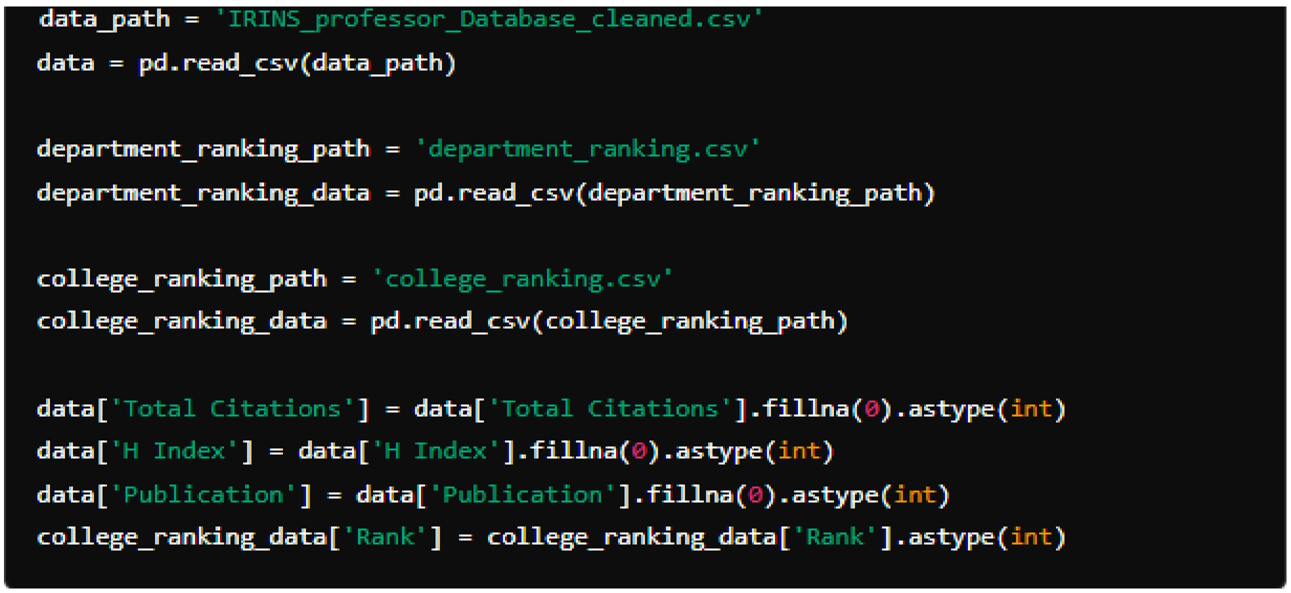
- Import necessary libraries.
- Initialize the Flask application.
- Create a folder ‘static’ and initialize it to store css style sheets.



```
from flask import Flask, render_template, request, redirect, url_for
import pandas as pd

app = Flask(__name__, static_folder='static')
```

b) Loading and Cleaning Data

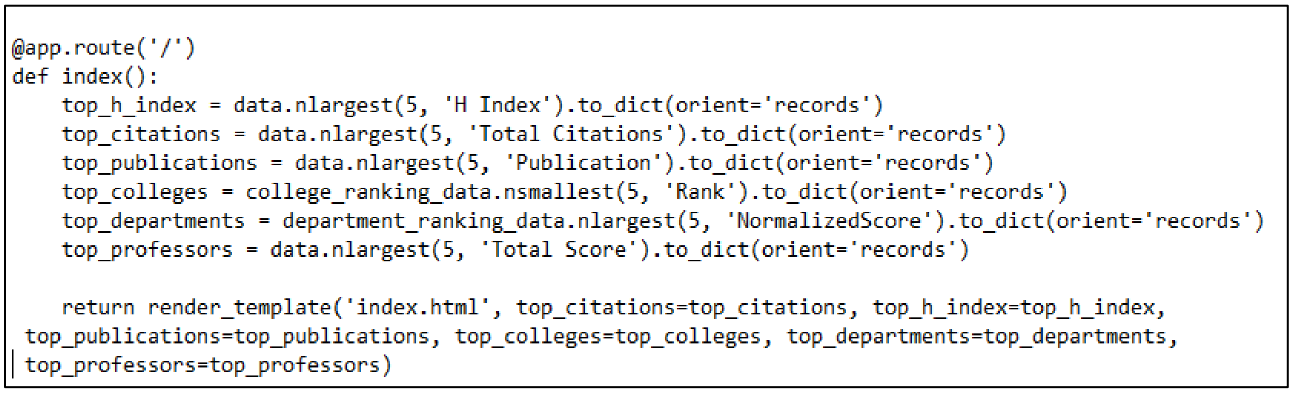
- Load professor, department ranking, and college ranking data from CSV files.
- Clean the data by filling missing values and converting data types.



```
data_path = 'IRINS_professor_Database_cleaned.csv'
data = pd.read_csv(data_path)

department_ranking_path = 'department_ranking.csv'
department_ranking_data = pd.read_csv(department_ranking_path)

college_ranking_path = 'college_ranking.csv'
college_ranking_data = pd.read_csv(college_ranking_path)

data['Total Citations'] = data['Total Citations'].fillna(0).astype(int)
data['H Index'] = data['H Index'].fillna(0).astype(int)
data['Publication'] = data['Publication'].fillna(0).astype(int)
college_ranking_data['Rank'] = college_ranking_data['Rank'].astype(int)
```

c) Index Route

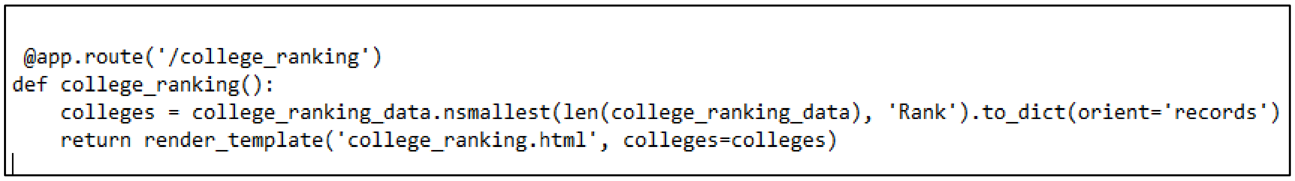
- Display the top 5 professors, colleges, and departments based on various criteria.



```
@app.route('/')
def index():
    top_h_index = data.nlargest(5, 'H Index').to_dict(orient='records')
    top_citations = data.nlargest(5, 'Total Citations').to_dict(orient='records')
    top_publications = data.nlargest(5, 'Publication').to_dict(orient='records')
    top_colleges = college_ranking_data.nsmallest(5, 'Rank').to_dict(orient='records')
    top_departments = department_ranking_data.nlargest(5, 'NormalizedScore').to_dict(orient='records')
    top_professors = data.nlargest(5, 'Total Score').to_dict(orient='records')

    return render_template('index.html', top_citations=top_citations, top_h_index=top_h_index,
 top_publications=top_publications, top_colleges=top_colleges, top_departments=top_departments,
 top_professors=top_professors)
```

d) College Ranking Route

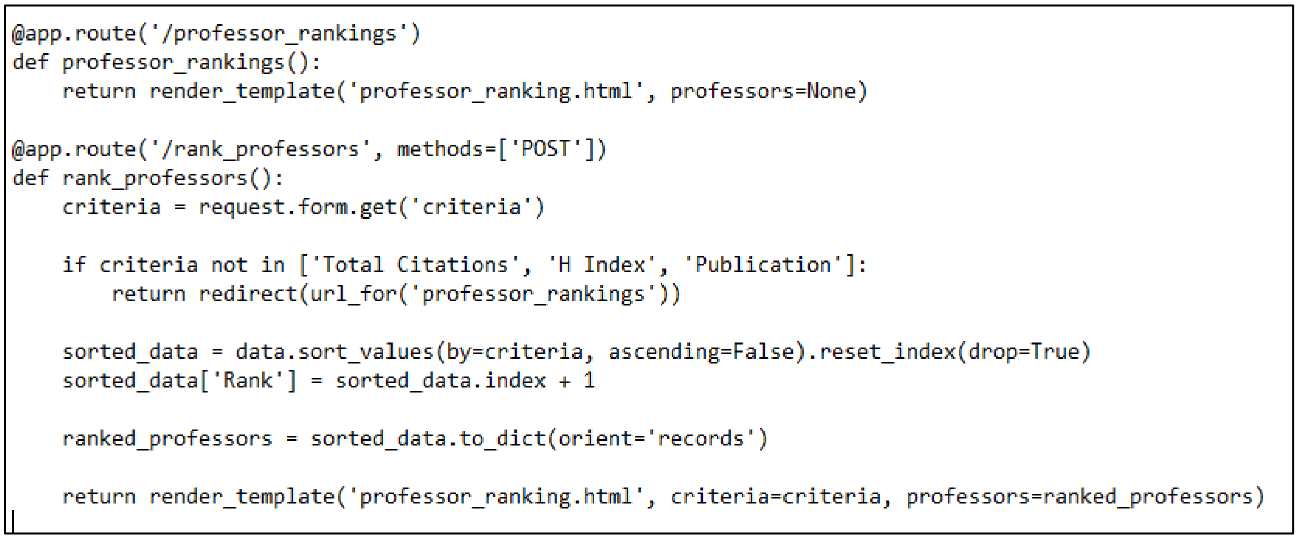
- Display the ranking of all colleges.



```
 @app.route('/college_ranking')
def college_ranking():
    colleges = college_ranking_data.nsmallest(len(college_ranking_data), 'Rank').to_dict(orient='records')
    return render_template('college_ranking.html', colleges=colleges)
```

e) Professor Ranking Routes

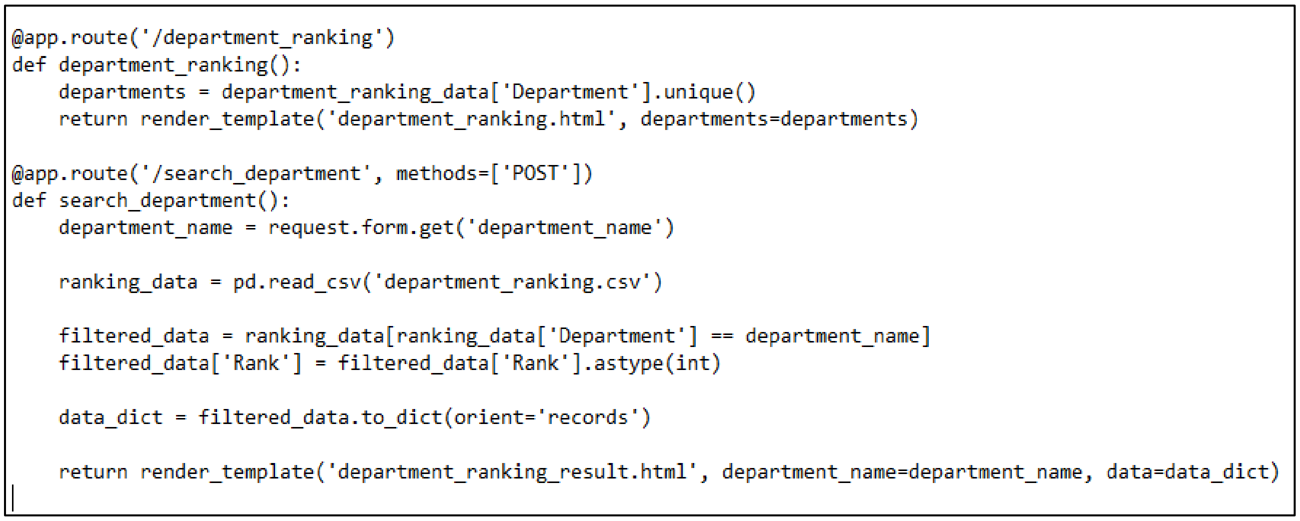
- Display a form for users to choose ranking criteria.
- Process the form submission and display the ranked professors based on the selected criteria.



```python
@app.route('/professor_rankings')
def professor_rankings():
    return render_template('professor_ranking.html', professors=None)

@app.route('/rank_professors', methods=['POST'])
def rank_professors():
    criteria = request.form.get('criteria')

    if criteria not in ['Total Citations', 'H Index', 'Publication']:
        return redirect(url_for('professor_rankings'))

    sorted_data = data.sort_values(by=criteria, ascending=False).reset_index(drop=True)
    sorted_data['Rank'] = sorted_data.index + 1

    ranked_professors = sorted_data.to_dict(orient='records')

    return render_template('professor_ranking.html', criteria=criteria, professors=ranked_professors)
```

f) Department Ranking Routes

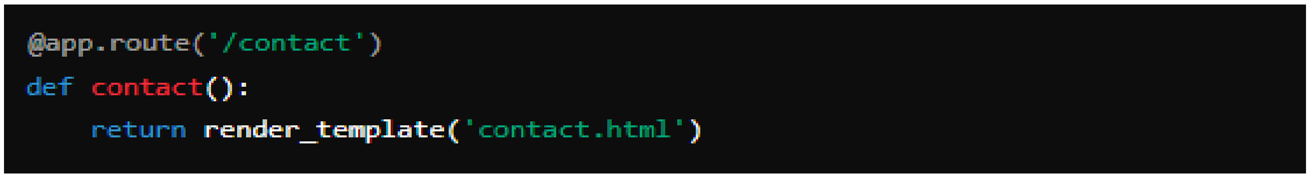
- Display a form for users to select a department.
- Process the form submission and display the ranking of the selected department.



```python
@app.route('/department_ranking')
def department_ranking():
    departments = department_ranking_data['Department'].unique()
    return render_template('department_ranking.html', departments=departments)

@app.route('/search_department', methods=['POST'])
def search_department():
    department_name = request.form.get('department_name')

    ranking_data = pd.read_csv('department_ranking.csv')

    filtered_data = ranking_data[ranking_data['Department'] == department_name]
    filtered_data['Rank'] = filtered_data['Rank'].astype(int)

    data_dict = filtered_data.to_dict(orient='records')

    return render_template('department_ranking_result.html', department_name=department_name, data=data_dict)
```

g) Contact Route

- Display a contact page.



```python
@app.route('/contact')
def contact():
    return render_template('contact.html')
```

h) Running the Application

- Start the Flask web application.

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

i) File Structure of Flask

```
flask_project/
|
├── app.py
├── requirements.txt
├── static/
|   ├── css/
|   |   └── styles.css
|   ├── images/
|   |   ├── logo.png
|   |   └── other_image.jpg
|   └── js/
|       └── scripts.js
└── templates/
    ├── base.html
    ├── index.html
    ├── professor_ranking.html
    ├── college_ranking.html
    └── department_ranking.html
```

Data Cleaning and Processing

Removing Extra Spaces:

```
df['Name'] = df['Name'].str.replace(r'\s+', ' ', regex=True).str.strip()
```

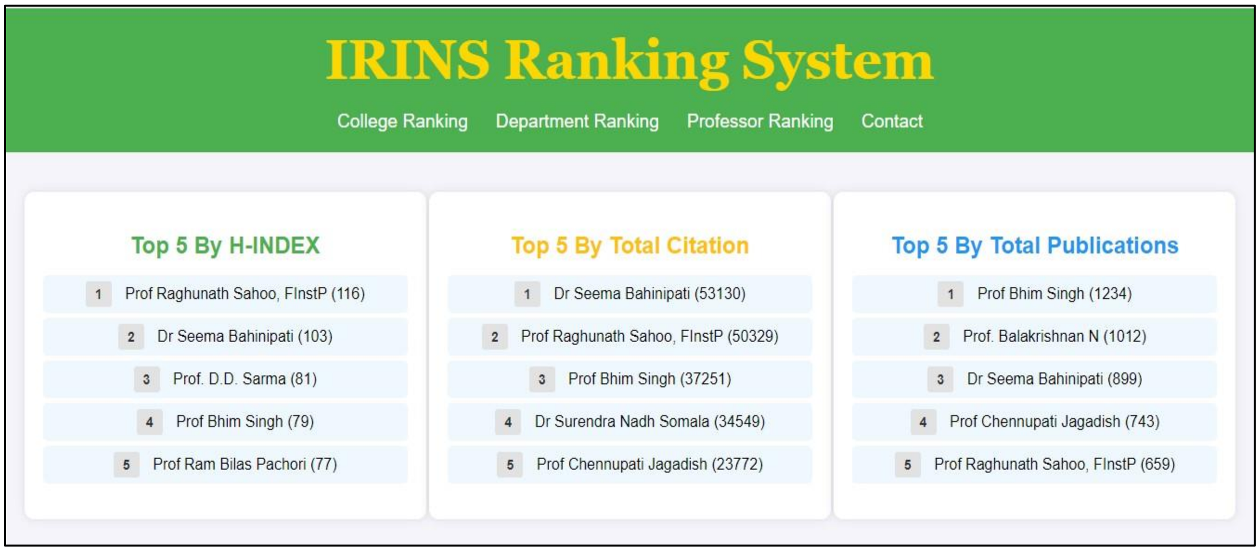
Calculating Total Score:

```
def calculate_total_score(row):
    total_score = (row['Total Citations'] + row['H Index'] + row['Publication']) / 3
    return total_score

df['Total Score'] = df.apply(calculate_total_score, axis=1)
```

Saving the Cleaned Data:

```python
cleaned_file_path = 'IRINS_professor_Database_cleaned.csv'
df.to_csv(cleaned_file_path, index=False)
```

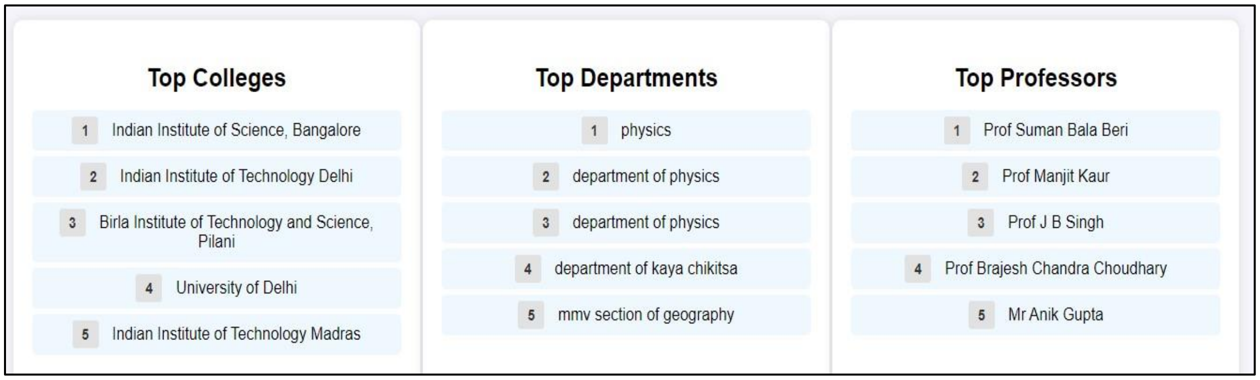
This Flask web application provides an interactive platform to view and rank professors, colleges, and departments based on various academic criteria. It includes data loading, cleaning, ranking, and rendering functionalities. The user can interact with the application to view the top performers and search for specific department rankings. The code ensures data integrity and efficient rendering of the information to the end-users.



IRINS Ranking System

College Ranking | Department Ranking | Professor Ranking | Contact

Top 5 By H-INDEX	Top 5 By Total Citation	Top 5 By Total Publications
1 Prof Raghunath Sahoo, FInstP (116)	1 Dr Seema Bahinipati (53130)	1 Prof Bhim Singh (1234)
2 Dr Seema Bahinipati (103)	2 Prof Raghunath Sahoo, FInstP (50329)	2 Prof. Balakrishnan N (1012)
3 Prof. D.D. Sarma (81)	3 Prof Bhim Singh (37251)	3 Dr Seema Bahinipati (899)
4 Prof Bhim Singh (79)	4 Dr Surendra Nadh Somala (34549)	4 Prof Chennupati Jagadish (743)
5 Prof Ram Bilas Pachori (77)	5 Prof Chennupati Jagadish (23772)	5 Prof Raghunath Sahoo, FInstP (659)



Top Colleges	Top Departments	Top Professors
1 Indian Institute of Science, Bangalore	1 physics	1 Prof Suman Bala Beri
2 Indian Institute of Technology Delhi	2 department of physics	2 Prof Manjit Kaur
3 Birla Institute of Technology and Science, Pilani	3 department of physics	3 Prof J B Singh
4 University of Delhi	4 department of kaya chikitsa	4 Prof Brajesh Chandra Choudhary
5 Indian Institute of Technology Madras	5 mmv section of geography	5 Mr Anik Gupta

Fig 1: IRINS Ranking System Homepage

7. RANKING OF PROFESSORS BY CRITERIA

Users can rank professors by Total Citations, H Index, or Publications. The application validates the input, sorts the data accordingly, and renders the results in a web template.

Step 1: Get User Input for Ranking Criteria: The first step involves retrieving the user-selected ranking criteria from the form submission.

```
criteria = request.form.get('criteria')
```

Step 2: Validate Criteria: The input criteria are validated to ensure they are one of the allowed options: 'Total Citations', 'H Index', or 'Publication'.

```
if criteria not in ['Total Citations', 'H Index', 'Publication']:
    return redirect(url_for('professor_rankings'))
```

Step 3: Sort Data Based on Selected Criteria: The professor data is sorted in descending order based on the selected criteria. The `reset_index` method is used to re-index the sorted DataFrame.

```
sorted_data = data.sort_values(by=criteria, ascending=False).reset_index(drop=True)
```

Step 4: Add Rank Column: A new column, 'Rank', is added to the DataFrame. The rank is assigned based on the sorted index, starting from 1.

```
sorted_data['Rank'] = sorted_data.index + 1
```

Step 5: Convert to Dictionary for Template: The sorted and ranked DataFrame is converted to a dictionary format, which is suitable for rendering in the HTML template.

```
ranked_professors = sorted_data.to_dict(orient='records')
```

Step 6: Render Template with Ranked Data: The Flask **`render_template`** function is used to render the HTML template with the ranked professor data and the selected criteria.

```
return render_template('professor_ranking.html', criteria=criteria, professors=ranked_prof
```

The **`rank_professors`** function enables users to rank professors based on specific academic metrics. By validating the input, sorting the data, adding rank information, and rendering the results, the function provides an interactive and user-friendly way to compare professors based on their academic performance. This process is crucial for highlighting top-performing professors and facilitating data-driven decisions in an academic setting.

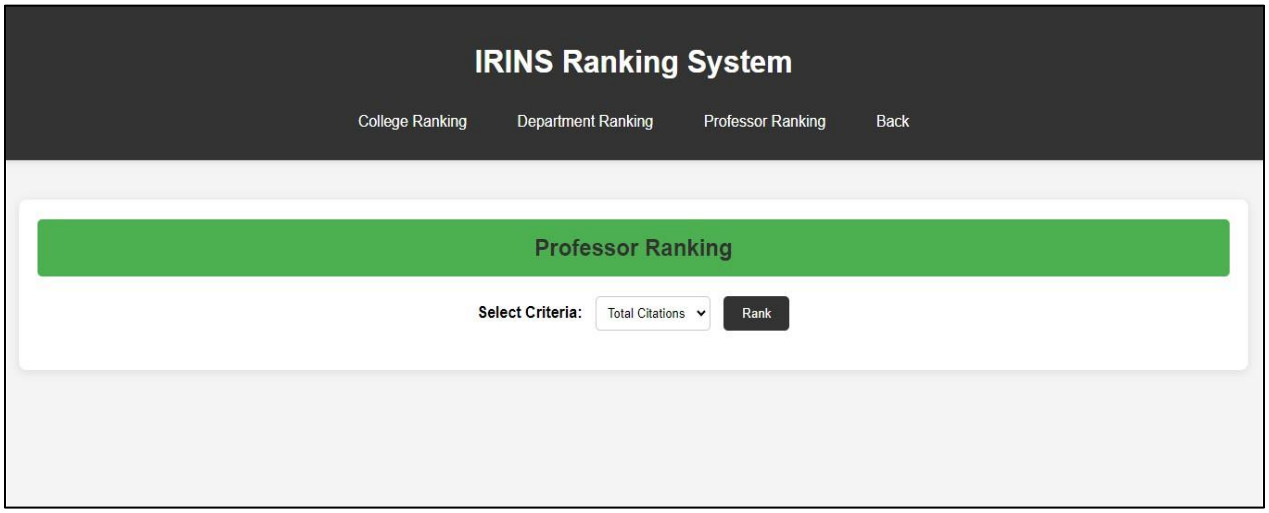
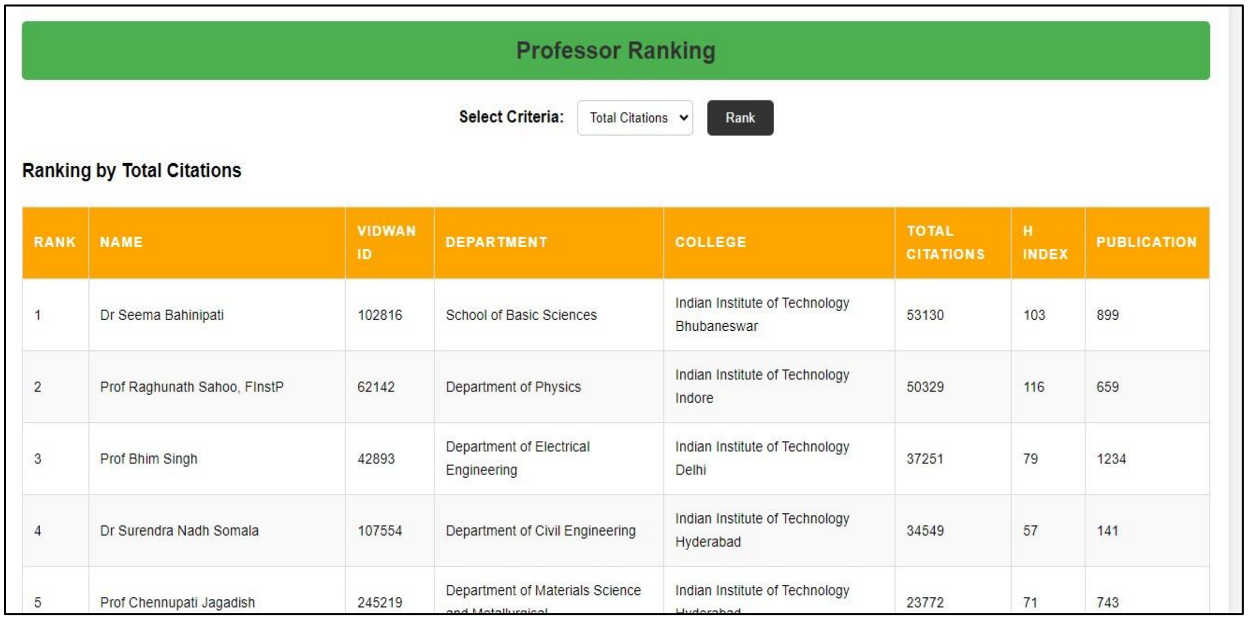


Fig 2: Professor Ranking Page


Professor Ranking

Select Criteria: Total Citations Rank

Ranking by Total Citations

RANK	NAME	VIDWAN ID	DEPARTMENT	COLLEGE	TOTAL CITATIONS	H INDEX	PUBLICATION
1	Dr Seema Bahinipati	102816	School of Basic Sciences	Indian Institute of Technology Bhubaneswar	53130	103	899
2	Prof Raghunath Sahoo, FInstP	62142	Department of Physics	Indian Institute of Technology Indore	50329	116	659
3	Prof Bhim Singh	42893	Department of Electrical Engineering	Indian Institute of Technology Delhi	37251	79	1234
4	Dr Surendra Nadh Somala	107554	Department of Civil Engineering	Indian Institute of Technology Hyderabad	34549	57	141
5	Prof Chennupati Jagadish	245219	Department of Materials Science	Indian Institute of Technology	23772	71	743

Fig 3: Professor Ranking by Total Citations

8. DEPARTMENT RANKING

The objective is to create a normalized ranking of departments within colleges, considering academic contributions measured through metrics such as H Index, Total Citations, and Publications.

Load the Data: The first step involves loading the cleaned dataset containing professor data from a CSV file.

```
file_path = 'IRINS_professor_Database_cleaned.csv'
data = pd.read_csv(file_path)
```

Case Sensitivity Removal of Department Names: To ensure case-insensitive grouping, department names are converted to lowercase.

```
data['Department'] = data['Department'].str.lower()
```

Group by College and Department: The data is grouped by **`College`** and **`Department`**, aggregating metrics such as the sum of H Index, Total Citations, Publications, and the count of professors.

```
grouped = data.groupby(['College', 'Department']).agg({
    'H Index': 'sum',
    'Total Citations': 'sum',
    'Publication': 'sum',
    'Name': 'size'  # Assuming 'Name' column represents the count of professors
}).reset_index()

grouped.rename(columns={'Name': 'ProfCount'}, inplace=True)
```

Calculate Total Score and Normalize: A total score is calculated for each department by summing the aggregated metrics. This score is then normalized by dividing by the number of professors.

```
grouped['TotalScore'] = grouped['H Index'] + grouped['Total Citations'] + grouped['Publicati
grouped['NormalizedScore'] = grouped['TotalScore'] / grouped['ProfCount']
```

Filter Out Colleges with Zero Scores: Departments with a total score of zero are filtered out to maintain meaningful rankings.

```
grouped = grouped[grouped['TotalScore'] > 0]
```

Rank the Departments Based on Normalized Score: Departments are ranked based on their normalized score within each department. The ranking is done in descending order, with the highest score receiving the top rank.

```
grouped['Rank'] = grouped.groupby('Department')['NormalizedScore'].rank(ascending=False, method='min')
```

Sort by Rank: The final dataset is sorted by rank, and only the relevant columns are selected and reordered.

```
grouped = grouped.sort_values(['Rank']).reset_index(drop=True)
final_df = grouped[['Rank', 'College', 'Department', 'NormalizedScore']]
```

Save the Final Ranking to a CSV File: The resulting rankings are saved to a new CSV file for further analysis and reporting.

```
output_file = 'department_ranking.csv'
final_df.to_csv(output_file, index=False)

print(f"Ranking has been saved to {output_file}")
```

It provides a comprehensive approach to ranking departments within colleges based on normalized academic metrics of professors. The steps ensure that the data is cleaned, aggregated, and normalized appropriately to reflect the relative performance of departments in a fair and meaningful way. The resulting CSV file can be used for various analytical and reporting purposes.

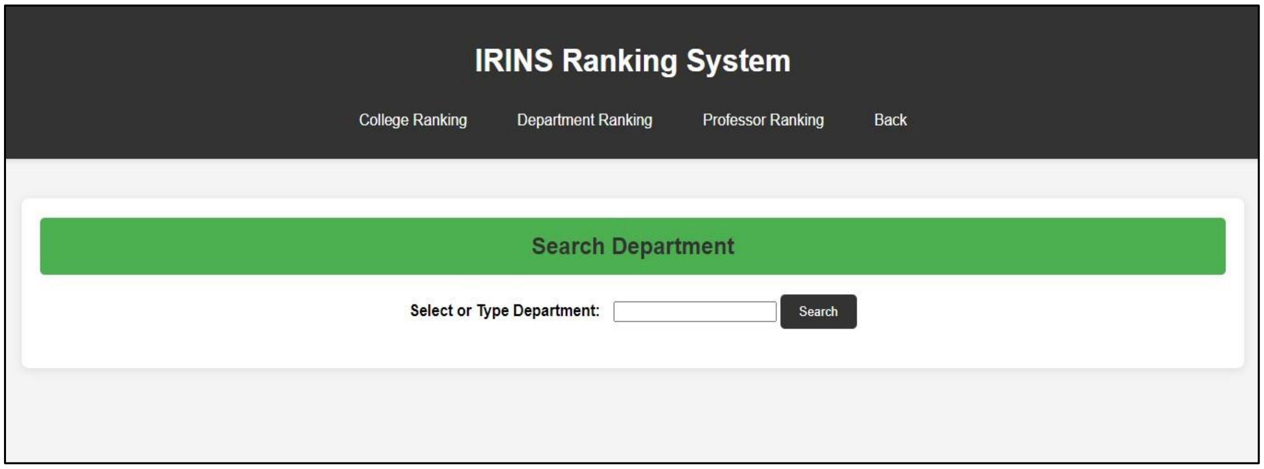
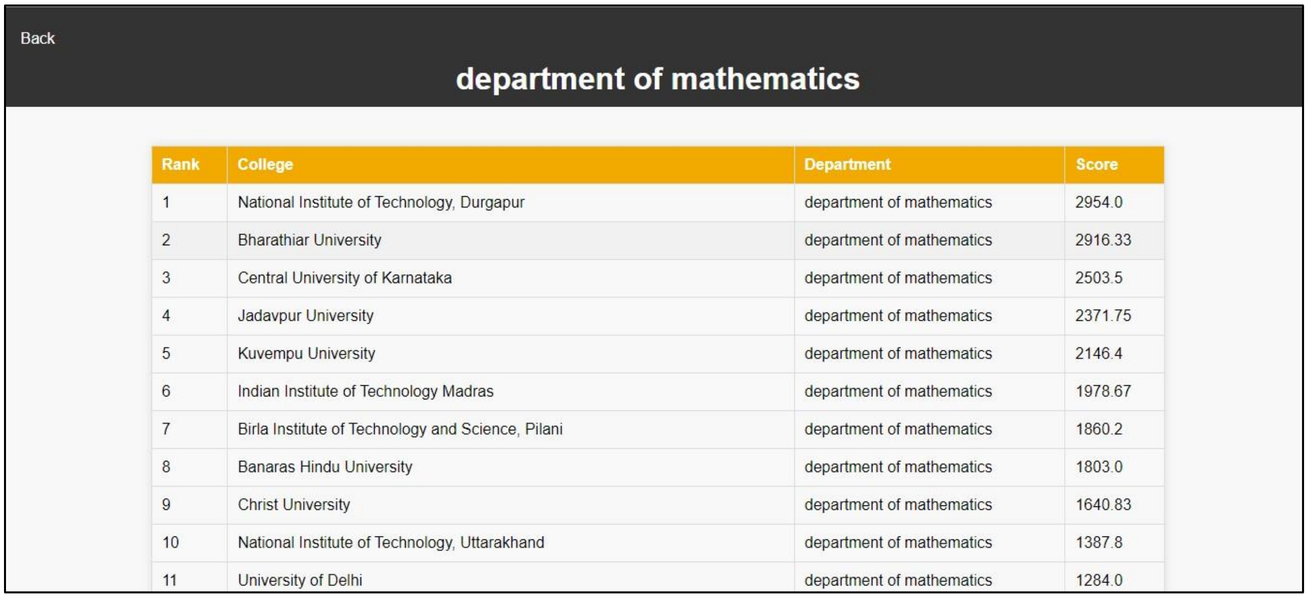


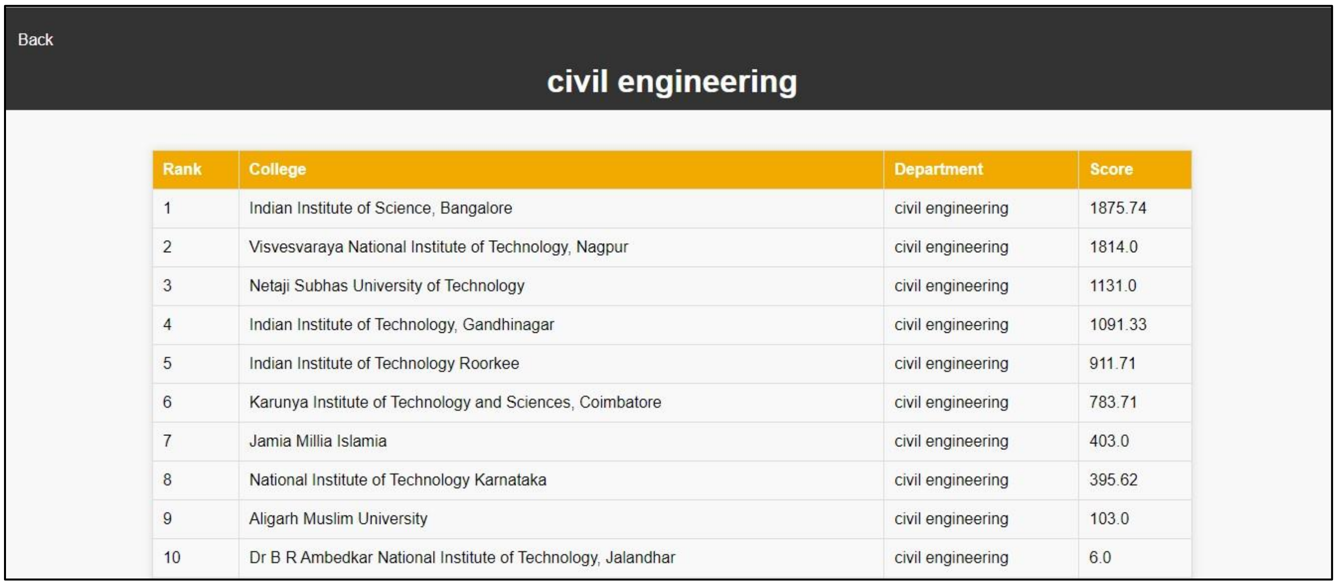
Fig 4: Department Selection Page


Back

department of mathematics

Rank	College	Department	Score
1	National Institute of Technology, Durgapur	department of mathematics	2954.0
2	Bharathiar University	department of mathematics	2916.33
3	Central University of Karnataka	department of mathematics	2503.5
4	Jadavpur University	department of mathematics	2371.75
5	Kuvempu University	department of mathematics	2146.4
6	Indian Institute of Technology Madras	department of mathematics	1978.67
7	Birla Institute of Technology and Science, Pilani	department of mathematics	1860.2
8	Banaras Hindu University	department of mathematics	1803.0
9	Christ University	department of mathematics	1640.83
10	National Institute of Technology, Uttarakhand	department of mathematics	1387.8
11	University of Delhi	department of mathematics	1284.0

Fig 5: College Ranking based on 'Department of Mathematics'


Back

civil engineering

Rank	College	Department	Score
1	Indian Institute of Science, Bangalore	civil engineering	1875.74
2	Visvesvaraya National Institute of Technology, Nagpur	civil engineering	1814.0
3	Netaji Subhas University of Technology	civil engineering	1131.0
4	Indian Institute of Technology, Gandhinagar	civil engineering	1091.33
5	Indian Institute of Technology Roorkee	civil engineering	911.71
6	Karunya Institute of Technology and Sciences, Coimbatore	civil engineering	783.71
7	Jamia Millia Islamia	civil engineering	403.0
8	National Institute of Technology Karnataka	civil engineering	395.62
9	Aligarh Muslim University	civil engineering	103.0
10	Dr B R Ambedkar National Institute of Technology, Jalandhar	civil engineering	6.0

Fig 6: College Ranking based on 'civil engineering'

9. COLLEGE RANKING

The objective is to provide a normalized ranking of colleges, accounting for the academic contributions of professors across different departments. The metrics considered here include the H Index, Total Citations, and Publications.

Load the Data: The initial step involves loading the cleaned dataset containing professor data.

```
file_path = 'IRINS_professor_Database_cleaned.csv'
data = pd.read_csv(file_path)
```

Normalize Department Names: To ensure case-insensitive grouping, department names are converted to lowercase.

```
data['Department'] = data['Department'].str.lower()
```

Group by College and Department: The data is grouped by `College` and `Department`, aggregating metrics such as the sum of H Index, Total Citations, Publications, and the count of professors.

```
grouped = data.groupby(['College', 'Department']).agg({
    'H Index': 'sum',
    'Total Citations': 'sum',
    'Publication': 'sum',
    'Name': 'size'  # Assuming 'Name' column represents the count of professors
}).reset_index()

grouped.rename(columns={'Name': 'ProfCount'}, inplace=True)
```

Calculate Total Score and Normalize: A total score is calculated for each department by summing the aggregated metrics. This score is then normalized by dividing by the number of professors.

```
grouped['TotalScore'] = grouped['H Index'] + grouped['Total Citations'] + grouped['Publication']
grouped['NormalizedScore'] = grouped['TotalScore'] / grouped['ProfCount']
```

Filter Out Departments with Zero Scores: Departments with a total score of zero are filtered out to maintain meaningful rankings.

```
grouped = grouped[grouped['TotalScore'] > 0]
```

Aggregate Normalized Scores for Each College: The normalized scores are aggregated for each college by calculating the mean of the department scores.

```
college_scores = grouped.groupby('College').agg({
    'NormalizedScore': 'mean'
}).reset_index()
```

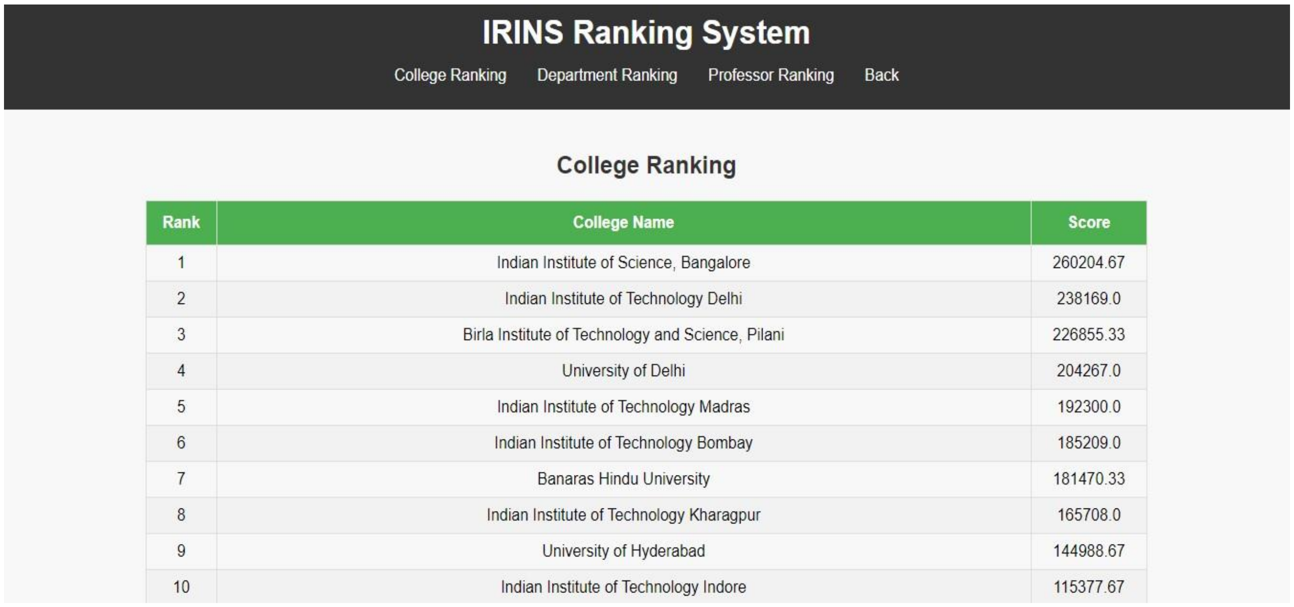
Rank the Colleges: Colleges are ranked based on their aggregated normalized score. The ranking is done in descending order, with the highest score receiving the top rank.

```
college_scores['Rank'] = college_scores['NormalizedScore'].rank(ascending=False, method='min')
```

Sort by Rank: The final dataset is sorted by rank, and only the relevant columns are selected and reordered.

```
college_scores = college_scores.sort_values(['Rank']).reset_index(drop=True)
final_df = college_scores[['Rank', 'College', 'NormalizedScore']]
```

It provides a comprehensive approach to ranking colleges based on normalized academic metrics of professors. The steps ensure that the data is cleaned, aggregated, and normalized appropriately to reflect the relative performance of colleges in a fair and meaningful way. The resulting CSV file can be used for various analytical and reporting purposes.



College Ranking

Rank	College Name	Score
1	Indian Institute of Science, Bangalore	260204.67
2	Indian Institute of Technology Delhi	238169.0
3	Birla Institute of Technology and Science, Pilani	226855.33
4	University of Delhi	204267.0
5	Indian Institute of Technology Madras	192300.0
6	Indian Institute of Technology Bombay	185209.0
7	Banaras Hindu University	181470.33
8	Indian Institute of Technology Kharagpur	165708.0
9	University of Hyderabad	144988.67
10	Indian Institute of Technology Indore	115377.67

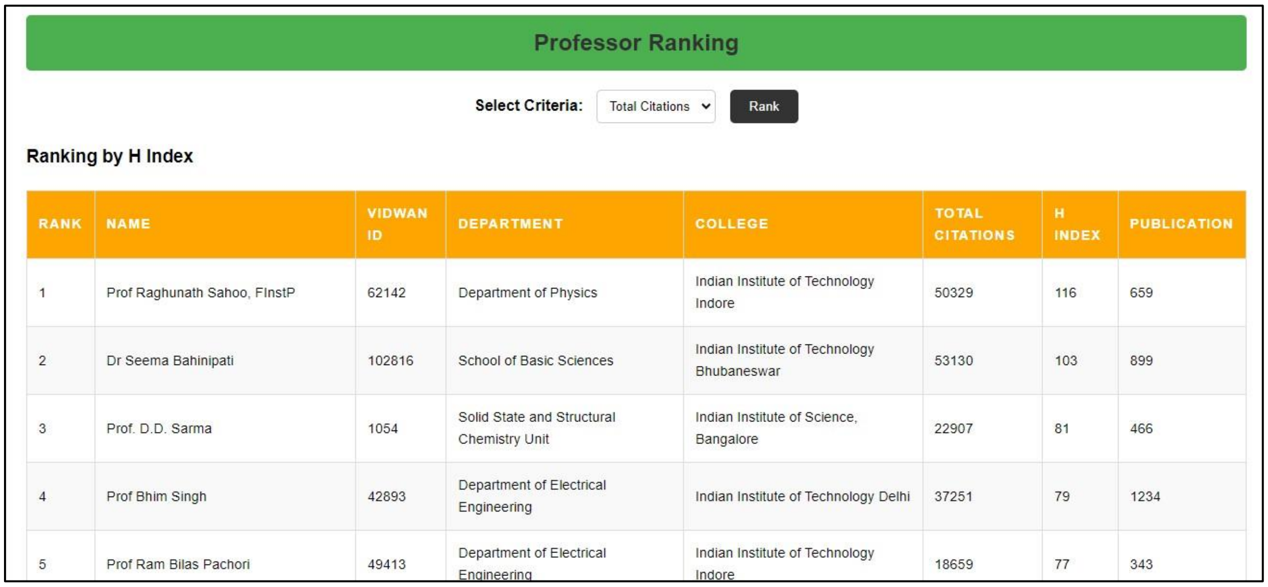
Fig 7: College Ranking Page

10. RESULT:

Total	**Count**
Total Professor/Scientists	20570
Total Department	1697
Total College/ University	204
Total Publications	768285

Table 1: Table containing Total Data Scraped

We successfully developed a system that ranks 204 universities, 1697 departments, and 20570 professors based on crucial academic metrics such as H-index, total citations, and total publications.



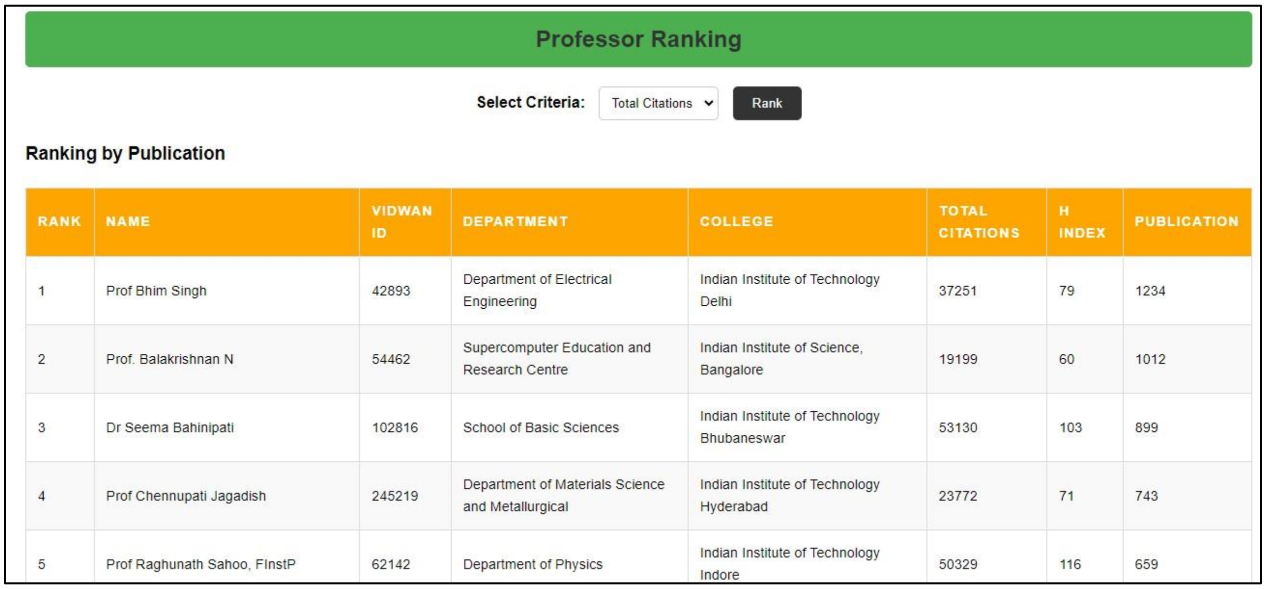
Professor Ranking

Select Criteria: Total Citations | Rank

Ranking by H Index

RANK	NAME	VIDWAN ID	DEPARTMENT	COLLEGE	TOTAL CITATIONS	H INDEX	PUBLICATION
1	Prof Raghunath Sahoo, FInstP	62142	Department of Physics	Indian Institute of Technology Indore	50329	116	659
2	Dr Seema Bahinipati	102816	School of Basic Sciences	Indian Institute of Technology Bhubaneswar	53130	103	899
3	Prof. D.D. Sarma	1054	Solid State and Structural Chemistry Unit	Indian Institute of Science, Bangalore	22907	81	466
4	Prof Bhim Singh	42893	Department of Electrical Engineering	Indian Institute of Technology Delhi	37251	79	1234
5	Prof Ram Bilas Pachori	49413	Department of Electrical Engineering	Indian Institute of Technology Indore	18659	77	343

Fig 8: Professor Ranking by H-Index



Professor Ranking

Select Criteria: Total Citations | Rank

Ranking by Publication

RANK	NAME	VIDWAN ID	DEPARTMENT	COLLEGE	TOTAL CITATIONS	H INDEX	PUBLICATION
1	Prof Bhim Singh	42893	Department of Electrical Engineering	Indian Institute of Technology Delhi	37251	79	1234
2	Prof. Balakrishnan N	54462	Supercomputer Education and Research Centre	Indian Institute of Science, Bangalore	19199	60	1012
3	Dr Seema Bahinipati	102816	School of Basic Sciences	Indian Institute of Technology Bhubaneswar	53130	103	899
4	Prof Chennupati Jagadish	245219	Department of Materials Science and Metallurgical	Indian Institute of Technology Hyderabad	23772	71	743
5	Prof Raghunath Sahoo, FInstP	62142	Department of Physics	Indian Institute of Technology Indore	50329	116	659

Fig 9: Professor Ranking by Publication

11.CONCLUSION

This project involves data analysis by ranking colleges using IRINS data through advanced web scraping techniques. By harnessing tools like Python, BeautifulSoup, Selenium, and Pandas, we automated the collection of key metrics like H-index, citations, and publications, overcoming challenges like dynamic content and pagination. This ensured our data was accurate and current.Our system not only simplifies data collection but also enhances accessibility through a user-friendly Flask web app. This allows anyone to explore and analyze rankings effortlessly. Overall, our approach sets a new standard in evaluating academic performance, making complex data simple and actionable for educational stakeholders.

12. FUTURE SCOPE

The methodologies and frameworks developed in this study can be extended and refined for broader applications. Future work could involve:

a) Expanding the dataset to include more institutions and a wider range of academic metrics.

b) Incorporating additional factors such as funding, collaboration networks, and societal impact to provide a more holistic evaluation.

c) Developing interactive platforms and dashboards to visualize the data and make it accessible to a wider audience.

13. REFERENCES

[1] Python Software Foundation. Python Language Reference, version 3. Available at: https://www.python.org/doc/

[2] Crummy, L. (2021). Beautiful Soup Documentation. Available at: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[3] SeleniumHQ. Selenium WebDriver. Available at: https://www.selenium.dev/documentation/webdriver/

[4] Indian Research Information Network System (IRINS). Available at: https://irins.inflibnet.ac.in/

[5] Cloudscraper GitHub Repository. Available at: https://github.com/VeNoMouS/cloudscraper

[6] Pallets Projects. Flask Documentation. Available at: https://flask.palletsprojects.com/