Engineering Branches

Contents

Problem Statement:

Engineering institutions offer a wide range of branches. However, students often lack insights into how branches perform in the real world—whether in terms of placements or salaries. Data exists, but it's often underutilized or scattered.

Description:

The **Engineering Branch Analysis Web App** is a data visualization and insight-generation platform built using **Python, Flask, Pandas, and Matplotlib**. The application reads structured CSV data consisting of year-wise enrollment, placement, and salary information across various engineering disciplines.

This tool is designed to:

- **Analyze trends** in student enrollments and placements.
- **Compare average salaries** across branches.
- Provide **interactive charts and summaries** in a web interface.

The main objective is to help:

- **Students**: make informed decisions about which branches to pursue.

- **Institutions**: assess the performance of branches and curriculum needs.
- **Industry**: understand the pipeline of graduates from different disciplines.

The web interface runs locally and allows users to:

- View automatic analysis for the uploaded dataset.
- Examine key metrics like highest/lowest enrollment, placement efficiency, and top-paying branches.
- Study various charts including bar graphs, line plots, scatter plots, and pie charts—all generated from backend data computations.

Data & Insights:

➢ **Data Description:**

- Fields: Year, Branch, Enrolled, Placed, Average Salary
- Data format: CSV
- Sample years: 2015 to 2023
- Branches: 8 major engineering disciplines including CSE, ECE, Civil, AI/ML, etc.

➢ **Key Metrics Analyzed:**

- Total enrollments per year and branch
- Placement count and ratios
- Average salary trends
- Year-wise and branch-wise comparisons

Purpose:

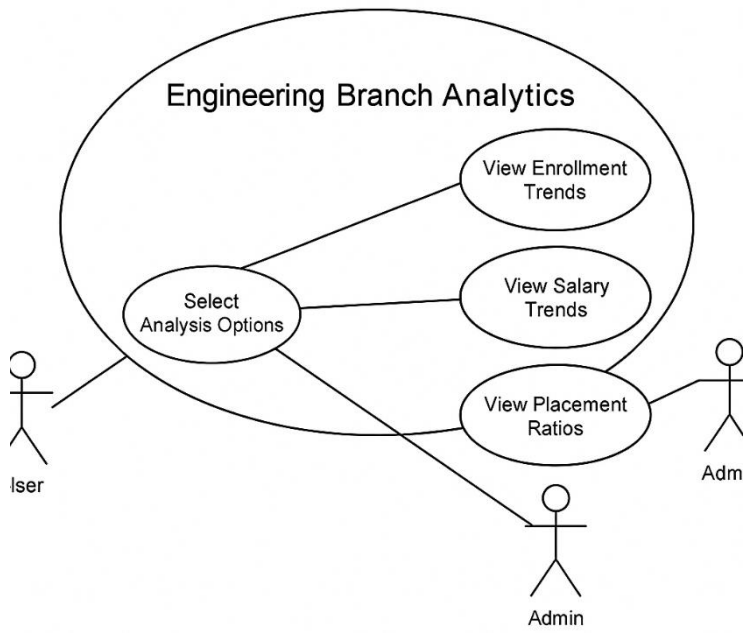
➢ To develop a lightweight web application that:

- Loads structured CSV data about engineering branch statistics.
- Performs comprehensive analysis of key metrics.
- Displays trends through clean and intuitive visualizations.
- Helps stakeholders make data-informed decisions.

Plan:

- **Steps Followed:**
  - Data ingestion using pandas
  - Cleaning and type conversion
  - Pivoting data for plotting
  - Plot generation using matplotlib
  - Integration into Flask backend
  - Web UI to trigger analysis and show results

Design:



Implementation:

- **Libraries Used:**
  - Pandas: for data loading and processing

- Numpy: for numerical operations
- Matplotlib: for plotting graph
- Flask: for web interface

➢ **Modules Implemented:**

- Data loader and processor
- Trend visualization (bar, line, pie)
- Summary and insights engine
- Web interface for viewing reports

Code & Explanation:

➢ engineering_branch.py

```
➢ #engineering_branch.py
➢
➢ import pandas as pd
➢ import matplotlib.pyplot as plt
➢ import numpy as np
➢ import os
➢
➢ def load_data(filepath):
➢     df = pd.read_csv(filepath)
➢     df['Enrolled'] = pd.to_numeric(df['Enrolled'], errors='coerce')
➢     df['Placed'] = pd.to_numeric(df['Placed'], errors='coerce')
➢     df['AvgSalary'] = pd.to_numeric(df['AvgSalary'], errors='coerce')
➢     df['PlacementRatio'] = df['Placed'] / df['Enrolled']
➢     return df
➢
➢ def create_pivots(df):
➢     return {
➢         'enroll': df.pivot(index='Year', columns='Branch', values='Enrolled'),
➢         'salary': df.pivot(index='Year', columns='Branch', values='AvgSalary'),
➢         'ratio': df.pivot(index='Year', columns='Branch', values='PlacementRatio'),
➢     }
➢
➢ def plot_enrollment(pivots, save_path=None): #save_path can save the plot to a file
➢     df = pivots['enroll']
➢     years = df.index
➢     branches = df.columns
➢     width = 0.1  # width of each bar in the bar chart
➢     x = np.arange(len(years))  # numeric positions for the x-axis, one per year
➢     plt.figure(figsize=(14, 6)) #14 inches wide × 6 inches tall
➢     for i, branch in enumerate(branches):
```

```
        plt.bar(x + i * width, df[branch], width, label=branch) #Plots bars for each year, slightly offset so that multiple branches don't overlap (x + i * width), uses branch name as a label.
    plt.title('Branch Enrollment Trends (Bar Graph)')
    plt.xlabel('Year')
    plt.ylabel('Students Enrolled')
    plt.xticks(x + width * len(branches) / 2, years)
    plt.legend()
    plt.grid(axis='y') #Adds horizontal grid lines
    plt.tight_layout() #adjusts the spacing between plot elements
    if save_path:
        plt.savefig(save_path)
        plt.close()
    else:
        plt.show()

def plot_salary(pivots, save_path=None):
    plt.figure(figsize=(12, 5))
    for branch in pivots['salary'].columns:
        plt.plot(pivots['salary'].index, pivots['salary'][branch], marker='s', linestyle='--', label=branch)
    plt.title('Average Salary Trend by Branch (LPA)')
    plt.xlabel('Year')
    plt.ylabel('Salary (LPA)')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path)
        plt.close()
    else:
        plt.show()

def plot_placement_ratio(pivots, save_path=None):
    plt.figure(figsize=(12, 5))
    for branch in pivots['ratio'].columns:
        plt.plot(pivots['ratio'].index, pivots['ratio'][branch], marker='^', label=branch)
    plt.title('Placement vs Enrollment Ratio by Branch')
    plt.xlabel('Year')
    plt.ylabel('Placement Ratio')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path)
        plt.close()
    else:
        plt.show()

def plot_pie(df, save_path=None): #enrollments for the latest year
```

```
    latest_year = df['Year'].max()
    latest_df = df[df['Year'] == latest_year]
    plt.figure(figsize=(8, 8))
    plt.pie(latest_df['Enrolled'], labels=latest_df['Branch'], autopct='%1.1f%%',
startangle=140)
    plt.title(f'Enrollment Share by Branch in {latest_year}')
    plt.axis('equal')
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path)
        plt.close()
    else:
        plt.show()

def get_summary(df):
    latest_year = df['Year'].max()
    latest_df = df[df['Year'] == latest_year]

    highest = latest_df.loc[latest_df['Enrolled'].idxmax()]
    lowest = latest_df.loc[latest_df['Enrolled'].idxmin()]
    salary = latest_df.loc[latest_df['AvgSalary'].idxmax()]
    best = latest_df.loc[latest_df['PlacementRatio'].idxmax()]
    worst = latest_df.loc[latest_df['PlacementRatio'].idxmin()]

    return {
        'year': latest_year,
        'highest_enrollment': (highest['Branch'], int(highest['Enrolled'])),
        'lowest_enrollment': (lowest['Branch'], int(lowest['Enrolled'])),
        'highest_salary': (salary['Branch'], salary['AvgSalary']),
        'best_ratio': (best['Branch'], round(best['PlacementRatio'], 2)),
        'worst_ratio': (worst['Branch'], round(worst['PlacementRatio'], 2))
    }

```

- **Explanation:**
  - *Importing Required Libraries :*

    This section imports essential libraries used for data analysis and visualization:
    - pandas is used for reading and manipulating CSV files.
    - matplotlib.pyplot is used for creating charts and graphs.
    - numpy is used for numerical calculations and array handling.
    - os is optionally used to handle file paths.
  - *Function: load_data(filepath)*

    This function reads data from a CSV file and prepares it for analysis.

    Working:
    - Reads the file using pandas.read_csv.
    - Converts the Enrolled, Placed, and AvgSalary columns to numeric format (to handle data issues).
    - Computes a new column PlacementRatio by dividing Placed by Enrolled.

- Returns the cleaned and enhanced DataFrame.

- *Function: create_pivots(df)*
  This function restructures the dataset into pivot tables for plotting.
  Working:

  - Creates three separate pivot tables:
    - 'enroll': Years as rows, Branches as columns, Enrolled values as data.
    - 'salary': Years as rows, Branches as columns, Average Salary as data.
    - 'ratio': Years as rows, Branches as columns, Placement Ratio as data.
  - Returns a dictionary of pivot tables.

- *Function: plot_enrollment(pivots, save_path=None)*
  This function plots a **bar graph** of enrollments across branches and years.

Working:

- Retrieves the enrollment pivot table.
- Creates a grouped bar chart, with bars slightly offset to avoid overlap.
- Displays years on the x-axis and enrollment numbers on the y-axis.
- Adds grid lines, legend, labels, and adjusts layout.
- Saves the plot if save_path is provided; otherwise, shows it.

- *Function: plot_salary(pivots, save_path=None)*
  This function creates a **line graph** to show average salary trends per branch over the years.
  Working:

  - Uses the salary pivot to plot salary data.
  - Uses square markers and dashed lines for visual clarity.
  - Displays titles, axis labels, grid, and legends.
  - Saves or displays the plot based on the input parameter.

- *Function: plot_placement_ratio(pivots, save_path=None)*
  This function draws a **line chart** comparing placement efficiency across branches.
  Working:

  - Uses the placement ratio pivot for plotting.
  - Plots data with triangle markers and lines.
  - Helps compare how well each branch converts enrolled students into placed candidates.
  - Optionally saves or displays the chart.

- *Function: plot_pie(df, save_path=None)*
  This function creates a **pie chart** showing the share of enrolled students by branch for the latest year.
  Working:

  - Identifies the most recent year from the dataset.
  - Filters the data for that year.
  - Creates a pie chart showing percentage enrollment of each branch.
  - Ensures circular proportion using axis('equal').

- Saves or displays the chart.

- *Function: get_summary(df)*
  This function returns a dictionary summarizing key statistics for the most recent year.
  Working:

  - Filters the dataset for the latest year.
  - Identifies:
    - The branch with the highest enrollment.
    - The branch with the lowest enrollment.
    - The branch with the highest average salary.
    - The branch with the best placement ratio.
    - The branch with the worst placement ratio.
  - Returns all these insights as a dictionary, which can be used in reports or dashboards.

➢ app.py

```
➢ from flask import Flask, render_template, request
➢ import os
➢ from engineering_branch import *
➢
➢ data_file = 'data/engineering_branch_data.csv'
➢ plot_folder = 'static/plots'
➢
➢ app = Flask(__name__)
➢ os.makedirs(plot_folder, exist_ok=True)
➢
➢ @app.route('/', methods=['GET', 'POST'])
➢ def index():
➢     summary = None
➢     plots = []
➢     options = request.form.getlist('option') if request.method == 'POST' else []
➢
➢     if os.path.exists(data_file):
➢         df = load_data(data_file)
➢         pivots = create_pivots(df)
➢
➢         if 'enrollment' in options:
➢             plot_enrollment(pivots, os.path.join(plot_folder, 'enrollment.png'))
➢             plots.append('enrollment.png')
➢
➢         if 'salary' in options:
➢             plot_salary(pivots, os.path.join(plot_folder, 'salary.png'))
➢             plots.append('salary.png')
➢
➢         if 'placement' in options:
➢             plot_placement_ratio(pivots, os.path.join(plot_folder, 'placement.png'))
➢             plots.append('placement.png')
➢
➢         if 'pie' in options:
➢             plot_pie(df, os.path.join(plot_folder, 'pie.png'))
```

```
            plots.append('pie.png')

        if 'summary' in options:
            summary = get_summary(df)

    return render_template('index.html', plots=plots, summary=summary)

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

- **Explanition:**
  - *Importing Required Modules*

    The code begins by importing necessary modules:

    - Flask, render_template, and request from the flask library are used to build the web app, render HTML templates, and handle user input.
    - os is used to handle file and folder operations like checking file existence and creating folders.
    - * from engineering_branch imports all the analysis functions and tools like load_data, create_pivots, and plotting functions.

  - *File and Folder Configuration*

    - The variable data_file stores the path to the CSV file containing the engineering branch data.
    - The variable plot_folder stores the path where generated plots (graphs) will be saved.
    - Flask(__name__) initializes the Flask web application.
    - os.makedirs(..., exist_ok=True) ensures that the static/plots folder is created if it does not already exist. This folder is used to store images that will be shown on the website.

  - *Home Route Definition (/)*

    The route '/' is defined using @app.route() and supports both GET and POST requests.
    Function: index()

    - **Variables Initialized:**
      - summary is set to None initially and will store the statistical summary.
      - plots is an empty list to collect names of plots selected by the user.
      - options is used to collect the selected form checkboxes when the user submits the form using POST.

  - *Data Handling (if File Exists)*

    The code checks if the data file exists:

    - If it does, load_data(data_file) loads and cleans the data.
      - create_pivots(df) is then used to prepare the data in a format that makes plotting easier (pivot tables for enrollment, salary, and placement ratio).

- *Handling User Selections*

Based on the options selected by the user in the web form, the corresponding functions are called:

- **Enrollment Plot**: If 'enrollment' is selected, the plot_enrollment function generates a bar chart of branch-wise enrollment trends and saves it as 'enrollment.png'.
- **Salary Plot**: If 'salary' is selected, plot_salary generates a line graph showing the salary trends by branch.
- **Placement Ratio Plot**: If 'placement' is selected, plot_placement_ratio generates a graph showing the placement-to-enrollment ratio.
- **Pie Chart**: If 'pie' is selected, plot_pie creates a pie chart showing enrollment share of each branch for the latest year.
- **Summary Stats**: If 'summary' is selected, get_summary computes and stores the statistical summary for the latest year (e.g., highest/lowest enrolled branch, best/worst placement ratio, etc.).

Each time a plot is generated, the filename is added to the plots list so it can be displayed later on the webpage.

- *Rendering the Web Page*

After processing the data and generating the required outputs:

- render_template('index.html', plots=plots, summary=summary) renders the HTML page.
- The selected plots and the summary (if any) are passed to the HTML template (index.html) to be displayed dynamically on the page.

- *Running the Flask App*

At the bottom of the file:

- if __name__ == '__main__': app.run(debug=True) ensures the app runs only when the script is executed directly.
- debug=True allows for real-time updates and detailed error messages during development, making it easier to debug the app.

➢ index.html

```
➢ <!doctype html>
➢ <html>
➢ <head>
➢     <title>Engineering Branch Analytics</title>
➢     <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
➢
➢ </head>
➢ <body>
➢     <h1>📊 Engineering Branch Analysis</h1>
➢     <form method="POST" enctype="multipart/form-data">
➢
➢         <div class="form-card">
```

```
    <h2>□ <strong>Select Analysis Options</strong></h2>
    <div class="checkbox-container">
        <label class="checkbox-item">
            <input type="checkbox" name="option" value="enrollment">
            📊 Enrollment Trend
        </label>
        <label class="checkbox-item">
            <input type="checkbox" name="option" value="salary">
            💼 Salary Trend
        </label>
        <label class="checkbox-item">
            <input type="checkbox" name="option" value="placement">
            📈 Placement Ratio
        </label>
        <label class="checkbox-item">
            <input type="checkbox" name="option" value="pie">
            □ Enrollment Pie Chart
        </label>
        <label class="checkbox-item">
            <input type="checkbox" name="option" value="summary">
            📋 Summary Report
        </label>
    </div>

    <button type="submit" class="styled-button">🚀 Run Analysis</button>
</div>

    {% if summary %}
<div class="card-section">
    <div class="summary-card interactive-card">
        <h2>📋 Summary Report ({{ summary.year }})</h2>
        <ul>
            <li>▲ Highest Enrollment: {{ summary.highest_enrollment[0] }} ({{
summary.highest_enrollment[1] }})</li>
            <li>▼ Lowest Enrollment: {{ summary.lowest_enrollment[0] }} ({{
summary.lowest_enrollment[1] }})</li>
            <li>💰 Highest Salary: {{ summary.highest_salary[0] }} ({{
summary.highest_salary[1] }} LPA)</li>
            <li>🎯 Best Placement Ratio: {{ summary.best_ratio[0] }} ({{
summary.best_ratio[1] }})</li>
            <li>⚠ Worst Placement Ratio: {{ summary.worst_ratio[0] }} ({{
summary.worst_ratio[1] }})</li>
        </ul>
    </div>
</div>
{% endif %}

{% if plots %}
<div class="card-section">
    <h2>📈 Generated Charts</h2>
```

```
    <div class="charts-grid">
        {% for plot in plots %}
            <div class="chart-card interactive-card">
                <img src="{{ url_for('static', filename='plots/' + plot) }}" alt="Chart {{ loop.index }}">
            </div>
        {% endfor %}
    </div>
</div>
{% endif %}

</body>
</html>
```

Explanition:

1. <!doctype html>
   Declares the document type as HTML5.
2. <html>
   Opens the root element of the HTML document.
3. <head>
   Contains metadata and links for styling and page title.
4. <title>Engineering Branch Analytics</title>
   Sets the browser tab title for the web page.
5. <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
   Loads the external CSS stylesheet stored in the static/css/ folder using Flask's url_for function.
6. <body>
   Opens the body of the webpage where visible content appears.
7. <h1>📊 Engineering Branch Analysis</h1>
   Displays a heading at the top of the page with an emoji for visual enhancement.
8. <form method="POST" enctype="multipart/form-data">
   Starts a form element that uses the POST method to send data to the server. The enctype is set to handle file uploads if required later.
   Form Section: Analysis Options
9.
   Wraps the form in a styled container (using a class from CSS for a card-like effect).
10. <h2>□ Select Analysis Options</h2>
    Provides a subheading that introduces the available analysis options.
11.
    Begins a div that groups the checkbox inputs together for layout and styling.
    12–16. Each <label class="checkbox-item">...</label>
    Represents a checkbox input allowing users to choose which type of analysis they want. Each has:
    - An <input type="checkbox" name="option" value="..."> to specify the option.
    - An emoji and label for visual appeal and clarity:
        - 📊 Enrollment Trend
        - 💼 Salary Trend
        - 📈 Placement Ratio

- □ Enrollment Pie Chart
- 📋 Summary Report

17.
    Ends the checkbox container.
18. <button type="submit" class="styled-button">🚀 Run Analysis</button>
    Adds a styled submit button that triggers the form submission. The emoji adds a dynamic touch.
19.
    Ends the form card section.
    Conditional Summary Section (Jinja2)
20. {% if summary %}
    A Jinja2 conditional block that checks if a summary dictionary is passed to the template.
21.
    Begins a section to display the summary card.
22.
    Wraps the summary inside a styled interactive card.
23. <h2>📋 Summary Report ({{ summary.year }})</h2>
    Displays the year of the summary using Jinja2 templating syntax.
    24–28. ...
    Lists key summary insights:
    - Highest and lowest enrollment branches and values.
    - Highest salary offered.
    - Best and worst placement ratios.
29. and
    Close the summary card and section respectively.
30. {% endif %}
    Ends the Jinja2 conditional block for the summary.
    Conditional Plots Section (Jinja2)
31. {% if plots %}
    A Jinja2 block that checks if there are any plots to display.
32.
    Begins a new section to display the charts.
33. <h2>📈 Generated Charts</h2>
    Introduces the section with a heading.
34.
    Creates a CSS grid layout to display chart images.
35. {% for plot in plots %}
    A Jinja2 loop that iterates through each plot file name in the plots list.
36.
    Wraps each chart image in a styled card with interactive effects.
37. <img src="{{ url_for('static', filename='plots/' + plot) }}" alt="Chart {{ loop.index }}">
    Dynamically sets the image source for each chart using the plot filename, and assigns an alt text based on the loop index.
38.
    Closes each chart card.

39. {% endfor %}
    Ends the loop.
40. and
    Closes the charts grid and the section.
41. {% endif %}
    Ends the Jinja2 conditional block for plots.
42. </body> and </html>
    Closes the body and HTML document.

- style.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f9;
    color: #333;
    padding: 20px;
    text-align: center;
}

h1, h2 {
    color: #2c3e50;
}

form {
    margin: 20px auto;
    background-color: #ffffff;
    padding: 20px;
    border-radius: 12px;
    width: 50%;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

input[type="checkbox"] {
    margin-right: 8px;
}

button {
    padding: 10px 20px;
    font-size: 16px;
    border-radius: 8px;
    background-color: #3498db;
    color: white;
    border: none;
    cursor: pointer;
    margin-top: 10px;
}

button:hover {
    background-color: #2980b9;
}
```

```
➢
➢  ul {
➢      text-align: left;
➢      display: inline-block;
➢      margin-top: 10px;
➢  }
➢
➢  img {
➢      border: 1px solid #ccc;
➢      border-radius: 10px;
➢      margin-bottom: 15px;
➢  }
➢  .checkbox-group {
➢      display: flex;
➢      flex-direction: column;
➢      align-items: flex-start;
➢      margin-top: 10px;
➢      gap: 8px;
➢  }
➢
➢  .checkbox-group label {
➢      font-weight: normal;
➢      font-size: 16px;
➢  }
➢  .checkbox-container {
➢      background-color: #ffffff;
➢      padding: 20px;
➢      margin-top: 15px;
➢      border-radius: 12px;
➢      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);
➢      display: grid;
➢      grid-template-columns: repeat(auto-fit, minmax(240px, 1fr));
➢      gap: 15px;
➢      text-align: left;
➢  }
➢
➢  .checkbox-item {
➢      font-size: 16px;
➢      padding: 10px 12px;
➢      background-color: #f7f9fc;
➢      border-radius: 8px;
➢      transition: background-color 0.3s ease, transform 0.2s ease;
➢      display: flex;
➢      align-items: center;
➢      gap: 10px;
➢      cursor: pointer;
➢      border: 1px solid #ddd;
➢  }
➢
➢  .checkbox-item:hover {
```

```
    background-color: #e8f0fe;
    transform: translateY(-2px);
}

.checkbox-item input[type="checkbox"] {
    transform: scale(1.2);
    accent-color: #3498db;
}
body {
    font-family: 'Segoe UI', sans-serif;
    background: linear-gradient(135deg, #e0ecff, #f7faff);
    margin: 0;
    padding: 40px;
    color: #333;
}

h1 {
    font-size: 32px;
    margin-bottom: 20px;
}

.form-card {
    background: rgba(255, 255, 255, 0.9);
    border-radius: 16px;
    padding: 30px;
    max-width: 700px;
    margin: auto;
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
    backdrop-filter: blur(10px);
    transition: transform 0.3s ease;
}

.form-card:hover {
    transform: scale(1.01);
}

.checkbox-container {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
    gap: 15px;
    margin-top: 20px;
    margin-bottom: 20px;
}

.checkbox-item {
    font-size: 16px;
    padding: 12px 16px;
    background-color: #f1f5fb;
    border: 1px solid #ccd9f0;
    border-radius: 10px;
```

```
    transition: all 0.3s ease;
    cursor: pointer;
    display: flex;
    align-items: center;
    gap: 10px;
    user-select: none;
}

.checkbox-item:hover {
    background-color: #e6f0ff;
    box-shadow: 0 4px 12px rgba(52, 152, 219, 0.2);
}

.checkbox-item input[type="checkbox"] {
    transform: scale(1.2);
    accent-color: #3498db;
}

.styled-button {
    background-color: #3498db;
    color: white;
    border: none;
    padding: 12px 24px;
    font-size: 16px;
    border-radius: 8px;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease;
    box-shadow: 0 4px 12px rgba(52, 152, 219, 0.3);
}

.styled-button:hover {
    background-color: #2c80b4;
    transform: translateY(-2px);
}
.card-section {
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 30px;
    margin-top: 40px;
}

.summary-card,
.chart-card {
    background: rgba(255, 255, 255, 0.8);
    border-radius: 16px;
    padding: 20px;
    max-width: 700px;
    width: 90%;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.1);
```

```
    backdrop-filter: blur(10px);
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.charts-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(320px, 1fr));
    gap: 24px;
    width: 100%;
    padding: 0 20px;
}

.chart-card img {
    width: 100%;
    border-radius: 12px;
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.interactive-card:hover,
.interactive-card:focus-within {
    transform: scale(1.03);
    box-shadow: 0 12px 25px rgba(0, 0, 0, 0.15);
}

.chart-card:hover img,
.chart-card:focus-within img {
    transform: scale(1.45);
}
```

- **Explanition:**

1. Global and Layout Styles

- body styles set a clean font (Segoe UI), light gradient background, centered text, and padding for spacious layout.
- h1, h2 headings are styled with dark shades for contrast and clarity.

2. Form Styling

- .form-card creates a **glassmorphic card** effect with:
  - White translucent background (rgba)
  - Rounded corners
  - Shadow and blur for a soft, modern look
  - Hover effect to slightly enlarge the card (transform: scale(1.01))
- form and .form-card are centered, with max width and padding for good visual balance.

3. Checkbox Styling

- .checkbox-container arranges checkboxes in a **responsive grid**, adapting to screen size using auto-fit and minmax.
- Each .checkbox-item:
  - Has padding, border, and background color for card-like appearance.
  - Uses display: flex for alignment with icons.
  - Includes a hover effect that changes background and adds subtle shadow.

- Scales the checkbox using transform: scale(1.2) and colors it using accent-color.

4. Button Styling

- .styled-button is visually enhanced with:
  - Rounded edges
  - Bold color (#3498db)
  - Hover effect with darker shade and upward movement
  - Drop shadow for depth

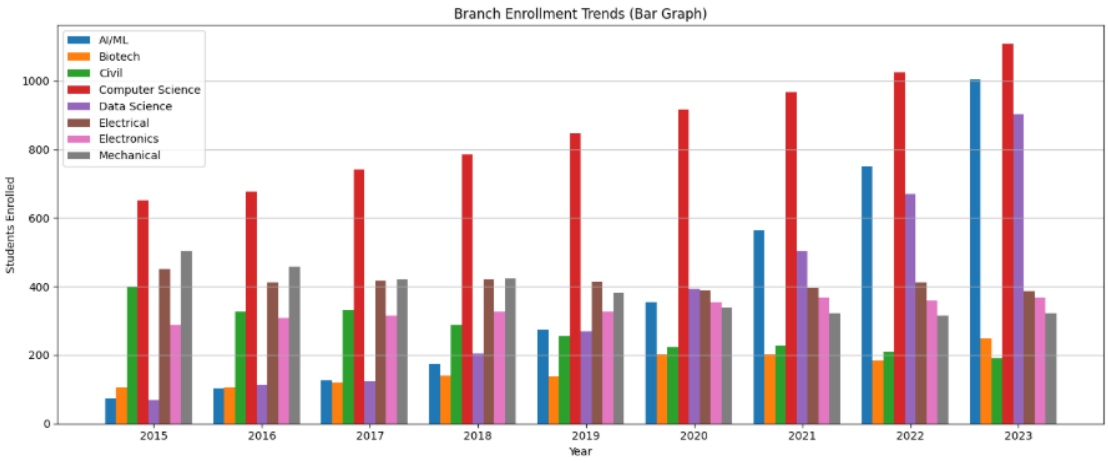
5. Card and Section Layout

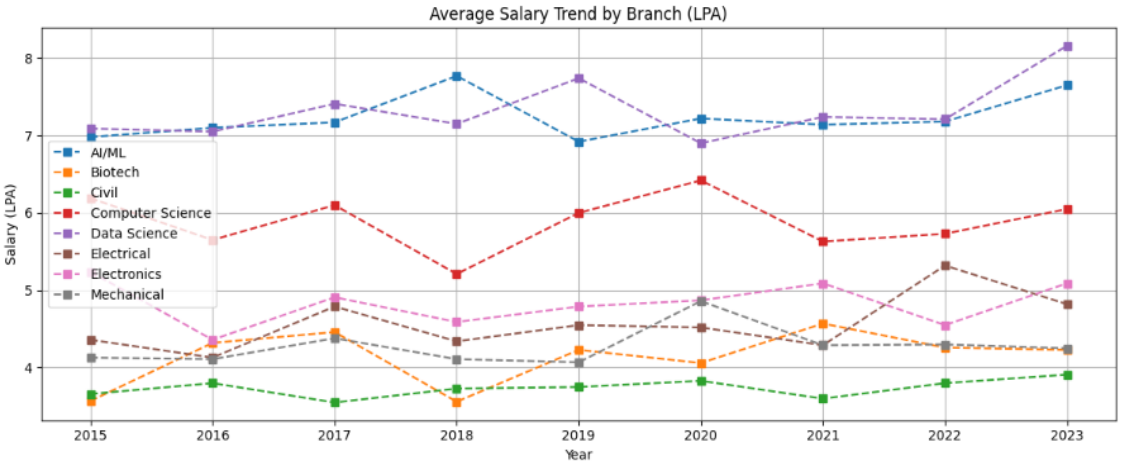
- .card-section vertically stacks summary and chart cards, with spacing.
- .summary-card and .chart-card both:
  - Share a frosted glass look
  - Have blur and shadow for an elevated, modern feel
  - Animate on hover for interactivity

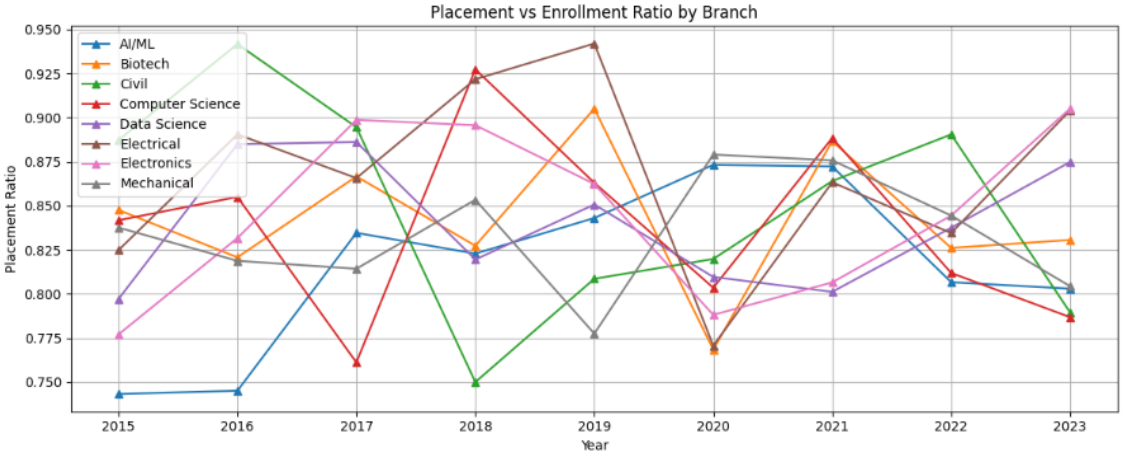
6. Charts and Image Display

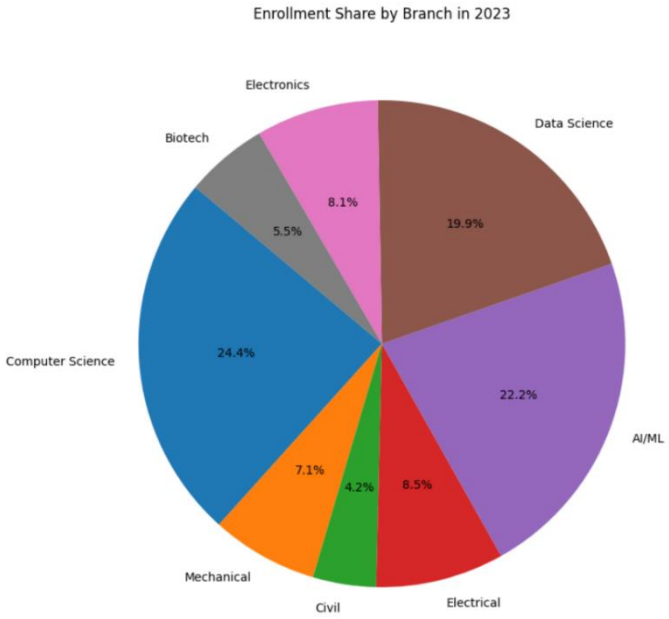
- .charts-grid displays charts in a flexible grid with spacing.
- Chart images are:
  - Full width inside cards
  - Rounded corners
  - Enlarged on hover (transform: scale(1.45)) for better readability
  - Combined with .interactive-card:hover to create a lifting animation

Output Screenshots:




Average Salary Trend by Branch (LPA)
Salary (LPA)
Year
AI/ML
Biotech
Civil
Computer Science
Data Science
Electrical
Electronics
Mechanical
2015
2016
2017
2018
2019
2020
2021
2022
2023


Placement vs Enrollment Ratio by Branch
Placement Ratio
Year
AI/ML
Biotech
Civil
Computer Science
Data Science
Electrical
Electronics
Mechanical


Enrollment Share by Branch in 2023
Electronics
Data Science
Biotech
8.1%
19.9%
5.5%
24.4%
Computer Science
22.2%
AI/ML
7.1%
4.2%
8.5%
Mechanical
Civil
Electrical

📋 Summary Report (2023)

- 🔼 Highest Enrollment: Computer Science (1107)
- 🔽 Lowest Enrollment: Civil (190)
- 💰 Highest Salary: Data Science (8.16 LPA)
- 🎯 Best Placement Ratio: Electronics (0.91)
- ⚠️ Worst Placement Ratio: Computer Science (0.79)

Conclusion:

The analysis reveals significant shifts in student preferences and market demand within engineering education.

- **Computer Science** continues to attract the highest number of students but suffers from **low placement efficiency**, signaling market saturation.
- **Data Science and AI/ML** show strong promise with **high salaries and growing enrollments**, making them attractive options for future aspirants.
- **Electronics Engineering**, though less popular, offers the **best placement ratio**, indicating a hidden opportunity.
- **Traditional branches** like Civil and Mechanical face declining interest and relatively poor job outcomes, suggesting a need for modernization or curriculum overhaul.

Overall, this analysis highlights the importance of aligning **student choices, institutional focus**, and **industry demand** to ensure sustainable academic and career outcomes in the engineering domain.

Bibliography:

- Python: https://docs.python.org/3/
- Pandas: https://pandas.pydata.org/
- Matplotlib: https://matplotlib.org/
- Flask: https://flask.palletsprojects.com/
- Data Source : Custom-generated for educational analysis