ENGINEERING BRANCHES REPORT

Table of Contents

Topic Name

Engineering Branch Demand Analysis

Problem Statement

Engineering institutions often lack a clear, visual understanding of demand trends for various engineering branches. This project solves that problem by offering an interactive data visualization tool that highlights student interest over years, along with placement and salary insights, enabling data-driven decisions for colleges and administrators.

Description

This project provides an interactive web-based dashboard using Streamlit and Plotly to analyze:

- Engineering branch trends from 2005 to 2025
- Latest years most and least in-demand branches
- Branches with the most growth/decline
- Placement ratios and average salary per branches
- It enables exploration year-wise and visualizes data using pie charts and bar graphs.

Data and Output Purpose

Input Data (CSV):

- year
- branch
- studentcount
- placement_ratio
- average_salary

Output Purpose and Benefits:

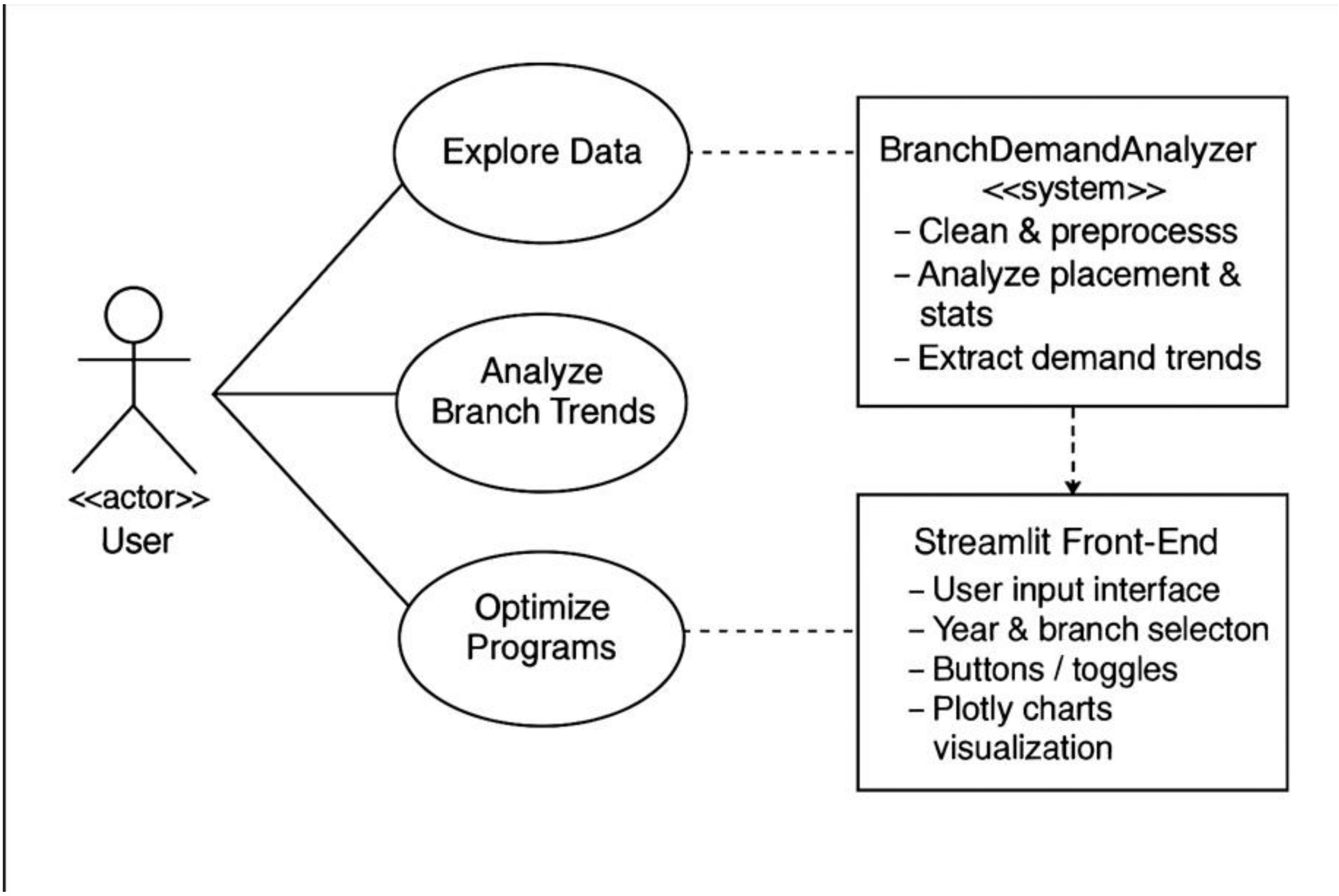
- Identify most/least preferred branches each year
- Discover trends over time
- Inform faculty and admissions strategy
- Correlate student demand with placement/salary outcomes.

Solution Plan

1. Data Ingestion from CSV
2. Analysis via BranchDemandAnalyzer class
3. Streamlit UI with buttons and charts
4. Visualization by year (Pie and Bar charts)

5. Placement and Salary analysis
6. Toggle functionality with session state

Diagram Design



Actor: user

This is the person interacting with the system. They could be an admin, data analyst, placement officer, or even a student interested in trends. The actor initiates all actions.

Use cases

These are the tasks the user performs through the system:

- **Explore Data**: Users can browse demand stats across branches and years.
- **Analyze Branch Trends**: Users request analysis of year-over-year changes.
- **Optimize Programs**: Users make decisions based on trend reports (e.g., which branches to promote or scale back).

These use cases highlight the functional goals from the user's perspective.

System Component: **BranchDemandAnalyzer**

This is the backend Python class that does the heavy lifting. Its responsibilities include:

- **Clean & preprocess**: Handling missing data, converting data types.
- **Stats & trends**: Calculating current demand, growth rates.
- **Placement analysis**: Optional future extension—linking demand with placement stats.

This component maps to your object-oriented class structure (which you've already implemented!).

Interface Layer: **Streamlit Front-End**

This is the user-facing dashboard. It offers:

- **Year selection**: Users choose which years to analyze.
- **Buttons & toggles**: Controls for what kind of analysis to run.
- **Plotly charts**: Interactive visualizations of demand data.

It translates backend functionality into an intuitive user experience.

`User Decision Making`

This final block captures the **goal** of the system:

- **Explore demand trends**: Understand which branches are popular or declining.
- **Optimize programs**: Make informed decisions—such as starting a new program or revamping an existing one.

Implementation

Tools: Python, pandas, plotly, streamlit

Files:

- app.py (Streamlit interface)
- analyzer.py (Analysis class)
- Engineering_Students_Data.csv (Dataset)

Code & Explanation

analyzer.py

```
import pandas as pd
from scipy.stats import linregress

class BranchDemandAnalyzer:
    def __init__(self, data):
        self.data = data.copy()
        # Don't read CSV here—just store the DataFrame
        if not isinstance(data, pd.DataFrame):
            raise TypeError(" BranchDemandAnalyzer expects a pandas DataFrame!")

        self.data = data.copy()
        self.latest_year = None
        self.latest_data = None
        self.trends = {}
        self.highest_demand_branch = None
        self.lowest_demand_branch = None
        self.branch_most_growth = None
        self.branch_most_decline = None
        self.placement_stats = None

    def clean_data(self):
        required_columns = ['year', 'branch', 'studentcount']
        missing = [col for col in required_columns if col not in self.data.columns]
        if missing:
            raise ValueError(f" Missing columns in CSV: {missing}")

        self.data.dropna(subset=required_columns, inplace=True)
        self.data['year'] = self.data['year'].astype(int)
        self.data['studentcount'] = self.data['studentcount'].astype(int)


        # Handle optional placement and salary columns

        if 'placement_ratio' in self.data.columns:
            self.data['placement_ratio'] =
self.data['placement_ratio'].fillna(0).astype(float)
        if 'avg_salary' in self.data.columns:
            self.data['avg_salary'] = self.data['avg_salary'].fillna(0).astype(float)
    def get_latest_year_data(self):
        self.latest_year = self.data['year'].max()
        self.latest_data = self.data[self.data['year'] == self.latest_year]

    def analyze_current_demand(self):
        if self.latest_data.empty:
            raise ValueError(" No data for the latest year!")
        highest_row = self.latest_data.loc[self.latest_data['studentcount'].idxmax()]
        lowest_row = self.latest_data.loc[self.latest_data['studentcount'].idxmin()]
        self.highest_demand_branch = highest_row['branch']
        self.lowest_demand_branch = lowest_row['branch']

    def analyze_trends(self):
        branches = self.data['branch'].unique()
        self.trends = {}
        for branch in branches:
            branch_data = self.data[self.data['branch'] == branch]
            if len(branch_data) < 2:
                self.trends[branch] = None
                continue
            slope, *_ = linregress(branch_data['year'], branch_data['studentcount'])
            self.trends[branch] = slope

        trends_clean = {k: v for k, v in self.trends.items() if v is not None}
        if trends_clean:
            self.branch_most_growth = max(trends_clean, key=trends_clean.get)
            self.branch_most_decline = min(trends_clean, key=trends_clean.get)

    def analyze_placement_and_salary(self):
        if 'placement_ratio' not in self.data.columns or 'avg_salary' not in
self.data.columns:
            raise ValueError(" CSV missing 'placement_ratio' or 'avg_salary' columns.")
        self.placement_stats = self.data.groupby('branch').agg({
            'placement_ratio': 'mean',
            'avg_salary': 'mean'
        }).reset_index()
```

Explanation of analyzer.py:

__init__(self,data):

Purpose: Initializes the analyzer with a dataset.

1. Type Checking**:**
    - Verifies that the `data` passed is a `pandas.DataFrame`.
    - If not, it raises a `TypeError` for safety.
2. Stores a copy of the dataset internally to avoid modifying the original.
3. Initializes attributes**:**
    - `latest_year`, `latest_data`: Used to store filtered data for the most recent year.
    - `trends`: Dictionary for slope (growth/decline) per branch.
    - `highest_demand_branch`, `lowest_demand_branch`: Identified during analysis.
    - `branch_most_growth`, `branch_most_decline`: Branches showing strongest upward/downward trend.
    - `placement_stats`: Stores placement & salary analysis result.

clean_data(self):

Purpos**e**: Cleans and preprocesses the data to ensure it's ready for analysis.

1. Checks for required columns**:**
    - Ensures `['year', 'branch', 'studentcount']` are present.
    - If any are missing, it raises a `ValueError`.
2. Drops rows with missing values in required columns using `dropna`.
3. Converts data types**:**
    - `year` → `int`
    - `studentcount` → `int`
    - Ensures consistent numeric formats.
4. Handles optional columns**:**
    - If `placement_ratio` exists:
        - Fills missing values with `0` and converts to float.
    - Same for `avg_salary`.

get_latest_year_data(self):

Purpose**:** Isolates the data corresponding to the most recent year.

1. Finds the latest year using `max()` on the `'year'` column.
2. Filters the dataset to include only rows for that year.
3. Stores
    - The year in `self.latest_year`
    - The filtered data in `self.latest_data`

analyze_current_demand(self):

Purpose**:** Determines which branches currently have the highest and lowest demand (i.e., most and fewest students).

1. Checks if **`latest_data`** is empty**:**
    - If there's no data for the latest year, raises an error.

2. Finds the row with the max student count (`idxmax`).
    - Saves its `branch` value to `self.highest_demand_branch`.
3. Finds the row with the min student count (`idxmin`).
    - Saves its **`branch`** value to **`self.lowest_demand_branch.`**

analyze_trends(self):

Purpose: Analyzes how demand for each branch has changed over the years.

1. Loops through each unique branch in the data.
2. For each branch:
    - Filters only its data.
    - If fewer than 2 data points exist, skips trend analysis.
    - Otherwise:
        - Applies `scipy.stats.linregress()` to compute the **slope** of student count vs. year.
3. Stores the slope (rate of change) for each branch in `self.trends`.
4. Identifies**:**
    - `branch_most_growth`: Branch with the steepest positive slope (rising popularity).
    - `branch_most_decline`: Branch with steepest negative slope (falling popularity).

Interpretation**:**

- Positive slope → growing demand.
- Negative slope → declining demand

analyze_placement_and_salary(self):

Purpose**:** Analyzes average placement success and average salary for each branch.

1. Checks if columns **`placement_ratio`** and **`avg_salary`** exist**.**
    - If not, raises a `ValueError`.
2. Groups data by **`branch`** and calculates:
    - Mean placement ratio
    - Mean salary
3. Stores result in **`self.placement_stats`** as a DataFrame.

analyzer.py

- **`clean_data()`:** Ensures types and handles missing values
- **`get_latest_year_data()`:** Pulls the most recent year
- **`analyze_current_demand()`:** Finds most/least demanded branches
- **`analyze_trends()`:** Uses linear regression to find demand slope per branch
- **`analyze_placement_and_salary()`:** Averages placement & salary by branch

app.py

```
import streamlit as st
import pandas as pd
import plotly.express as px
from analyzer import BranchDemandAnalyzer

# PAGE CONFIG

st.set_page_config(page_title="Engineering Branch Analysis", layout="wide")
st.title(" Engineering Branch Demand Analysis")

# LOAD DATA

DATA_PATH = "hackathon\Engineering_Students_Data.csv"

try:
    data = pd.read_csv(DATA_PATH)
    st.success(f" Loaded data from {DATA_PATH}")
except FileNotFoundError:
    st.error(f" File not found: {DATA_PATH}")
    st.stop()

analyzer = BranchDemandAnalyzer(data)

# Show Raw Data Checkbox

if st.checkbox("Show Raw Data"):
    st.subheader(" Raw Data")
    st.dataframe(analyzer.data)

# Clean Data Button

if st.button(" Clean Data"):
    try:
        analyzer.clean_data()
        st.success(" Data cleaned successfully!")
        st.dataframe(analyzer.data)
    except Exception as e:
        st.error(f" Error: {e}")

# Basic Statistics Button

if st.button(" Show Basic Statistics"):
    st.subheader(" Descriptive Statistics")
    st.dataframe(analyzer.data.describe()
# Latest Year Demand Analysis

if st.button("Analyze Latest Year Demand"):
    try:
        analyzer.get_latest_year_data()
        analyzer.analyze_current_demand()

        st.subheader(f" Latest Year: {analyzer.latest_year}")
        st.dataframe(analyzer.latest_data)

        st.success(f" Highest Demand Branch: {analyzer.highest_demand_branch}")
        st.warning(f" Lowest Demand Branch: {analyzer.lowest_demand_branch}")
    except Exception as e:
        st.error(f" Error: {e}")

# Analyze Branch Trends
if st.button(" Analyze Branch Trends"):
    try:
        analyzer.analyze_trends()

        st.subheader(" Demand Trends (Slope per Branch)")
        st.dataframe(pd.DataFrame(list(analyzer.trends.items()), columns=["Branch",
"Slope"]))

        if analyzer.branch_most_growth:
            st.success(f" Branch with Most Growth Potential: {analyzer.branch_most_growth}")

        if analyzer.branch_most_decline:
            st.error(f" Branch More Likely to Decline: {analyzer.branch_most_decline}")
    except Exception as e:
        st.error(f" Error: {e}")

st.header(" Visualize Branch Demand by Year")

#  Select year
available_years = sorted(analyzer.data['year'].unique())
selected_year = st.selectbox(" Select Year", available_years)

#  Filter for selected year
year_data = analyzer.data[analyzer.data['year'] == selected_year]

if year_data.empty:
    st.warning(" No data available for the selected year.")
else:
    st.success(f" Showing data for {selected_year}")

    #  Compute highest and lowest demand branch for this year
    highest_branch = year_data.loc[year_data['studentcount'].idxmax(), 'branch']
    lowest_branch = year_data.loc[year_data['studentcount'].idxmin(), 'branch']
    st.subheader(f" Demand in {selected_year}")
    st.markdown(f" **Highest-demand branch:** `{highest_branch}`")
    st.markdown(f" **Lowest-demand branch:** `{lowest_branch}`")

    #  Also show previous year's highest and lowest
    prev_year = selected_year - 1
    if prev_year in available_years:
        prev_data  = analyzer.data[analyzer.data['year'] == prev_year]
        prev_highest = prev_data.loc[prev_data['studentcount'].idxmax(), 'branch']
        prev_lowest = prev_data.loc[prev_data['studentcount'].idxmin(), 'branch']

        st.subheader(f" Demand in {prev_year}")
        st.markdown(f" **Highest-demand branch:** `{prev_highest}`")
        st.markdown(f" **Lowest-demand branch:** `{prev_lowest}`")
    else:
        st.info(f" No data found for previous year ({prev_year})")

    #  pie Chart
    st.subheader(f" Pie Chart of Branch Demand ({selected_year})")
    fig_pie = px.pie(
        year_data,
        names='branch',
        values='studentcount',
        title=f"Branch Share in {selected_year}",
        hole=0.3

    )
    fig_pie.update_layout(width=400, height=600)
    st.plotly_chart(fig_pie, use_container_width=True)

    #  Vertical Bar Chart
    st.subheader(f" Vertical Bar Chart of Branch Demand ({selected_year})")
    fig_bar = px.bar(
        year_data,
        x='branch',
        y='studentcount',
        color='branch',
        title=f"Branch Demand in {selected_year}"
    )
    fig_bar.update_layout(width=400, height=600)
    st.plotly_chart(fig_bar, use_container_width=True)



# placement and salary
if st.button(" Analyze Placement and Salary"):
    try:
        analyzer.analyze_placement_and_salary()
        st.subheader(" Average Placement Ratio and Salary by Branch")
        st.dataframe(analyzer.placement_stats)
        st.subheader(" Placement Ratio per Branch")
        fig1 = px.bar(
            analyzer.placement_stats,
            x='branch',
            y='placement_ratio',
            color='branch',
            title="Average Placement Ratio by Branch")
        fig1.update_layout(width=400, height=600)
        st.plotly_chart(fig1, use_container_width=True)
        st.subheader(" Average Salary per Branch")
        fig2 = px.bar(
            analyzer.placement_stats,
            x='branch',
            y='avg_salary',
            color='branch',
            title="Average Salary by Branch")
        fig2.update_layout(width=400, height=600)
        st.plotly_chart(fig2, use_container_width=True)
    except Exception as e:
        st.error(f" Error: {e}")

         st.header(" Final Summary: Branch Demand Insights")
         try:
              analyzer.get_latest_year_data()
              analyzer.analyze_current_demand()
              analyzer.analyze_trends()
          col1, col2 = st.columns(2)
          with col1:
        st.markdown(f"<h4 style='color: green;'> Latest Year Analyzed:
{analyzer.latest_year}</h4>", unsafe_allow_html=True)
        st.markdown(f"<h4 style='color: blue;'> Highest Demand Branch:
{analyzer.highest_demand_branch}</h4>", unsafe_allow_html=True)
        st.markdown(f"<h4 style='color: orange;'> Lowest Demand Branch:
{analyzer.lowest_demand_branch}</h4>", unsafe_allow_html=True)

    with col2:
        st.markdown(f"<h4 style='color: green;'> Most Likely to Grow:
{analyzer.branch_most_growth}</h4>", unsafe_allow_html=True)
        st.markdown(f"<h4 style='color: red;'> Most Likely to Decline:
{analyzer.branch_most_decline}</h4>", unsafe_allow_html=True)

except Exception as e:
    st.error(f" Failed to generate summary: {e}")
```

Expalantion of app.py

`st.set_page_config(...)`

- Purpose: Sets the basic configuration of the Streamlit web app.
- Parameters:
  - `page_title`: Sets the title of the browser tab.
  - `layout='wide'`: Allows the app to use full screen width (better for data dashboards).

St.title(…)

- Purpose: Displays a large title at the top of the web page.
- Usage: Here it shows `"Engineering Branch Demand Analysis"` as the main heading.

Data Loading:

- Purpose: Loads the engineering student data from a CSV file.
- Error Handling: If the file is missing, `FileNotFoundError` is caught and an error message is displayed with `st.error`, then execution is stopped with `st.stop()`.

BranchDemandAnalyzer(data)

- Purpose: Instantiates a custom class (you've defined in `analyzer.py`) that encapsulates the logic for cleaning, analyzing, and visualizing the data.

st.checkbox("Show Raw Data")

- Purpose: Provides an optional checkbox to show the raw loaded data.
- Action: If checked, displays the full DataFrame (`analyzer.data`) in a scrollable table.

st.button("Clean Data")

- Purpose: When clicked, triggers the `clean_data()` method from `BranchDemandAnalyzer`.
- Functionality:
  - Cleans missing values or inconsistent entries.
  - Updates the internal DataFrame in the analyzer instance.
  - Displays the cleaned data if successful.
  - Displays errors if anything fails

st.button("Show Basic Statistics")

- Purpose: Triggers a summary of numerical data in the DataFrame.
- Details: Uses `pandas.DataFrame.describe()` to show:
  - Mean, std dev, min, max, and percentiles for numerical columns like `studentcount`, `salary`, etc.

st.button("Analyze Latest Year Demand")

- Purpose**:** Focuses on data from the most recent year.
- Steps Involved**:**
    1. `analyzer.get_latest_year_data()` – extracts data only for the latest year.
    2. `analyzer.analyze_current_demand()` – computes the branch with:
        - Highest student count (demand).
        - Lowest student count (low demand).
- Displays**:**
    - Year being analyzed.
    - DataFrame for that year.
    - Highlight of the most and least demanded branches.

st.button("Analyze Branch Trends")

- Purpose**:** Analyzes how demand for each branch changes over time.
- Steps**:**
    1. `analyzer.analyze_trends()`:
        - Computes a linear regression slope of demand (student count) over years for each branch.
        - Positive slope → rising demand.
        - Negative slope → declining demand.
- Displays**:**
    - A DataFrame of slopes per branch.
    - Names of branches with:
        - Most growth potential (highest positive slope).
        - Most likely to decline (most negative slope).

Year-wise Branch Demand Visualization

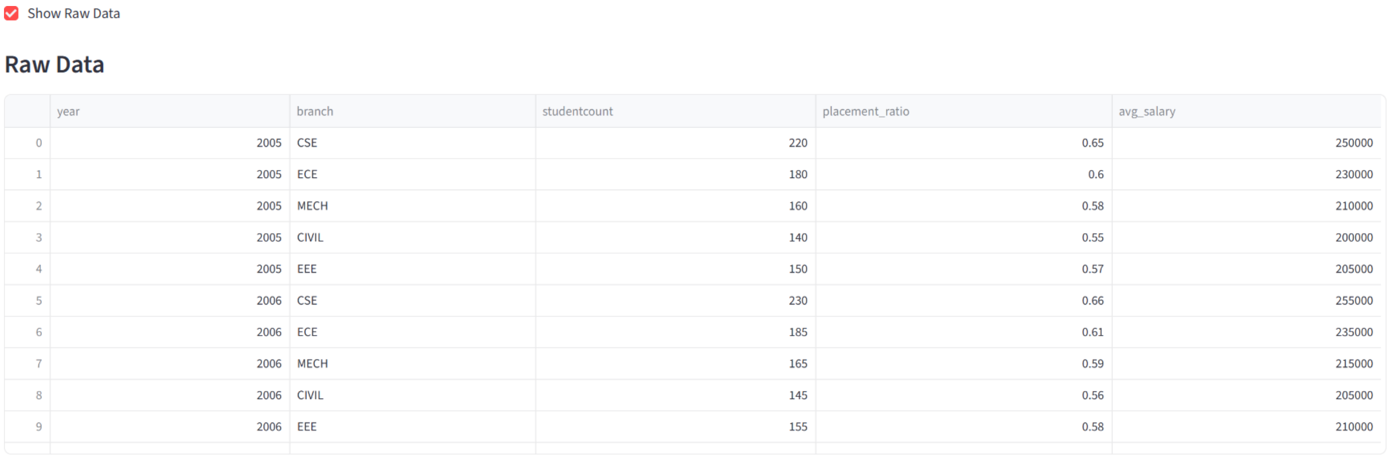
- Purpose: Lets the user choose a year and explore demand for branches in that year.
- Components**:**
    - `st.selectbox(...)`: Dropdown to pick a year.
    - Filters data for that year using `year_data = ...`.
    - Displays highest and lowest demand branches.
    - Optionally compares it with the previous year.
    - Two visualizations:
        - Pie Chart: Shows the percentage share of each branch.
        - Bar Chart: Vertical bar chart comparing student count across branches.

st.button("Analyze Placement and Salary")

- Purpose: Explores placement performance and average salary per branch.
- Steps**:**
    1. `analyzer.analyze_placement_and_salary()`:
        - Computes:
            - Placement Ratio = students placed / total students.
            - Average Salary per branch.
        - Stores this in `analyzer.placement_stats`.
- Visual Output**:**
    - Displays the `placement_stats` table.
    - Bar charts for:
        - Placement Ratio per Branch.
        - Average Salary per Branch.

Output Screenshots

View raw data

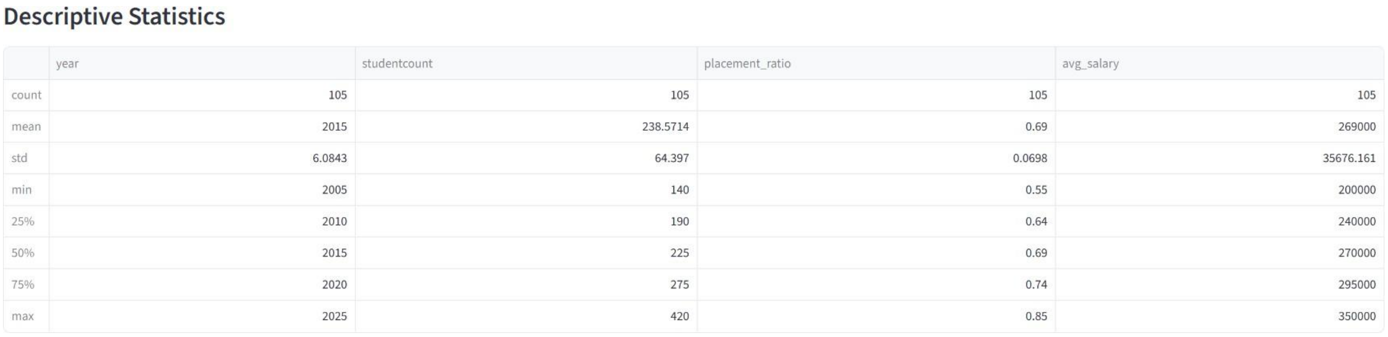


☑ Show Raw Data

Raw Data

	year	branch	studentcount	placement_ratio	avg_salary
0	2005	CSE	220	0.65	250000
1	2005	ECE	180	0.6	230000
2	2005	MECH	160	0.58	210000
3	2005	CIVIL	140	0.55	200000
4	2005	EEE	150	0.57	205000
5	2006	CSE	230	0.66	255000
6	2006	ECE	185	0.61	235000
7	2006	MECH	165	0.59	215000
8	2006	CIVIL	145	0.56	205000
9	2006	EEE	155	0.58	210000

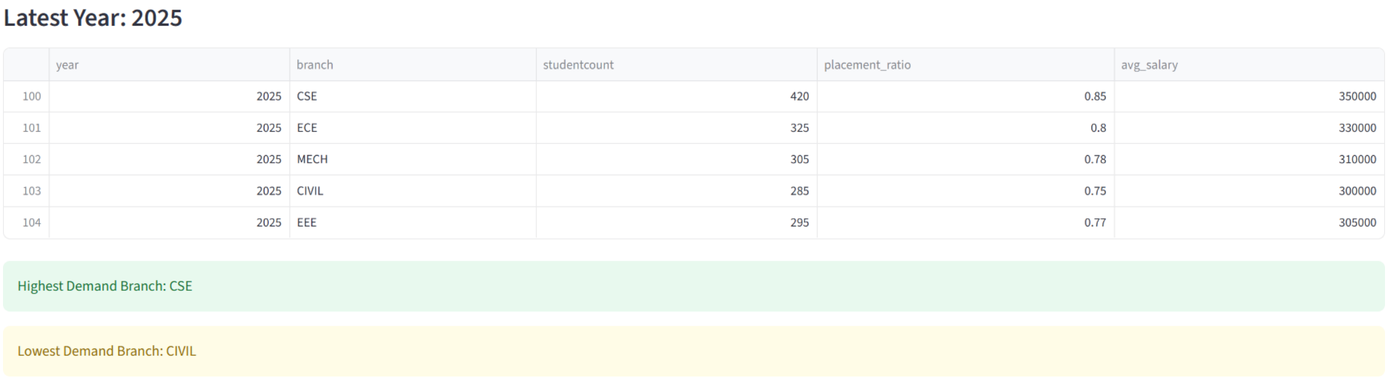
Descriptive stats



Descriptive Statistics

	year	studentcount	placement_ratio	avg_salary
count	105	105	105	105
mean	2015	238.5714	0.69	269000
std	6.0843	64.397	0.0698	35676.161
min	2005	140	0.55	200000
25%	2010	190	0.64	240000
50%	2015	225	0.69	270000
75%	2020	275	0.74	295000
max	2025	420	0.85	350000

Analyze latest year demands

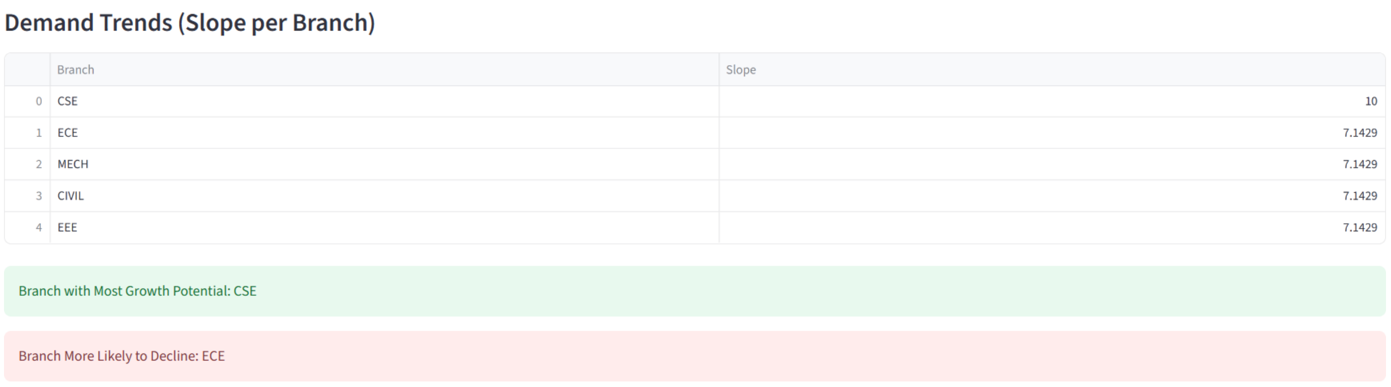


Latest Year: 2025

	year	branch	studentcount	placement_ratio	avg_salary
100	2025	CSE	420	0.85	350000
101	2025	ECE	325	0.8	330000
102	2025	MECH	305	0.78	310000
103	2025	CIVIL	285	0.75	300000
104	2025	EEE	295	0.77	305000

Highest Demand Branch: CSE

Lowest Demand Branch: CIVIL



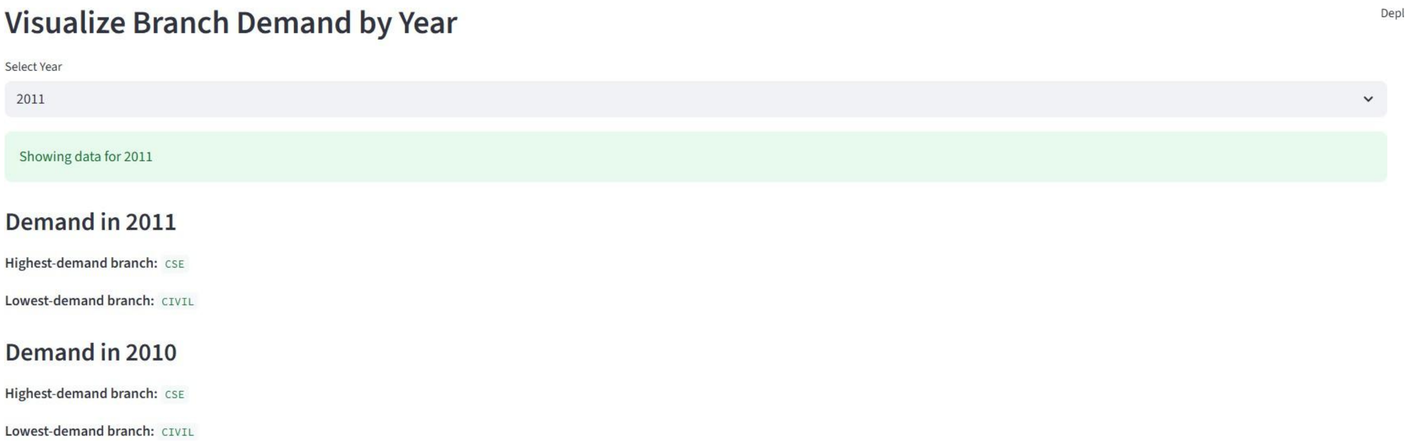
Demand Trends (Slope per Branch)

	Branch	Slope
0	CSE	10
1	ECE	7.1429
2	MECH	7.1429
3	CIVIL	7.1429
4	EEE	7.1429

Branch with Most Growth Potential: CSE

Branch More Likely to Decline: ECE

Branch demand by year



Visualize Branch Demand by Year

Depl

Select Year

2011

Showing data for 2011

Demand in 2011

Highest-demand branch: `CSE`

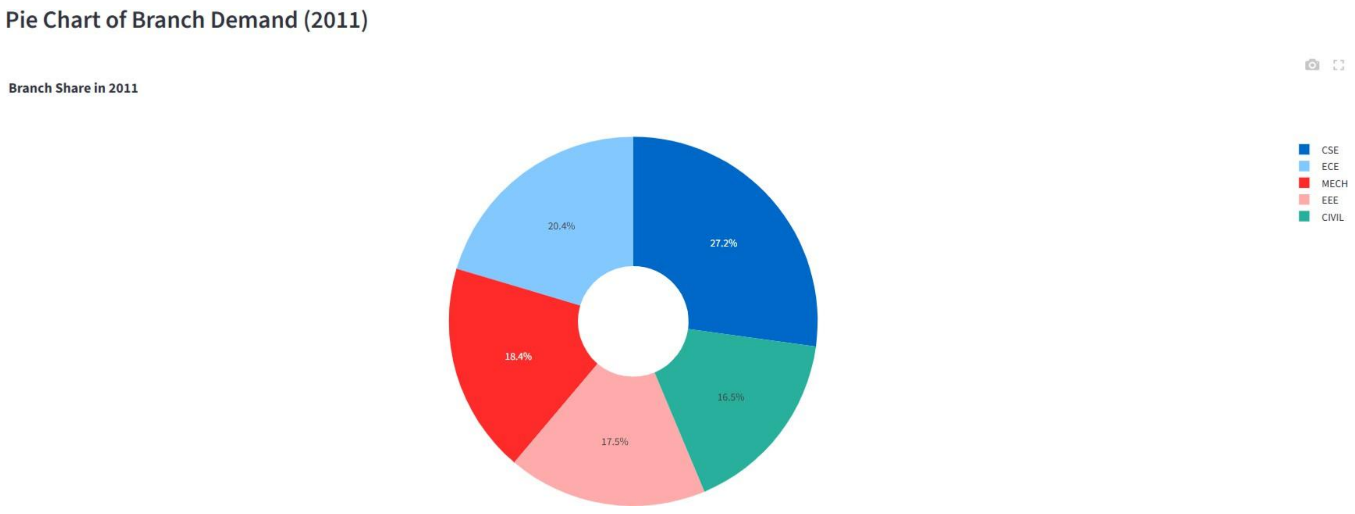
Lowest-demand branch: `CIVIL`

Demand in 2010

Highest-demand branch: `CSE`

Lowest-demand branch: `CIVIL`

piechart of the selected year

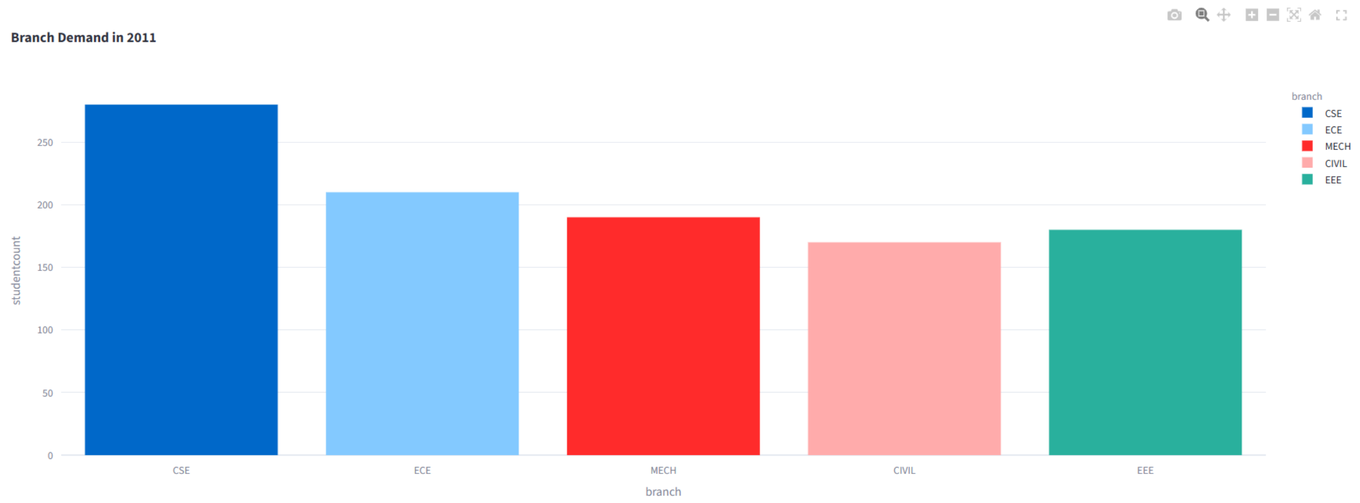


Pie Chart of Branch Demand (2011)

Branch Share in 2011

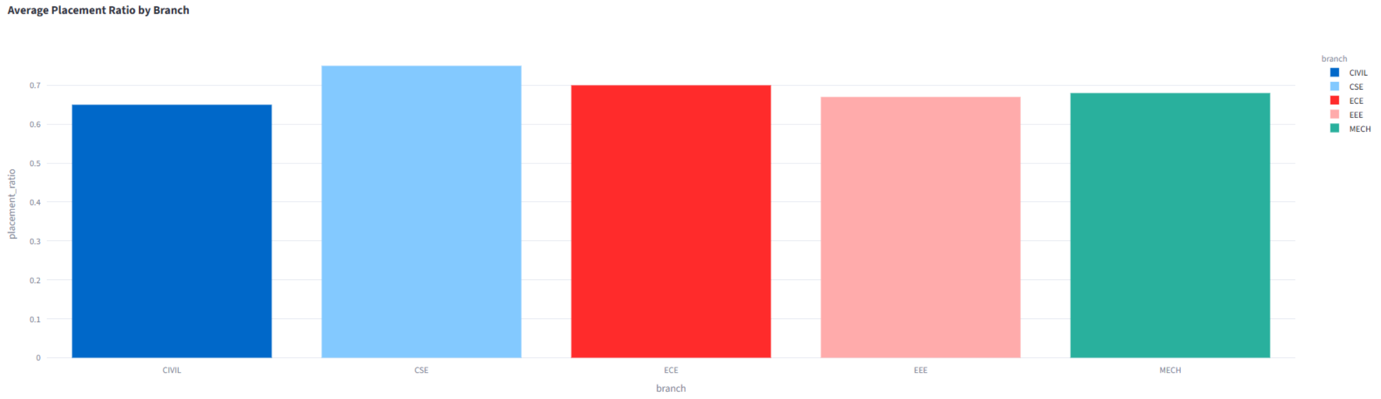
Vertical bar graph of the selected year

Vertical Bar Chart of Branch Demand (2011)

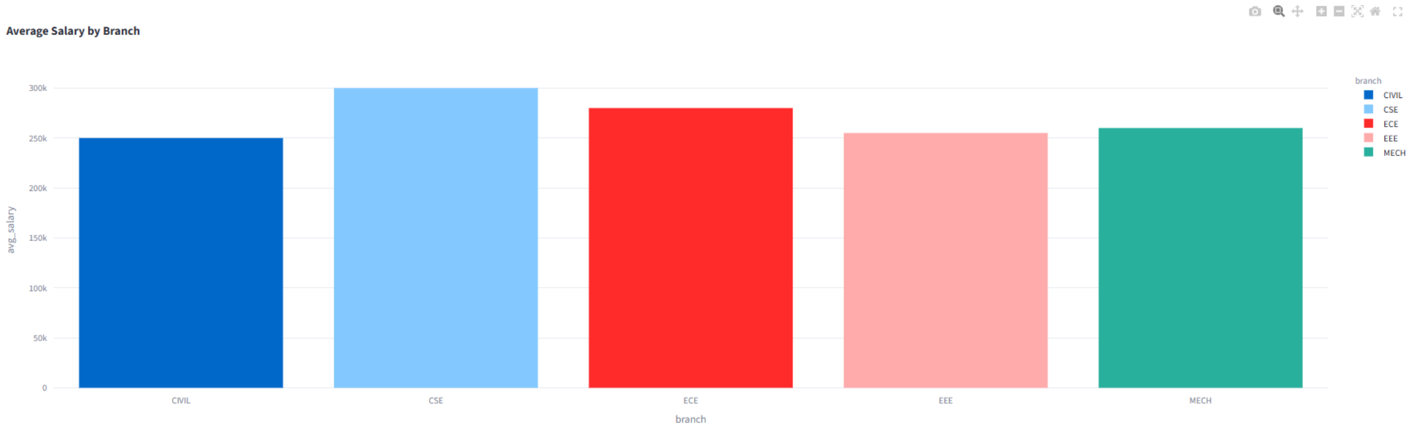


Placement & Salary charts

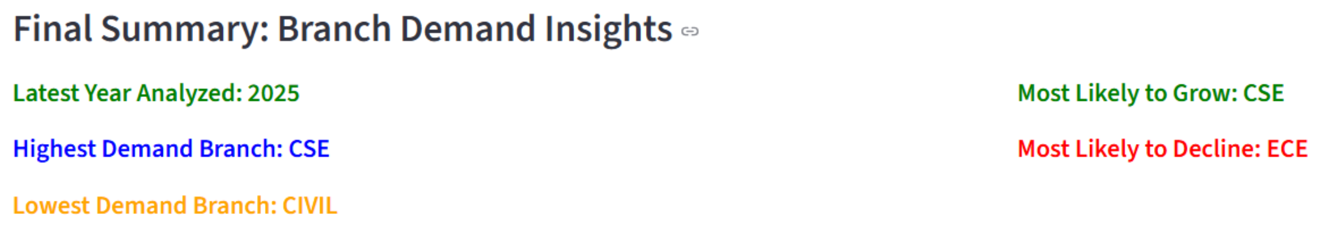
Placement Ratio per Branch



Average Salary per Branch



Summary for latest year



Final Summary: Branch Demand Insights

Latest Year Analyzed: 2025

Highest Demand Branch: CSE

Lowest Demand Branch: CIVIL

Most Likely to Grow: CSE

Most Likely to Decline: ECE

Conclusion

The analysis highlights a clear shift in student interest toward **technology-focused branches**, especially those aligned with emerging fields like AI, Data Science, and Cybersecurity. Traditional branches like **Mechanical**, **Civil**, and **Biotechnology** show either stagnation or decline in demand, indicating the need for curriculum upgrades or better industry integration.

Overall, this analysis can help:

- **Students** make informed career choices
- **Colleges** align their offerings with current trends
- **Policymakers** prioritize resource allocation toward in-demand fields

Closure

This project delivers an interactive, data-driven solution for understanding engineering education trends. It is built entirely with Python and is easy to maintain, extend, and deploy for institutions to support curriculum planning and admissions.

Bibliography

- Streamlit Documentation: https://docs.streamlit.io/

- Pandas Documentation: https://docs.pandas.pydata.org/

- Python Documentation: https://docs.python.org/

- Plotly Express: https://plotly.com/python/plotly-express/

- SciPy linregress: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html

- Engineering dataset created/simulated for academic visualization.