Engineering Branch Analytics Web Application

Problem

The process of selecting an engineering branch is often overwhelming for students due to the lack of comprehensive, data-driven insights. With numerous branches spread across various institutes, students struggle to identify which options align with current demand, future growth potential, or risks of declining relevance. Most decisions are based on word of mouth or outdated rankings, leading to confusion and misaligned career paths. Existing datasets—like student preferences, rank trends, and institutional offerings—are fragmented and rarely analysed in a way that helps students understand patterns. In such a competitive environment, this absence of clear analytics hinders informed, confident decision-making during a critical phase of a student's academic journey.

Description

This project develops a web application utilizing Flask, a Python web framework, to visualize engineering branch data from IITs, sourced from an Excel file (dataset1.xlsx). The application supports multiple plot types, including bar charts, line graphs, boxplots, heatmaps, and pie charts, generated using Matplotlib and Seaborn. Users interact with the application via a web interface, selecting plot types to analyse trends such as closing ranks and branch availability across institutes. The system processes user requests, generates visualizations, and displays them dynamically.

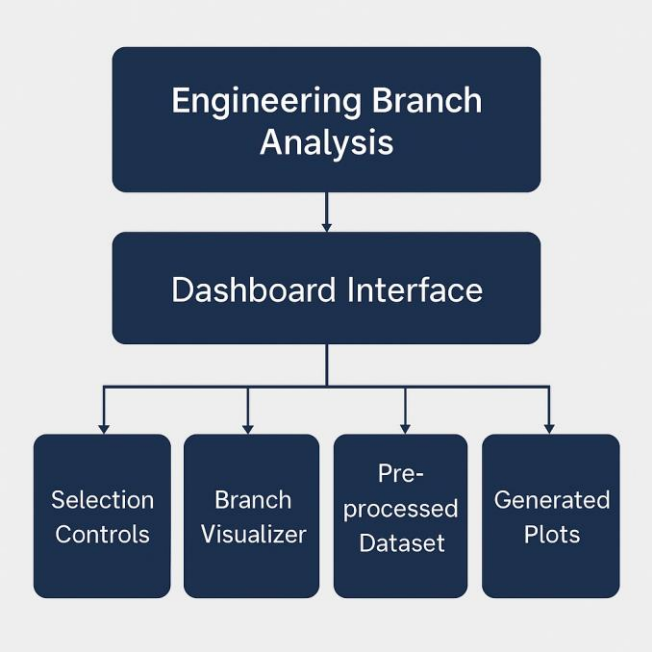
Purpose

The primary purpose of this project is to provide a data-driven solution that assists engineering aspirants in making informed decisions when selecting their preferred branches during admissions. With the increasing complexity and competitiveness in the field of engineering education, students often lack reliable, structured insights into trends such as branch popularity, institute availability, and rank-based demand. This project aims to bridge that gap by offering a visual analytics platform that consolidates historical admission data and transforms it into interactive graphs and charts. By doing so, it empowers students to evaluate their options more clearly and choose branches that align with both their interests and future growth potential.

Outcome

- A fully functional, responsive analytics dashboard that presents visual insights into branch-related data.
- Clear identification of trends such as rising or declining demand for branches over years.
- Automatic generation of six plot types, making insights accessible and easy to interpret.
- Easy navigation for users to explore institute count, closing rank averages, and popularity distributions.
- The application works offline or online and is easily deployable after cloning.

Diagram



Implementation

The project was implemented using a modular, full-stack approach combining Python for the backend logic and visualization, and HTML/CSS/JavaScript for the frontend. Below are the key steps and modules used in the implementation:

1. Backend Setup (Flask & Python)

- A Flask server was created using app.py to handle HTTP requests.
- Routes:
    - '/': Renders the homepage using index.html.
    - '/plot': Receives a POST request with a selected plot type and calls the corresponding visualization function from visuals.py.

2. Data Handling & Visualization (visuals.py)

- Data is loaded from an Excel file (dataset1.xlsx) using pandas.
- A custom class BranchVisualizer was created to encapsulate plotting methods.
- The six plotting methods generate graphs using **Matplotlib** and **Seaborn** and save them as .png files in a static/plots directory.
- A helper method _save_plot() was written to save plots and return file paths for use in the frontend.

3. Frontend (index.html + Bootstrap + JavaScript)

- A simple, clean HTML interface using **Bootstrap 5** allows the user to select a plot type from a dropdown menu.
- A button triggers the visualization process using **JavaScript**'s fetch() API.
- The generated plot image is dynamically loaded and displayed in the browser without refreshing the page.

4. Integration & Templating (Jinja2)

- The render_template() function in Flask uses **Jinja2 templating** to embed references to static CSS and scripts.
- This ensures dynamic but organized loading of UI components and styles.

5. Responsiveness and Styling

- A style.css file located in the static/ folder adds branding and a calm dark-theme aesthetic.
- The layout is fully responsive and suitable for both desktop and tablet devices.

6. File and Folder Structure

Project \

- app.py
- visuals.py
- dataset1.xlsx
- templates \
  - index.html
- static \
  - plots
  - style.css

A	B	C	D	E	F	G	H	I
year	round	institute	branch	quota	seat type	gender	pening rank	losing rank
2021	6	Indian Institute of Technology Bhubaneswar	Civil Engineering	AI	OPEN	Gender-Neutral	8471	12396
2021	6	Indian Institute of Technology Bhubaneswar	Computer Science and Engineering	AI	OPEN	Gender-Neutral	1621	2304
2021	6	Indian Institute of Technology Bhubaneswar	Electrical Engineering	AI	OPEN	Gender-Neutral	4556	5877
2021	6	Indian Institute of Technology Bhubaneswar	Electronics and Communication Engineering	AI	OPEN	Gender-Neutral	3006	4174
2021	6	Indian Institute of Technology Bhubaneswar	Mechanical Engineering	AI	OPEN	Gender-Neutral	7084	8741
2021	6	Indian Institute of Technology Bombay	Aerospace Engineering	AI	OPEN	Gender-Neutral	123	2118
2021	6	Indian Institute of Technology Bombay	Chemical Engineering	AI	OPEN	Gender-Neutral	1266	2213
2021	6	Indian Institute of Technology Bombay	Civil Engineering	AI	OPEN	Gender-Neutral	2226	3304
2021	6	Indian Institute of Technology Bombay	Computer Science and Engineering	AI	OPEN	Gender-Neutral	1	67
2021	6	Indian Institute of Technology Bombay	Electrical Engineering	AI	OPEN	Gender-Neutral	95	434

- snip of dataset used

Code & Explanation

This section breaks down the primary code files used in the project: app.py, visuals.py, and the HTML frontend files (index.html, plot.html).

1. **app.py – Flask Backend**

```
from flask import Flask, render_template, request, jsonify
from visuals import BranchVisualizer

app = Flask(__name__)
visualizer = BranchVisualizer(data_path='dataset1.xlsx')
```

- Flask is imported for web server functionality.
- BranchVisualizer (defined in visuals.py) is initialized with the dataset.

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

- Renders the main UI when the root URL is accessed.

```
@app.route('/plot', methods=['POST'])
def plot():
    data = request.get_json()
    plot_type = data.get('plot_type')

    plot_methods = {
        'avg_closing_rank': visualizer.avg_closing_rank,
        'trend': visualizer.trend,
        'boxplot': visualizer.boxplot,
        'institute_count': visualizer.institute_count,
        'heatmap': visualizer.heatmap,
        'pie_branch': visualizer.pie_branch,
    }

    if plot_type in plot_methods:
        plot_path = plot_methods[plot_type]()
        return jsonify({'plot_url': '/' + plot_path})
    else:
        return jsonify({'error': 'Invalid plot type'}), 400
```

- Receives a plot request via POST.
- Based on the user's selected plot_type, the corresponding visualization function is called.
- The resulting plot path is returned as a JSON response.

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

- Starts the Flask development server.

2. **visuals.py – Data Handling and Plotting**

```
import pandas as pd
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

- Pandas handles Excel data.
- Matplotlib and Seaborn are used for plotting.
- 'Agg' is used for headless backend (no GUI display).

```
class BranchVisualizer:
    def __init__(self, data_path='dataset1.xlsx', output_dir='static/plots'):
        self.df = pd.read_excel(data_path)
        self.output_dir = output_dir
        os.makedirs(output_dir, exist_ok=True)
```

- Loads Excel data into self.df and ensures the plot directory exists.

```
def _save_plot(self, fig, filename):
        path = os.path.join(self.output_dir, filename)
        fig.savefig(path, bbox_inches='tight')
        plt.close(fig)
        return f'static/plots/{filename}'
```

- A helper function to save each plot and return its path.

Visualization Methods:

Each method generates a different plot:

```
def avg_closing_rank(self):
        avg_rank = self.df.groupby('branch')['closing rank'].mean().sort_values()
        fig, ax = plt.subplots(figsize=(10, 6))
        sns.barplot(x=avg_rank.values, y=avg_rank.index, ax=ax, palette="magma")
        ax.set_title('Average Closing Rank by Branch')
        ax.set_xlabel('Average Closing Rank')
        ax.set_ylabel('Branch')
        return self._save_plot(fig, 'avg_closing_rank.png')
```

- **Bar plot** showing average closing ranks by branch

```
def trend(self):
        fig, ax = plt.subplots(figsize=(12, 8))
        for branch in self.df['branch'].unique():
            trend = self.df[self.df['branch'] == branch].groupby('year')['closing
rank'].mean()
            ax.plot(trend.index, trend.values, marker='o', label=branch)
        ax.set_title('Average Closing Rank Trend per Year (All Branches)')
        ax.set_xlabel('Year')
```

```
        ax.set_ylabel('Avg Closing Rank')
        ax.legend(fontsize='small', loc='upper right', bbox_to_anchor=(1.3, 1.0))
        ax.grid(True)
        return self._save_plot(fig, 'trend.png')
```

- **Line plot** to visualize year-wise trends in average closing ranks for each branch.

```
def boxplot(self):
        fig, ax = plt.subplots(figsize=(12, 6))
        sns.boxplot(data=self.df, x='branch', y='closing rank', ax=ax, palette="coolwarm")
        ax.set_title('Closing Rank Distribution by Branch')
        ax.set_xlabel('Branch')
        ax.set_ylabel('Closing Rank')
        plt.xticks(rotation=45)
        return self._save_plot(fig, 'boxplot.png')
```

- **Box plot** to show distribution (spread) of ranks for each branch.

```
def institute_count(self):
        inst_count =
self.df.groupby('branch')['institute'].nunique().sort_values(ascending=False)
        fig, ax = plt.subplots(figsize=(10, 6))
        sns.barplot(x=inst_count.values, y=inst_count.index, ax=ax, palette="magma")
        ax.set_title('Number of IITs Offering Each Branch')
        ax.set_xlabel('Number of IITs')
        ax.set_ylabel('Branch')
        return self._save_plot(fig, 'institute_count.png')
```

- **Bar plot** showing the number of institutes offering each branch.

```
def heatmap(self):
        pivot = self.df.pivot_table(index='branch', columns='year', values='closing rank',
aggfunc='mean')
        fig, ax = plt.subplots(figsize=(12, 6))
        sns.heatmap(pivot, annot=True, fmt=".0f", cmap="magma", ax=ax)
        ax.set_title('Heatmap of Average Closing Ranks per Branch per Year')
        return self._save_plot(fig, 'heatmap.png')
```

- **Heatmap** of average closing ranks by branch and year

```
def pie_branch(self):
        counts = self.df['branch'].value_counts()
        fig, ax = plt.subplots(figsize=(8, 8))
        ax.pie(counts.values, labels=counts.index, autopct='%1.1f%%', startangle=140)
        ax.set_title('Distribution of Branch Entries')
        return self._save_plot(fig, 'pie_branch.png')
```

- **Pie chart** showing distribution of total entries per branch.

3. index.html – Frontend Dashboard

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Branch Visualizer</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link href="{{ url_for('static', filename='style.css') }}" rel="stylesheet">
</head>
<body>
<div class="container mt-5 text-center">
    <h1 class="display-5">Engineering Branch Analysis</h1>
    <p class="text-muted">Engineering Branch Trends and Statistics</p>

    <div class="row justify-content-center mb-4 mt-4">
        <div class="col-auto">
            <select id="plotType" class="form-select">
                <option value="avg_closing_rank">Average Closing Rank</option>
                <option value="trend">Trend Over Years</option>
                <option value="boxplot">Boxplot by Branch</option>
                <option value="institute_count">Institutes per Branch</option>
                <option value="heatmap">Heatmap (Rank/Year)</option>
                <option value="pie_branch">Pie Chart (Branch)</option>
            </select>
        </div>
        <div class="col-auto">
            <button id="generateBtn" class="btn btn-primary">Show Plot</button>
        </div>
    </div>

    <div class="row justify-content-center">
        <div class="col-auto">
            <img id="plotImage" src=""
                class="img-fluid rounded shadow mx-auto d-block"
                style="display:none; max-height: 600px;" aria-hidden="true">
        </div>
    </div>
</div>

<script>
    document.getElementById('generateBtn').addEventListener('click', function () {
        const plotType = document.getElementById('plotType').value;
        fetch('/plot', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ plot_type: plotType })
        })
        .then(response => response.json())
        .then(data => {
            const img = document.getElementById('plotImage');
            img.src = data.plot_url.replace(/\\/g, '/');
            img.style.display = 'block';
```

```
        })
        .catch(err => console.error(err));
    });
</script>
</body>
</html>
```

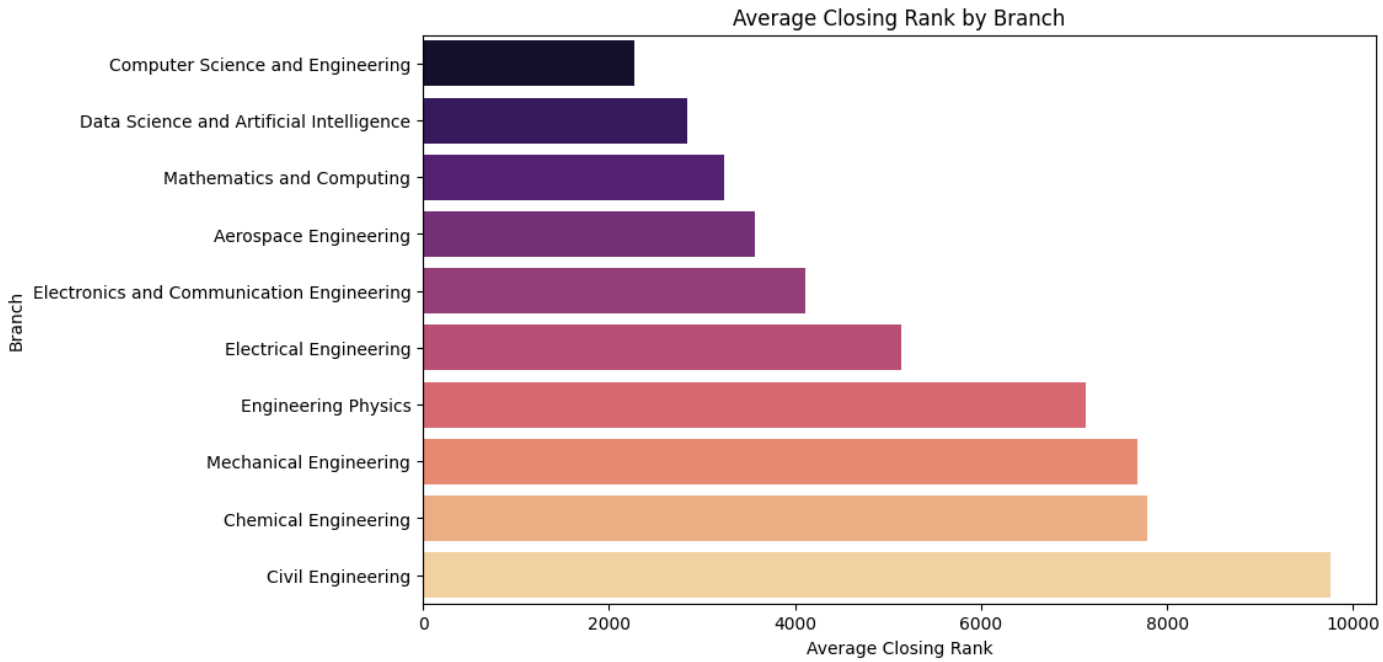
- Acts as the main user interface for selecting and displaying engineering branch visualizations.
- Provides a dropdown to choose plot types and a button to generate plots dynamically.
- Uses JavaScript to fetch and display plots from the Flask backend without reloading the page.

4. style.css – background, buttons and layout

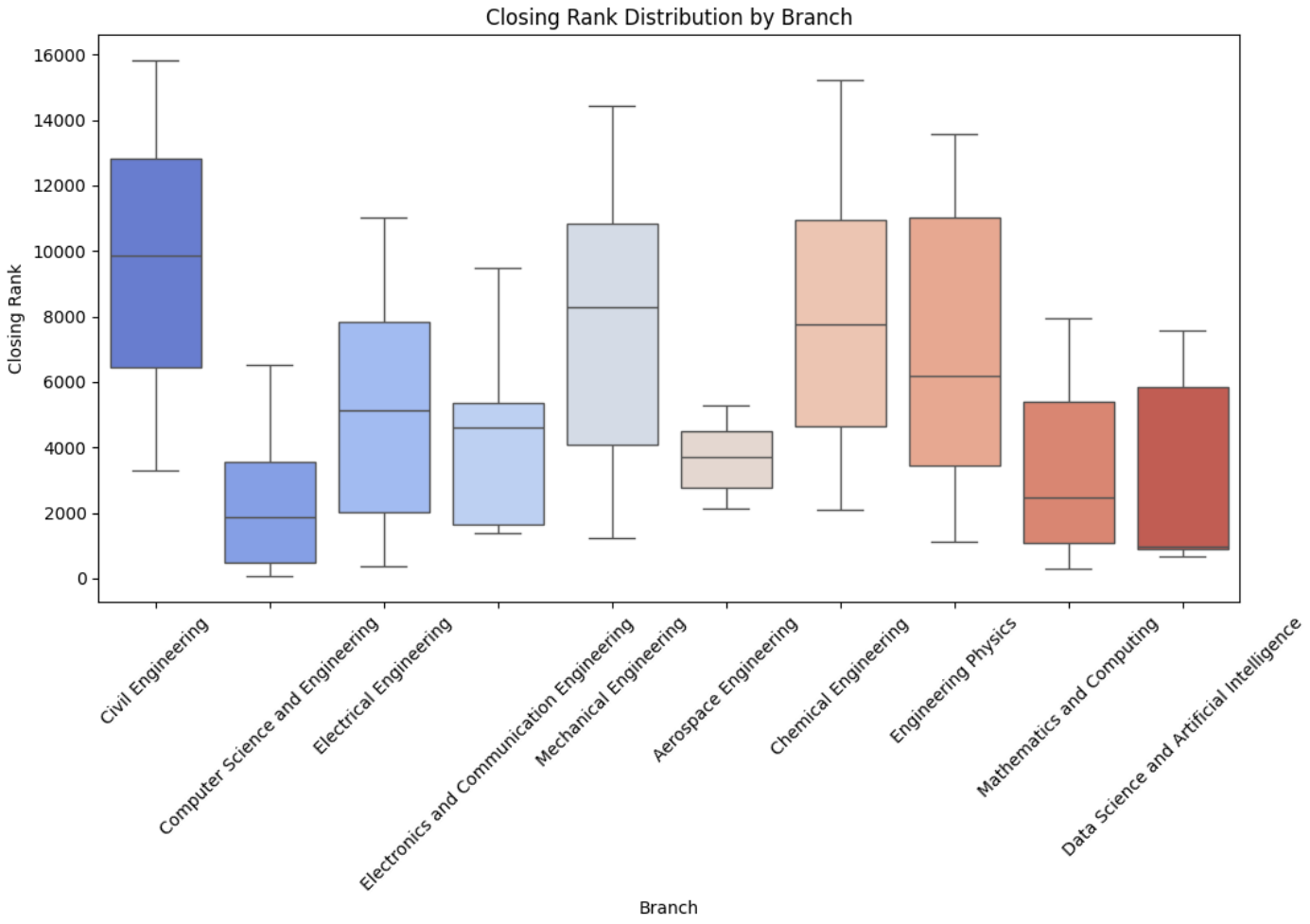
```
/* Full-height gradient background */
html, body {
    height: 100%;
    margin: 0;
    padding: 0;
    background: linear-gradient(to bottom right, #e5d4f9, #dbeafe);
    color: #2a2a2a;
    font-family: 'Segoe UI', sans-serif;
}

/* Headings and text */
h1, p {
    color: #2a2a2a;
}

/* Button styling - slightly deeper purple-blue for contrast */
.btn-primary {
    background-color: #a78bfa;
    border-color: #a78bfa;
    color: #ffffff;
    transition: all 0.3s ease;
}

.btn-primary:hover {
    background-color: #7c3aed;
    border-color: #7c3aed;
}

/* Dropdown select */
.form-select {
    background-color: #f8f5fc;
    color: #2a2a2a;
    border: 1px solid #d6d0ec;
    box-shadow: none;
    transition: all 0.3s ease;
}

.form-select:focus {
    border-color: #a78bfa;
```

```
    box-shadow: 0 0 0 0.2rem rgba(167, 139, 250, 0.25);
}

/* Image appearance */
img {
    border: 2px solid #ded8f5;
    border-radius: 8px;
    max-height: 600px;
    margin-top: 20px;
    transition: box-shadow 0.3s ease;
}

img:hover {
    box-shadow: 0 8px 24px rgba(167, 139, 250, 0.25);
}

/* Bottom scroll spacing */
.container {
    padding-bottom: 100px;
}
```

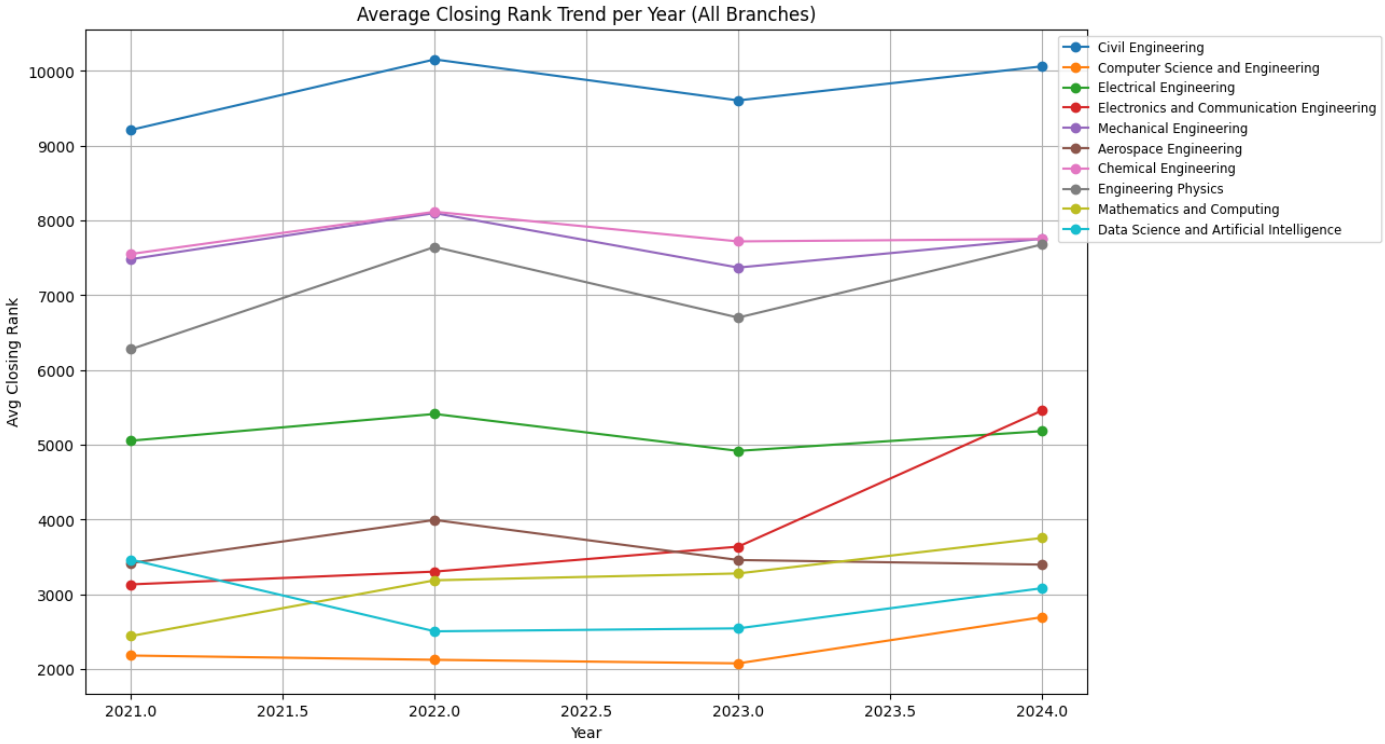
Output graphs



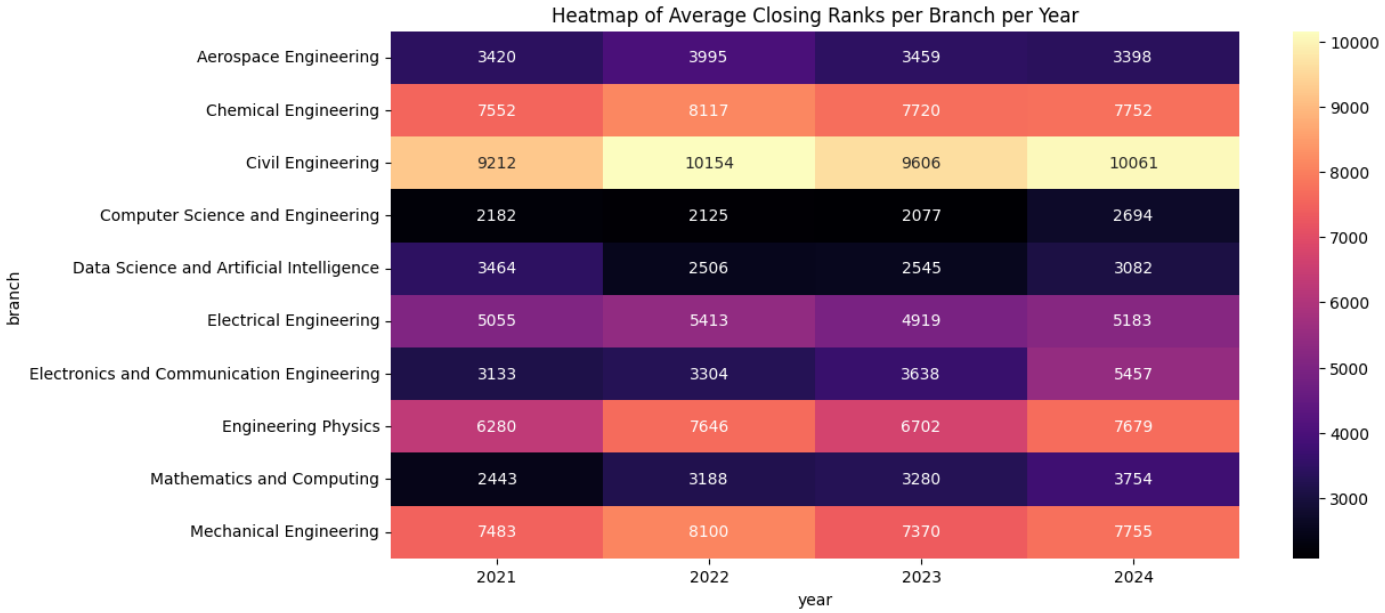
- Computer Science and Engineering often has the lowest average rank, showing it’s a popular and hard-to-get-into choice among the 15 branches.



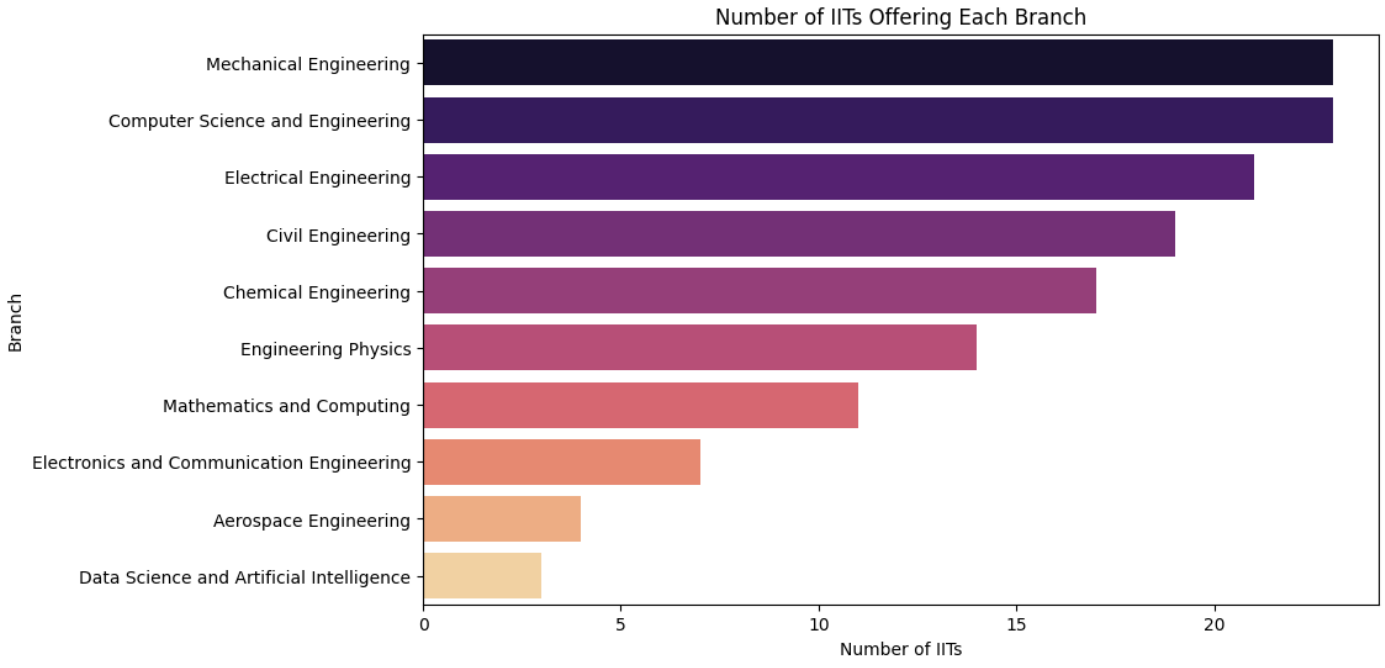
- Mechanical Engineering has a wide range of ranks, meaning it varies by institute, while Computer Science stays pretty steady and competitive



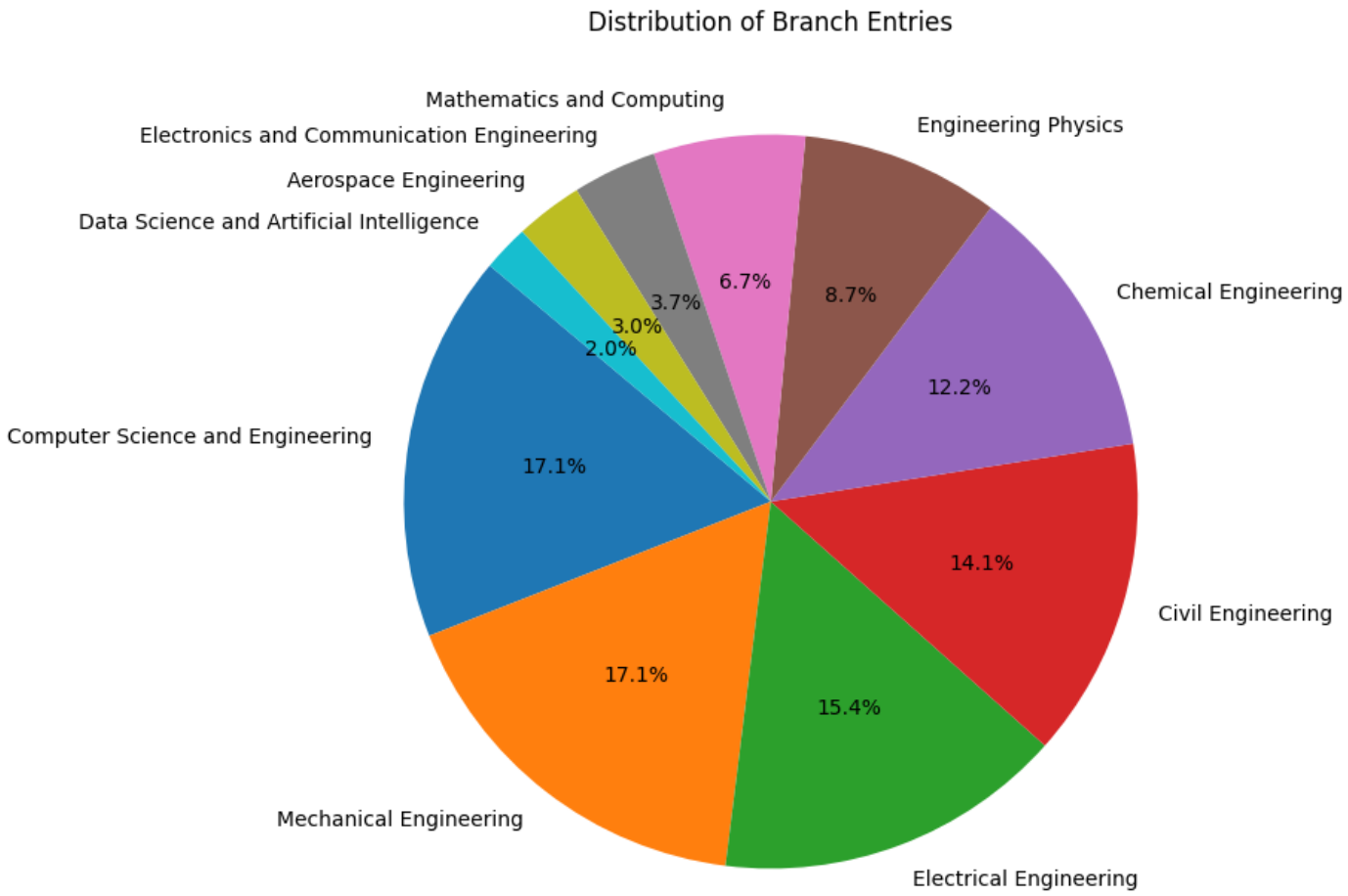
- The average closing ranks appear to rise from 2021 to 2024, which could suggest increasing competition or a possible reduction in seat availability across branches.



- The heatmap shows higher average closing ranks for most branches in 2024, which might indicate a recent increase in competition or a shift in student preferences.



- Branches such as Computer Science and Engineering are offered by the highest number of IITs, which may reflect their popularity and infrastructure support, correlating with lower closing ranks.



- The pie chart suggests that Computer Science and Engineering leads the entry distribution, supporting its status as a highly competitive branch.

Conclusion

This project presents a functional and data-driven platform to analyze trends in engineering branch selection. By integrating Flask with Matplotlib and Seaborn, it offers users an interactive dashboard to explore insights such as average closing ranks, branch-wise demand trends, and institutional availability. The visualizations are intended to assist students, educators, and counselors in making more informed decisions based on historical data.

Key findings from the data include:

- Branches like Computer Science and Artificial Intelligence show consistent signs of growth.
- Branches such as Metallurgy and Mining currently reflect lower demand.
- Computer Science holds the highest demand overall, with relatively low average closing ranks across institutes.
- Some branches, including Chemical Engineering, may be experiencing a gradual decline in interest.

The project highlights the importance of centralized, analytical tools in academic planning. Future improvements, such as real-time data integration and broader visualization options, can further enhance its relevance and usability

Bibliography

- Flask official site – https://flask.palletsprojects.com
- Matplotlib – https://matplotlib.org
- Seaborn – https://seaborn.pydata.org
- Pandas – https://pandas.pydata.org
- Bootstrap 5 – https://getbootstrap.com
- Datasets - https://www.kaggle.com/datasets/rahulkumarnitw/josaa-opening-and-closing-rank-dataset-2024 [aggregated and filtered]