**ENGINEERING BRANCHES**

**REPORT**

Table of Contents

[Topic Name 3](#_Toc203650453)

[Problem Statement 3](#_Toc203650454)

[Description 3](#_Toc203650455)

[Data and Output Purpose 3](#_Toc203650456)

[Solution Plan 3](#_Toc203650457)

[Diagram Design 4](#_Toc203650458)

[Code & Explanation 6](#_Toc203650459)

[Output Screenshots 16](#_Toc203650460)

[16](#_Toc203650461)

[Conclusion 19](#_Toc203650462)

[Closure 19](#_Toc203650463)

[Bibliography 19](#_Toc203650464)

# 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

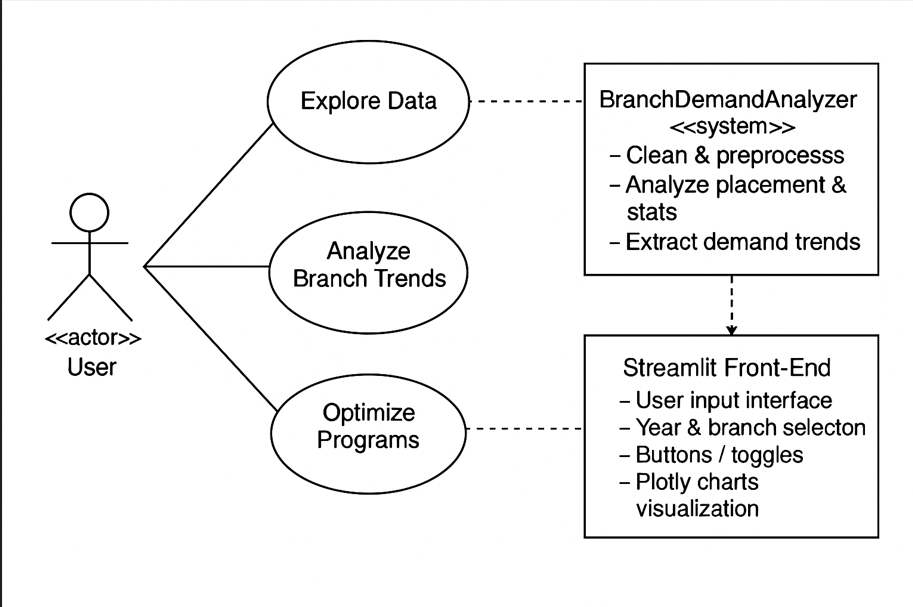
Output Purpose and Benefits:

* 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):**

**Purpose**: 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:**
  1. Year being analyzed.
  2. DataFrame for that year.
  3. 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:**
  1. A DataFrame of slopes per branch.
  2. 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:**
  1. Displays the placement\_stats table.
  2. Bar charts for:
     + Placement Ratio per Branch.
     + Average Salary per Branch.

# Output Screenshots

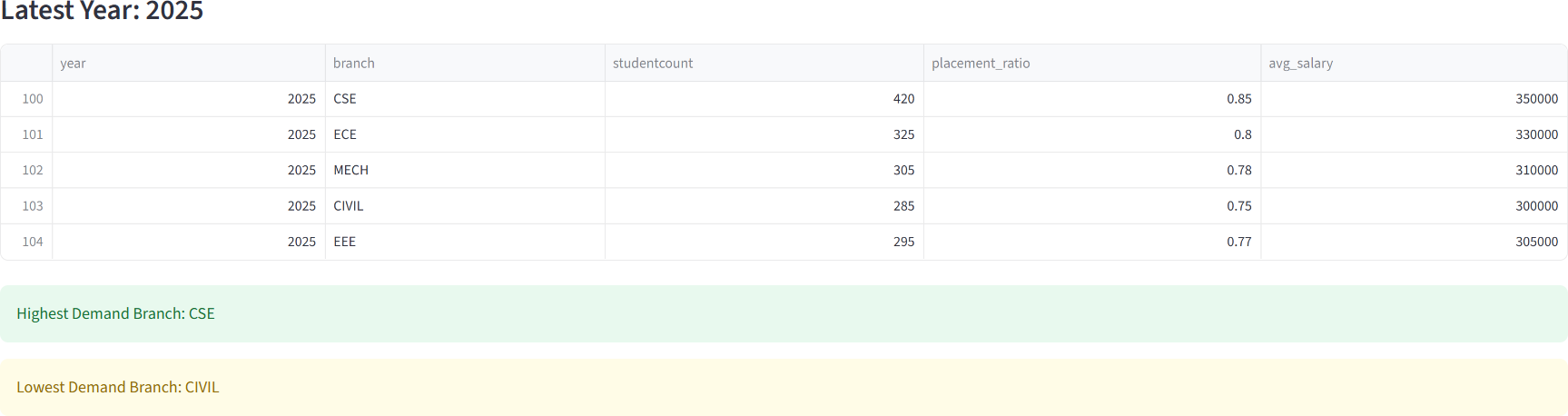
View raw data

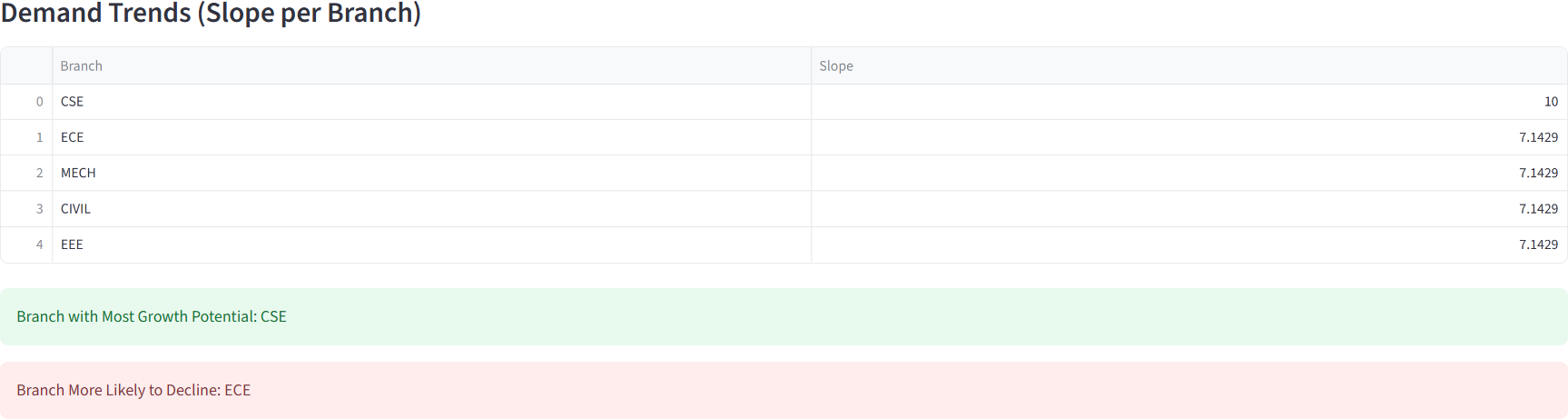
# 

Descriptive stats

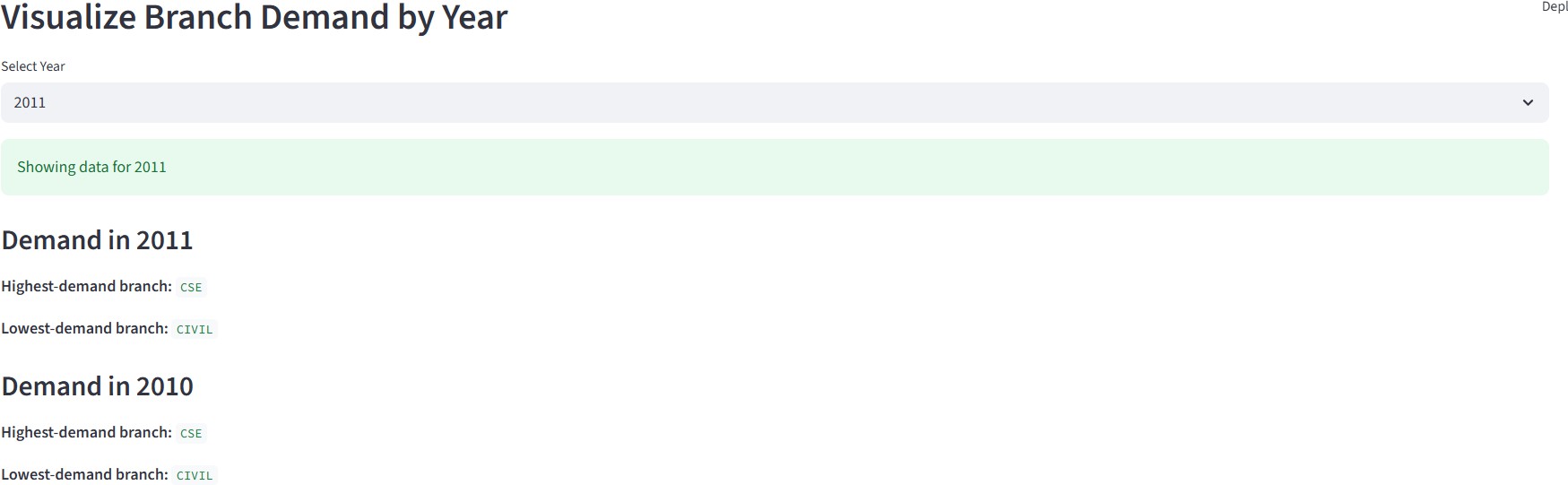


Analyze latest year demands

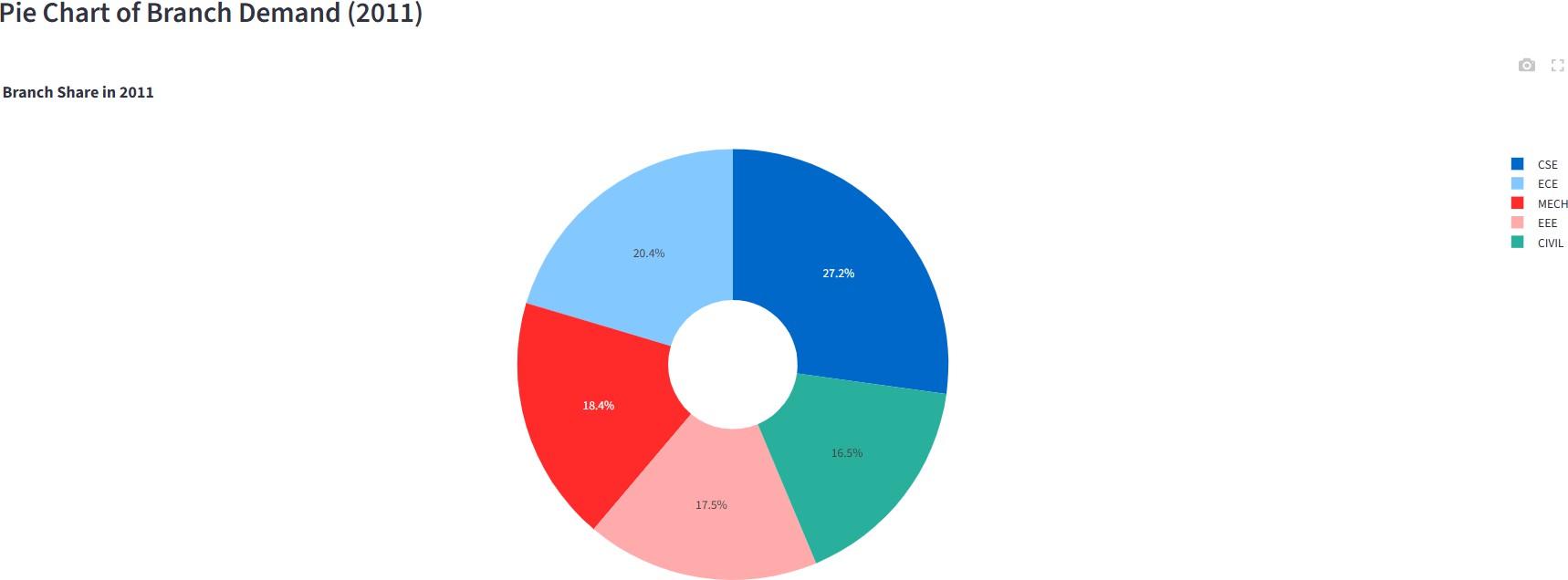




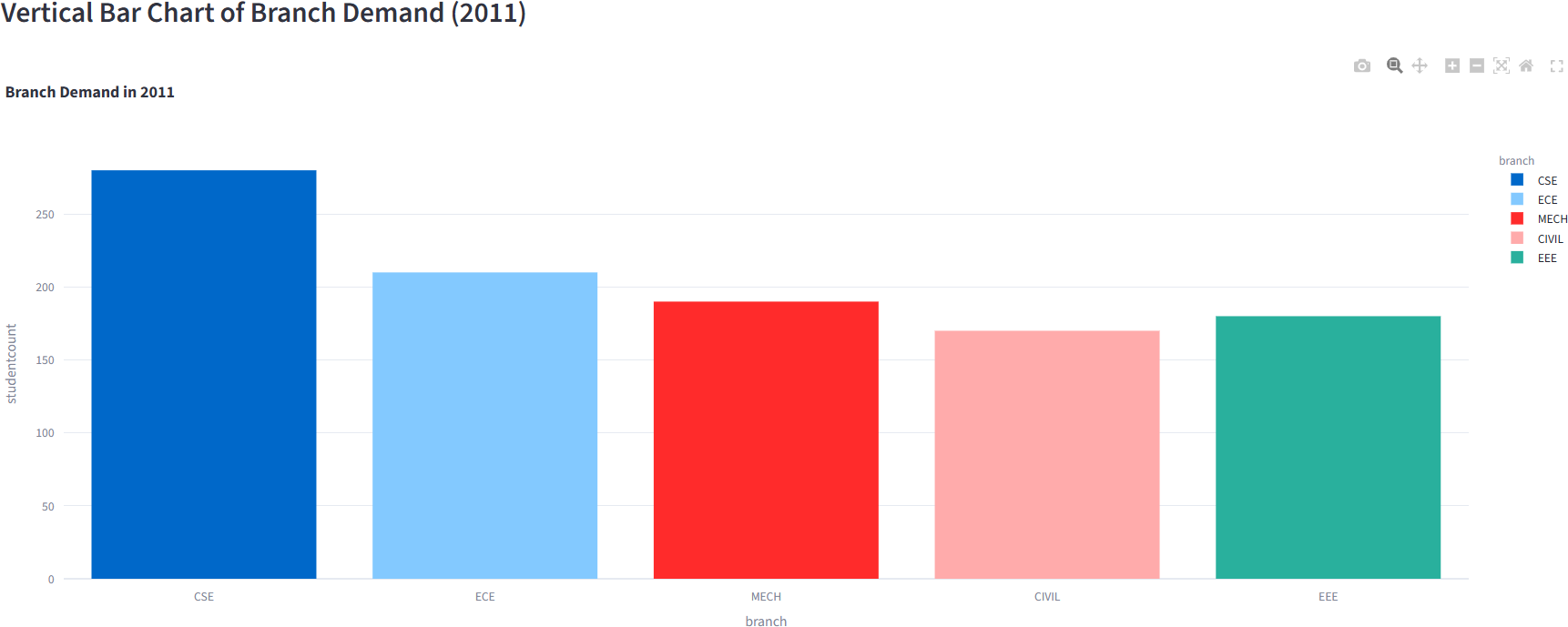
Branch demand by year



piechart of the selected year

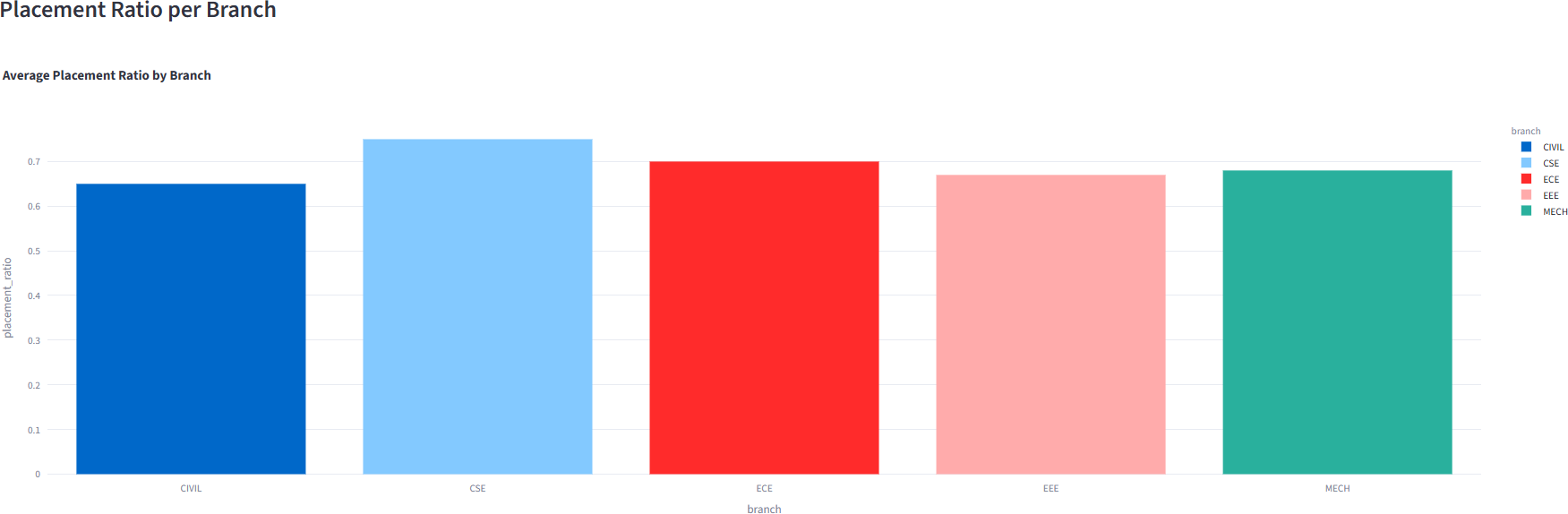


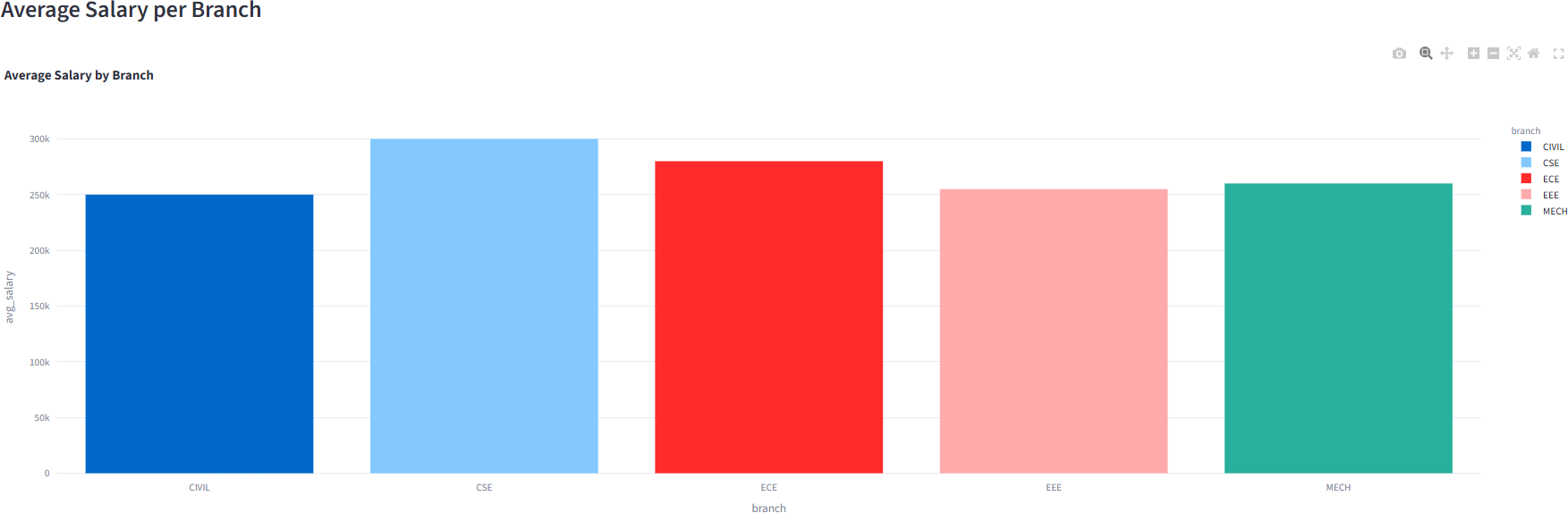
Vertical bar graph of the selected year



# 

Placement & Salary charts





Summary for latest year



# 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.