STUDENT PLACEMENT ANALYSIS DASHBOARD REPORT

# Name : Srinidhi R

# Usn : 4GW23CI053

# Email : ramdasnidhi@gmail.com

# INDEX :

1. **Project Tittle** ................................................. 3
2. **Problem Statement** ................................... 3
3. **Project Description** .......................... 3
4. **Data and Output Purpose** ...................... 4

## Input Data (MySQL tables)

## Output & Benefits

1. **Solution Plan** .............................................. 4
2. **Diagram Design**

## Activity Diagram

## Data Flow Diagram

1. **Code and Implementation**
2. **Output Screenshots & Explanation** .... 18
3. **Conclusion** ................................................

10. **Closure**

11. **Bibliography**

# 1.Project Tittle :

Placement Analysis Dashboard

# 2.Problem Statement :

The Placement Cell maintains large volumes of data about students, companies, interviews, offers, and recruitment outcomes. However, this data is often stored in raw form and lacks advanced insights. As a result, it becomes difficult to:

* Identify placement trends across batches, branches, and recruiters.
* Measure recruiter performance and student preparedness.
* Track salary variations and conversion rates.

Without a proper analytical system, placement officers rely on manual reports, which are time-consuming and less effective for decision-making.

There is a strong need for a **data-driven system** that performs efficient preprocessing and analysis, and uses visualization techniques to provide actionable insights. Such a system can improve placement strategies, optimize training programs, and enhance student preparedness.

# 3.Project Description :

* This project develops a **Python-based Placement Analytics Dashboard** that transforms raw placement data into meaningful insights. The system leverages **NumPy and Pandas** for data preprocessing and analysis, while **Matplotlib, Plotly, and Streamlit** are used for interactive dashboards and visualization.
* The system provides:
* **Placement Trends Analysis**: Year-wise, branch-wise, and company-wise statistics, along with salary trends (highest, lowest, average).
* **Performance Indicators**: Interview-to-offer conversion rates, top recruiting companies, high-performing students, and placement outcomes correlated with student academic performance (CGPA).
* **Recruiter Insights**: Identifying top recruiters, conversion efficiency, and salary distribution.
* **Student-Centric Insights**: Highlighting departments and CGPA ranges with strong or weak placement performance.
* By combining data analysis with rich visualization, this system acts as a **decision-support tool** for the Placement Cell, enabling administrators, faculty, and students to make informed, data-driven decisions that improve the overall placement process.

# 4.Data and Output Purpose :

## Input Data (MySQL tables) :

The system uses a **relational database (placement\_db)** where multiple tables are linked together to provide a complete view of student placement activity:

1. **student**
   * Attributes: *usn, name, dept, batch, cgpa*
   * Purpose: Stores core student information including academic performance (CGPA).
2. **company**
   * Attributes: *cid, company name*
   * Purpose: Maintains details of recruiters participating in placement drives.
3. **performance**
   * Attributes: *usn, cid, status*
   * Purpose: Tracks recruitment outcomes for each student against each company.
   * Status is stored as numeric codes (10 = Placed, 9 = Shortlisted , 0 = Not Eligible, 1 = Unable to Clear 1st Round, 2 = Unable to Clear GD, 3 = Unable to Clear Technical Round , 4 = Unable to Clear HR ).
4. **hiring**
   * Attributes: *hid, cid, ctc, dept, batch , date.*
   * Purpose: Stores salary package (Cost-to-Company) information offered by companies.

## Output & Benefits :

The system transforms raw data from the above tables into meaningful results:

1. **Placement KPIs (Key Performance Indicators)**
   * Total students appearing for placements.
   * Number of students placed.
   * Number of students shortlisted.
   * Overall placement rate (Placed + Shortlisted ÷ Total Students).
2. **Recruiter Insights**
   * Top recruiting companies by hires.
   * Salary trends: highest, lowest, and average package offered.
   * Interview-to-offer conversion rates for each company.
3. **Comparative Analysis**
   * Batch-wise placement outcomes to identify strong and weak years.
   * Department-wise success rates to compare academic streams.
4. **Decision-Support for Training Programs**
   * Identifies gaps in student performance.
   * Helps placement officers design better pre-placement training.
5. **Downloadable Hiring Records**
   * Hiring reports available in CSV format.
   * Facilitates documentation and sharing across departments.

# 5.Solution Plan :

**1. Data Ingestion**

* Data is fetched directly from the MySQL database using the **dbconfig.py** connector.
* Queries join multiple tables (student, performance, hiring, company) to create a consolidated dataset.
* Pandas DataFrames are used for efficient data handling.

**2. Analysis & Processing**

* Placement status codes are mapped into **human-readable categories** (e.g., 10 → Placed, 9 → Shortlisted).
* Filters are applied based on user input: batch, department, or company.
* KPIs such as placement rate, total hires, and salary averages are computed.

**3. Dashboard Interface (Streamlit)**

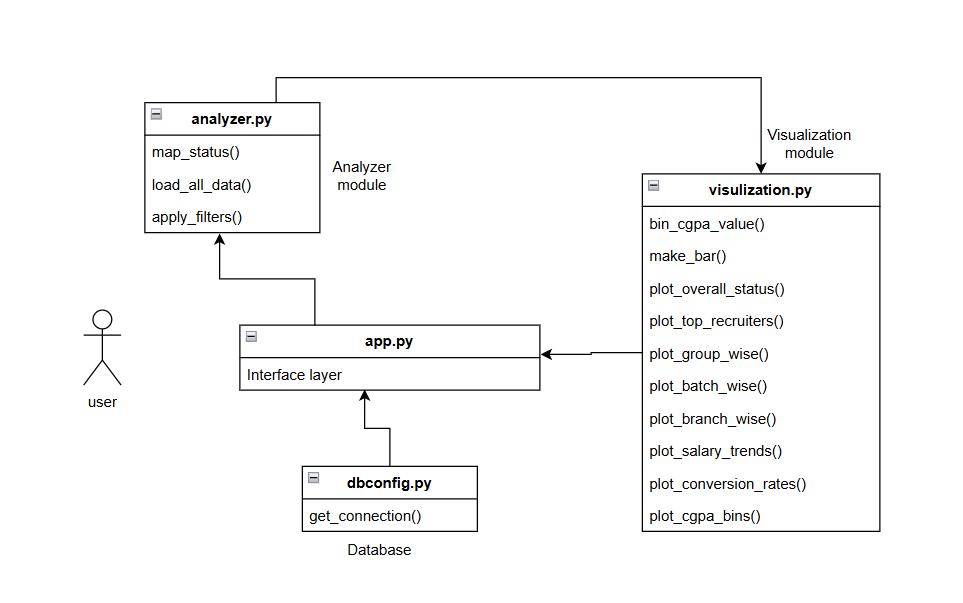
* Interactive sidebar allows users to set filters.
* KPIs are displayed in **card format** for clarity.
* Data tables are shown with sorting and searching options.
* A download button generates CSV reports for offline analysis.

**4. Visualization (Plotly)**

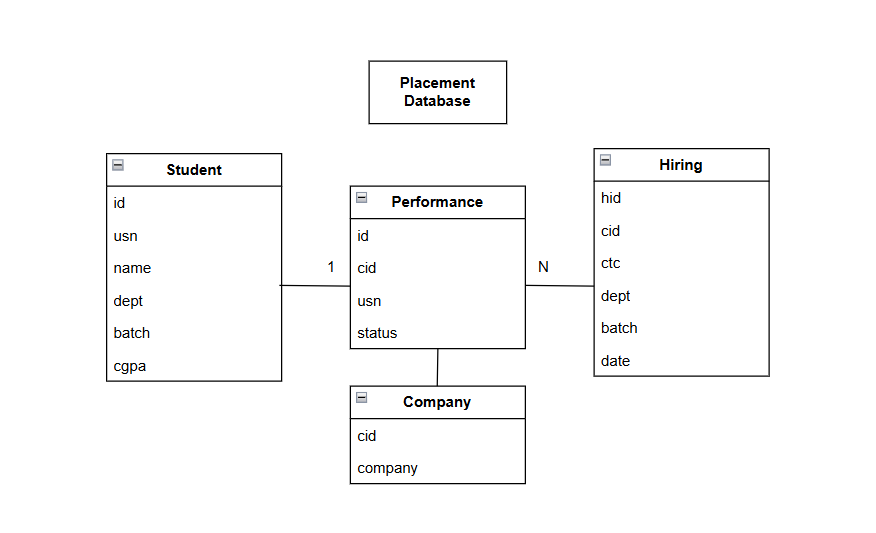
* **Overall Placement Distribution**: Pie/Bar charts of placement status.
* **Batch-wise Trends**: Placement statistics across years.
* **Branch-wise Trends**: Department-level comparisons.
* **Recruiter Analysis**: Top companies, salary trends, conversion rates.
* **CGPA Impact**: Placement outcomes grouped by CGPA bins.

# 6.Diagram Design:

## Activity Diagram :



## Data Flow Diagram :

* 

**Actor: User**

* Could be an Admin, Placement Officer, or Student.
* Interacts with dashboard via browser.

**Use Cases:**

* Apply filters (batch, dept, company).
* Analyze placement status.
* Compare salary trends.
* Export hiring records.

**System Components:**

* **Analyzer Module (analyzer.py)**
  + Loads & preprocesses data.
  + Maps status codes.
  + Applies filters.
* **Visualization Module (visualization.py)**
  + Generates charts (bar, pie, stacked graphs).
  + Provides text-based insights.
* **Interface Layer (app.py)**
  + Streamlit front-end.
  + Sidebar filters, KPI cards, graphs, data tables.
* **Database Connector (dbconfig.py)**
  + Connects to MySQL placement\_db.

# 7.Code and Implementation :

**1. dbconfig.py**

import mysql.connector

def get\_connection():

    conn = mysql.connector.connect(

        host="localhost",

        user="root",

        password="",

        port=3307,

        database="placement\_db"

    )

    return conn

**Explanation:**

* **Purpose:** Establishes a connection between the Python application and the MySQL database.
* **Working:**
  + Uses the mysql.connector library to open a connection.
  + Connects to the database placement\_db running on localhost:3307 with the username root.
  + Returns a live connection object which can be used in queries.
* **Role in System:** Acts as the *gateway* between the placement data and the analytics system. Without this, the app cannot fetch or update data.

**2. analyzer.py**

This is the **backend processing module** that loads, merges, and filters placement data.

**Placement Status Mapping**

import pandas as pd

from dbconfig import get\_connection

PLACEMENT\_ORDER = [

    "Not Eligible",

    "Unable to Clear 1st Round",

    "Unable to Clear Technicals",

    "Unable to Clear HR",

    "Shortlisted",

    "Placed",

    "Unable to Clear GD"

]

def map\_status(code: int) -> str:

    mapping = {

        0: "Not Eligible",

        1: "Unable to Clear 1st Round",

        3: "Unable to Clear Technicals",

        4: "Unable to Clear HR",

        9: "Shortlisted",

        10: "Placed",

        2: "Unable to Clear GD"

    }

    return mapping.get(code, "Unknown")

* Converts numeric codes from the **performance** table into **readable labels**.
* Ensures all charts and reports use consistent categories (PLACEMENT\_ORDER).

**Load Data and Join Tables**

def load\_all\_data():

    conn = get\_connection()

    student\_df = pd.read\_sql("SELECT \* FROM student", conn)

    company\_df = pd.read\_sql("SELECT \* FROM company", conn)

    performance\_df = pd.read\_sql("SELECT \* FROM performance", conn)

    hiring\_df = pd.read\_sql("SELECT \* FROM hiring", conn)

    # Build combined\_df with joins

    query = """

        SELECT s.usn, s.name, s.dept, s.batch, s.cgpa,

               p.status, c.company, h.ctc

        FROM student s

        LEFT JOIN performance p ON s.usn = p.usn

        LEFT JOIN hiring h ON p.cid = h.cid

        LEFT JOIN company c ON h.cid = c.cid

    """

    combined\_df = pd.read\_sql(query, conn)

    conn.close()

    combined\_df["Placement\_status"] = combined\_df["status"].apply(map\_status)

    combined\_df["Placement\_status"] = pd.Categorical(

        combined\_df["Placement\_status"],

        categories=PLACEMENT\_ORDER,

        ordered=True

    )

    return student\_df, company\_df, performance\_df, hiring\_df, combined\_df

* Connects to MySQL and loads four tables into **Pandas DataFrames**.
* Runs a SQL join to merge student, performance, hiring, and company info.
* Creates a **combined DataFrame** with columns: usn, name, dept, batch, cgpa, status, company, ctc.
* Adds a categorical column Placement\_status for ordering results.

**Apply Filters**

def apply\_filters(df, batch\_filter="All", dept\_filter="All", company\_filter="All"):

    filtered = df.copy()

    # Batch filter

    if batch\_filter == "Last 3 Years":

        all\_batches = sorted(filtered["batch"].dropna().unique())

        if len(all\_batches) >= 3:

            filtered = filtered[filtered["batch"].isin(all\_batches[-3:])]

    elif batch\_filter != "All":

        filtered = filtered[filtered["batch"] == batch\_filter]

    # Department filter

    if dept\_filter != "All":

        filtered = filtered[filtered["dept"] == dept\_filter]

    # Company filter

    if company\_filter != "All":

        filtered = filtered[filtered["company"] == company\_filter]

    return filtered

* Supports filtering by:
  + **Batch** (specific year or last 3 years).
  + **Department** (CSE, ECE, etc.).
  + **Company** (specific recruiter).

Output is a **filtered DataFrame** ready for visualization.

**3. app.py**

This is the **main Streamlit front-end application**.

**Page Configuration & Data Loading**

import streamlit as st

import pandas as pd

from analyzer import load\_all\_data, apply\_filters

import visualization as vz

# --- Page Config ---

st.set\_page\_config(page\_title="Placement Analysis Dashboard", layout="wide")

st.title("📊 Placement Analysis Dashboard")

# --- Load all data once ---

student\_df, company\_df, performance\_df, hiring\_df, combined\_df = load\_all\_data()

* Sets dashboard title and layout.
* Loads all required data once at startup.

**Sidebar Filters**

st.sidebar.header("Criteria ")

dept\_options = ["All"] + sorted(student\_df["dept"].dropna().unique().tolist())

dept\_filter = st.sidebar.selectbox("Select Department", dept\_options)

batch\_list = sorted(student\_df["batch"].dropna().unique().tolist())

batch\_options = ["All", "Last 3 Years"] + batch\_list

batch\_filter = st.sidebar.selectbox("Select Batch", batch\_options)

company\_options = ["All"] + sorted(company\_df["company"].dropna().unique().tolist())

company\_filter = st.sidebar.selectbox("Select Company", company\_options)

# --- Apply filters ---

df = apply\_filters(combined\_df, batch\_filter, dept\_filter, company\_filter)

Lets users filter placement records dynamically.

**KPI Summary Cards**

st.subheader("📌 Key Placement Metrics")

total\_students = df["usn"].nunique()

placed\_students = df[df["Placement\_status"] == "Placed"]["usn"].nunique()

shortlisted\_students = df[df["Placement\_status"] == "Shortlisted"]["usn"].nunique()

placement\_rate = ((placed\_students + shortlisted\_students) / total\_students \* 100) if total\_students > 0 else 0

card\_style = """

    <div style="background-color:{bg}; padding:20px; border-radius:15px;

                box-shadow:0px 4px 10px rgba(0,0,0,0.25); text-align:center;">

        <h2 style="margin:0; font-size:28px; color:#212121;">{value}</h2>

        <p style="margin:0; font-size:16px; font-weight:bold; color:#424242;">{label}</p>

    </div>

"""

col1, col2, col3, col4 = st.columns(4)

with col1:

    st.markdown(card\_style.format(bg="#bbdefb", value=total\_students, label="🎓 Total Students"), unsafe\_allow\_html=True)

with col2:

    st.markdown(card\_style.format(bg="#c8e6c9", value=placed\_students, label="✅ Placed Students"), unsafe\_allow\_html=True)

with col3:

    st.markdown(card\_style.format(bg="#ffe0b2", value=shortlisted\_students, label="📝 Shortlisted"), unsafe\_allow\_html=True)

with col4:

    st.markdown(card\_style.format(bg="#ffcdd2", value=f"{placement\_rate:.2f}%", label="📉 Placement Rate"), unsafe\_allow\_html=True)

st.markdown("---")

* Computes key metrics:
  + Total Students
  + Placed Students
  + Shortlisted Students
  + Placement Rate (%)
* Displays them in styled **info cards**.

**Graphs and Insights**

st.subheader("📊 Placement Analysis Graphs")

if not df.empty:

    color\_set = vz.plot\_overall\_status(df)

    vz.plot\_top\_recruiters(df)

    vz.plot\_batch\_wise(df)

    vz.plot\_branch\_wise(df)

    vz.plot\_salary\_trends(df)

    vz.plot\_conversion\_rates(df)

    vz.plot\_cgpa\_bins(df, color\_set)

else:

    st.info("⚠ No data available for selected filters.")

st.markdown("---")

* Calls visualization functions to show insights.
* Provides real-time text interpretations for better understanding.

**Hiring Records Table**

* st.subheader("📑 Hiring Records Table")

if not df.empty:

    pivot\_df = df.pivot\_table(

        index=["usn", "name", "dept", "batch", "cgpa"],

        columns="company",

        values="Placement\_status",

        aggfunc="first"

    ).reset\_index()

    st.dataframe(pivot\_df, use\_container\_width=True, height=500)

    st.download\_button(

        label="📥 Download CSV",

        data=pivot\_df.to\_csv(index=False).encode("utf-8"),

        file\_name="hiring\_records.csv",

        mime="text/csv"

    )

else:

    st.info("⚠ No hiring records available for the selected filters.")

# --- Footer ---

st.markdown("""--------------""")

* Builds a **pivot table**: students vs. companies.
* Provides a **download button** for CSV reports.

1. **visualization.py**

import pandas as pd

import plotly.express as px

import streamlit as st

from analyzer import PLACEMENT\_ORDER

# --- Utils ---

def bin\_cgpa\_value(cgpa):

try:

cgpa = float(cgpa)

except:

return "Unknown"

return "<6" if cgpa < 6 else "6-7" if cgpa < 7 else "7-8" if cgpa < 8 else "8-9" if cgpa < 9 else "9-10"

def make\_bar(df, x, y, color, title, text="count", barmode=None, order=None, height=500, angle=0, hover=None):

fig = px.bar(df, x=x, y=y, color=color, text=text, height=height,

title=title, category\_orders={color: order} if order else None)

fig.update\_traces(texttemplate="%{text}", textposition="inside", hovertemplate=hover)

fig.update\_layout(xaxis\_title=x.capitalize(), yaxis\_title=y.capitalize(),

legend\_title=color.replace("\_"," ").title(), xaxis\_tickangle=angle, barmode=barmode)

st.plotly\_chart(fig, use\_container\_width=True)

# --- Overall Placement ---

def plot\_overall\_status(df):

df = df[df["Placement\_status"] != "Unknown"]

total = df["usn"].nunique()

if total == 0:

return st.info("⚠ No placement data available.")

stats = df.groupby("Placement\_status")["usn"].nunique().reset\_index(name="count")

stats["percent"] = stats["count"]/total\*100

make\_bar(stats, "Placement\_status","percent","Placement\_status",

"Overall Placement Status (%)", text="count", order=PLACEMENT\_ORDER)

# --- Text Analysis ---

placed = stats.loc[stats["Placement\_status"] == "Placed", "percent"].sum()

shortlisted = stats.loc[stats["Placement\_status"] == "Shortlisted", "percent"].sum()

not\_placed = 100 - (placed + shortlisted)

st.markdown(f"""

🔎 \*\*Analysis:\*\*

Out of \*\*{total} students\*\*, \*\*{placed:.1f}% were placed\*\* and \*\*{shortlisted:.1f}% shortlisted\*\*.

Around \*\*{not\_placed:.1f}% of students could not secure offers\*\*.

""")

# --- Branch/Batch wise ---

def plot\_group\_wise(df, group\_col, title):

df = df[df["Placement\_status"] != "Unknown"]

if df.empty:

return st.info(f"⚠ No {group\_col}-wise data available.")

totals = df.groupby(group\_col)["usn"].nunique().reset\_index(name="total")

stats = df.groupby([group\_col,"Placement\_status"])["usn"].nunique().reset\_index(name="count")

stats = stats.merge(totals, on=group\_col)

stats["percent"] = stats["count"]/stats["total"]\*100

if stats.empty:

return st.info(f"⚠ No {group\_col}-wise data available.")

make\_bar(stats, group\_col,"percent","Placement\_status",

f"{title} Placement Status (%)", text="count", order=PLACEMENT\_ORDER, barmode="stack", height=600)

# --- Text Analysis ---

best\_group = stats.groupby(group\_col)["percent"].mean().idxmax()

worst\_group = stats.groupby(group\_col)["percent"].mean().idxmin()

st.markdown(f"""

🔎 \*\*Analysis:\*\*

\*\*{best\_group}\*\* shows the \*\*highest placement performance\*\* among all {group\_col}s.

\*\*{worst\_group}\*\* has the \*\*lowest placement outcomes\*\*.

""")

def plot\_branch\_wise(df):

plot\_group\_wise(df,"dept","Branch-wise")

def plot\_batch\_wise(df):

plot\_group\_wise(df,"batch","Batch-wise")

# --- Top Recruiters ---

def plot\_top\_recruiters(df):

top = df[df["Placement\_status"].isin(["Placed","Shortlisted"])]

top = top.groupby("company")["usn"].nunique().reset\_index(name="hires").sort\_values("hires",ascending=False)

if top.empty:

return st.info("⚠ No recruiter data.")

make\_bar(top,"company","hires","company","Top Recruiters (Placed+Shortlisted)", text="hires", height=600, angle=45)

# --- Text Analysis ---

top\_company = top.iloc[0]["company"]

top\_hires = top.iloc[0]["hires"]

st.markdown(f"""

🔎 \*\*Analysis:\*\*

\*\*{top\_company}\*\* emerged as the leading recruiter with \*\*{top\_hires} hires\*\*.

""")

# --- CGPA bins ---

def plot\_cgpa\_bins(df, colors):

df["cgpa\_bin"] = df["cgpa"].apply(bin\_cgpa\_value)

df = df[df["cgpa\_bin"]!="Unknown"]

order = ["<6","6-7","7-8","8-9","9-10"]

col1,col2 = st.columns(2)

with col1:

stats\_all = df.groupby("cgpa\_bin")["usn"].nunique().reset\_index(name="count")

make\_bar(stats\_all,"cgpa\_bin","count","cgpa\_bin","CGPA Distribution (All)", order=order, height=500)

with col2:

placed = df[df["Placement\_status"].isin(["Placed","Shortlisted"])]

if placed.empty:

st.info("⚠ No placed/shortlisted CGPA data.")

else:

stats\_p = placed.groupby("cgpa\_bin")["usn"].nunique().reset\_index(name="count")

make\_bar(stats\_p,"cgpa\_bin","count","cgpa\_bin","CGPA Distribution (Placed+Shortlisted)", order=order, height=500)

# --- Text Analysis ---

if not df.empty:

dominant\_bin = df["cgpa\_bin"].mode()[0]

st.markdown(f"""

🔎 \*\*Analysis:\*\*

Most students who are not placed fall in the \*\*{dominant\_bin} CGPA range\*\*.

""")

# --- Salary Trends ---

def plot\_salary\_trends(df):

stats = df[df["Placement\_status"]=="Placed"].groupby("company")["ctc"].agg(highest="max",lowest="min",average="mean").reset\_index()

if stats.empty:

return st.info("⚠ No salary data.")

fig = px.pie(stats, names="company", values="average", title="Average Salary Distribution", color\_discrete\_sequence=px.colors.qualitative.Set3)

fig.update\_traces(textinfo="percent+label")

st.plotly\_chart(fig, use\_container\_width=True)

# --- Text Analysis ---

top\_salary = stats["average"].max()

top\_company = stats.loc[stats["average"].idxmax(), "company"]

st.markdown(f"""

🔎 \*\*Analysis:\*\*

\*\*{top\_company}\*\* offers the \*\*highest average salary\*\* of approximately \*\*{top\_salary:.2f} LPA\*\*.

""")

# --- Conversion Rates ---

def plot\_conversion\_rates(df):

if df.empty:

return st.info("⚠ No conversion rate data.")

conv = df.groupby("company").apply(lambda x: x[x["Placement\_status"]=="Placed"]["usn"].nunique()/x["usn"].nunique()\*100 if x["usn"].nunique()>0 else 0).reset\_index().rename(columns={0:"conversion"})

if conv.empty:

return st.info("⚠ No conversion rate data.")

conv["conversion"] = conv["conversion"].round().astype(int)

make\_bar(conv.sort\_values("conversion",ascending=False),"company","conversion","company","Interview-to-Offer Conversion (%)", text="conversion", height=600, angle=45)

# --- Text Analysis ---

best\_company = conv.loc[conv["conversion"].idxmax(), "company"]

best\_rate = conv["conversion"].max()

st.markdown(f"""

🔎 \*\*Analysis:\*\*

\*\*{best\_company}\*\* shows the \*\*highest interview-to-offer conversion rate\*\* at \*\*{best\_rate}%\*\*.

""")

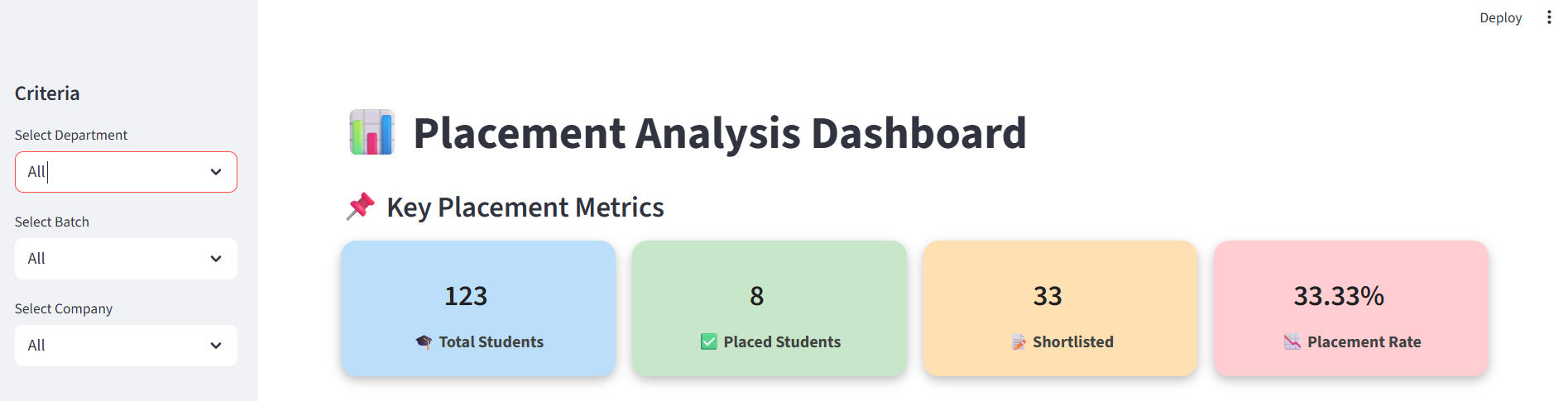
This module handles **all data visualizations**.

**Key Functions**

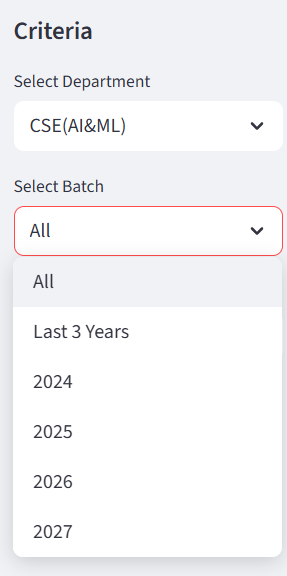
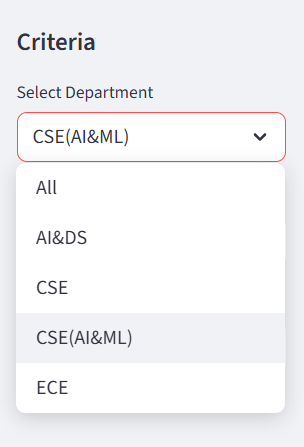
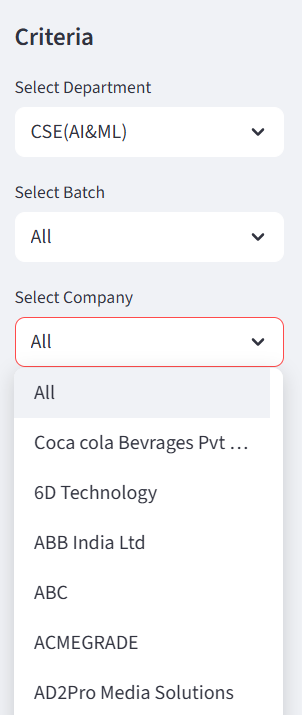
1. **plot\_overall\_status(df)**
   * Shows placement status distribution (Placed, Shortlisted, Not Eligible, etc.).
   * Provides percentage + count.
   * Adds text-based analysis.
2. **plot\_group\_wise(df, group\_col, title)**
   * Used by plot\_batch\_wise() and plot\_branch\_wise().
   * Compares placement results across years or departments.
   * Highlights best and worst performing groups.
3. **plot\_top\_recruiters(df)**
   * Lists top companies by number of hires.
   * Identifies the leading recruiter.
4. **plot\_salary\_trends(df)**
   * Creates pie chart of average salaries offered by companies.
   * Highlights the company with the best package.
5. **plot\_conversion\_rates(df)**
   * Calculates interview-to-offer conversion percentage for each recruiter.
   * Shows which company is most efficient.
6. **plot\_cgpa\_bins(df)**
   * Groups students into bins (<6, 6–7, 7–8, 8–9, 9–10).
   * Compares placement success across CGPA ranges.
   * Gives insights like “most placed students have CGPA between 8–9”.

# 8. Output Screenshots & Explanation:

**Dashboard KPIs** – Displays key placement metrics like total students, placed students, shortlisted students, and placement rate.



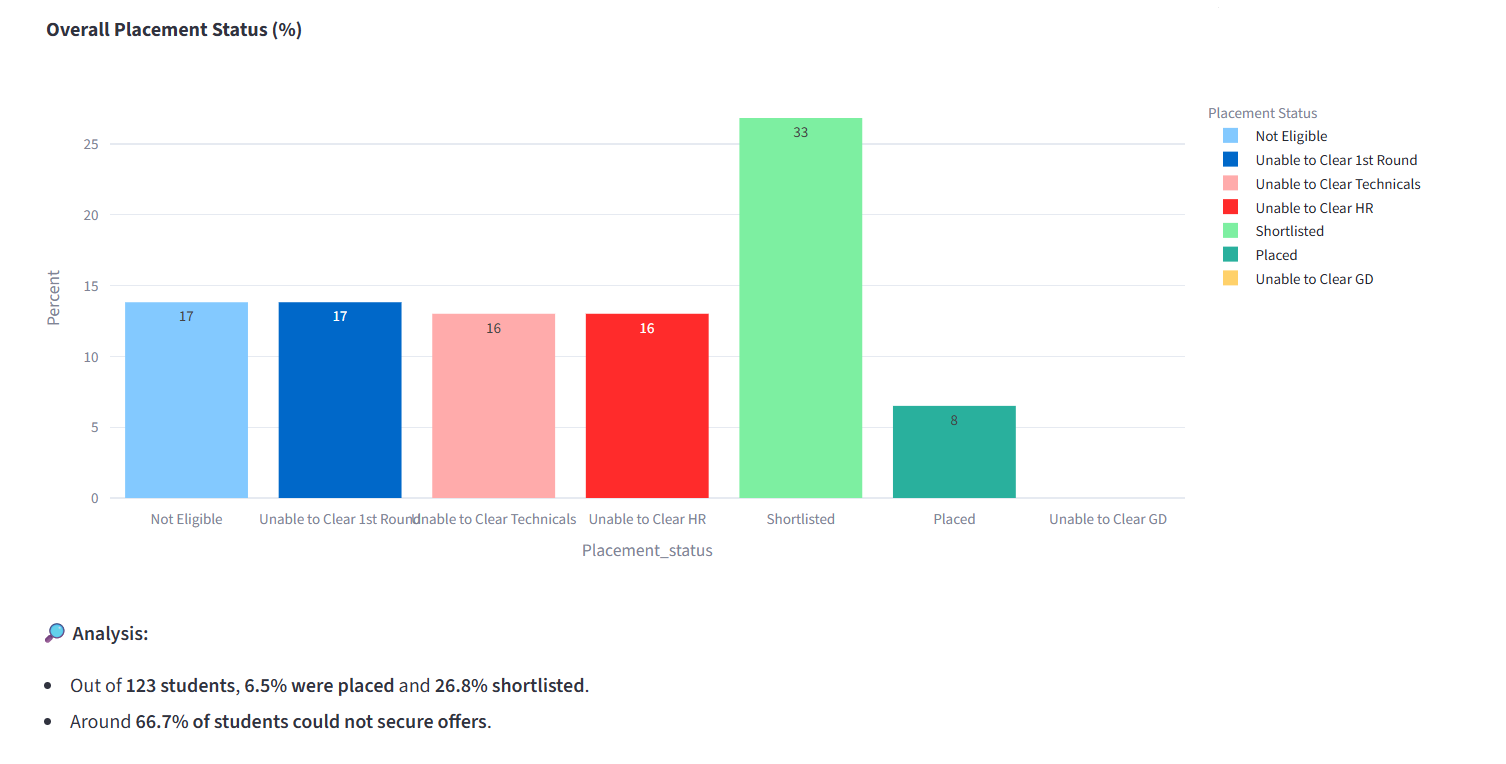
FILTERS :

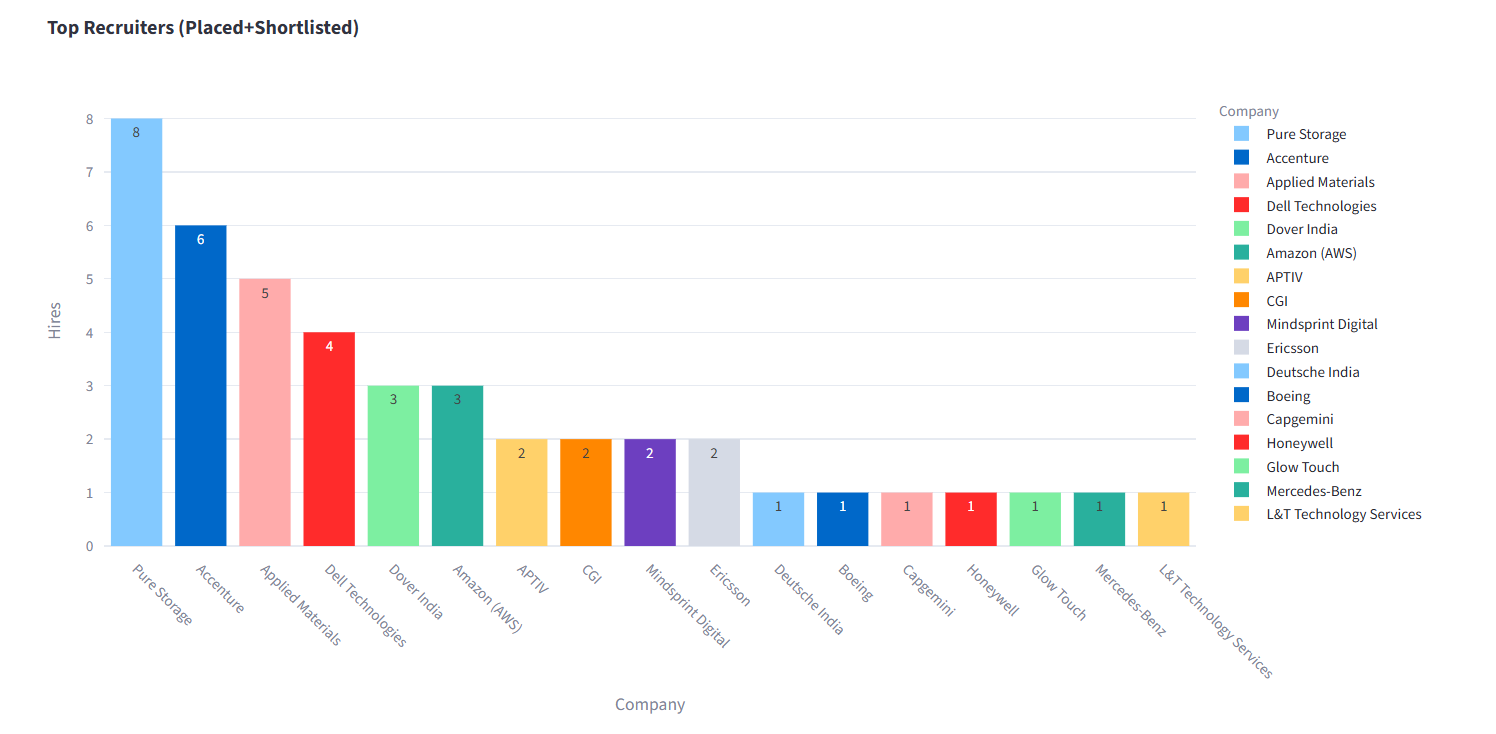
**Filter by Department** – Allows selection of a specific department (CSE, AI&DS, AI&ML, ECE, etc.) for analysis.

**Filter by Batch** – Enables analysis for specific batches (e.g., 2024, 2025, 2026) or across the last three years.

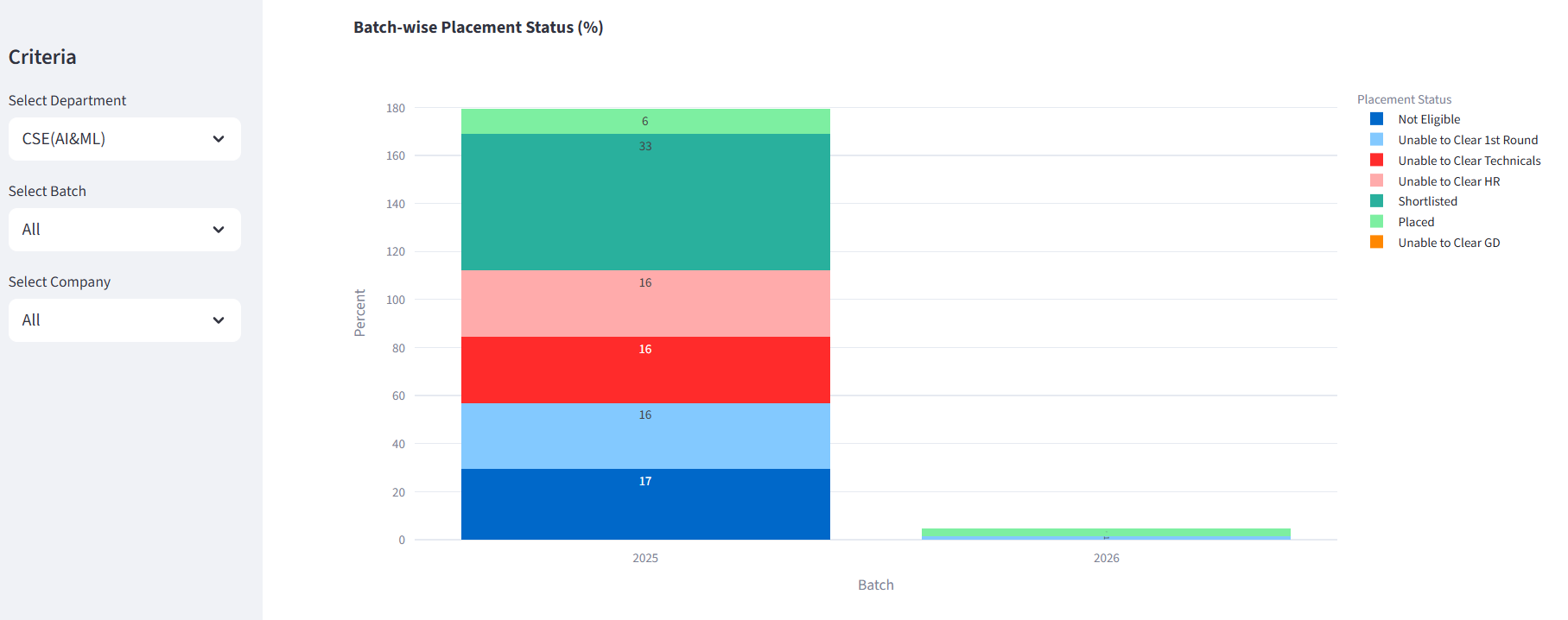
**Filter by Company** - Enables analysis for specific companies.



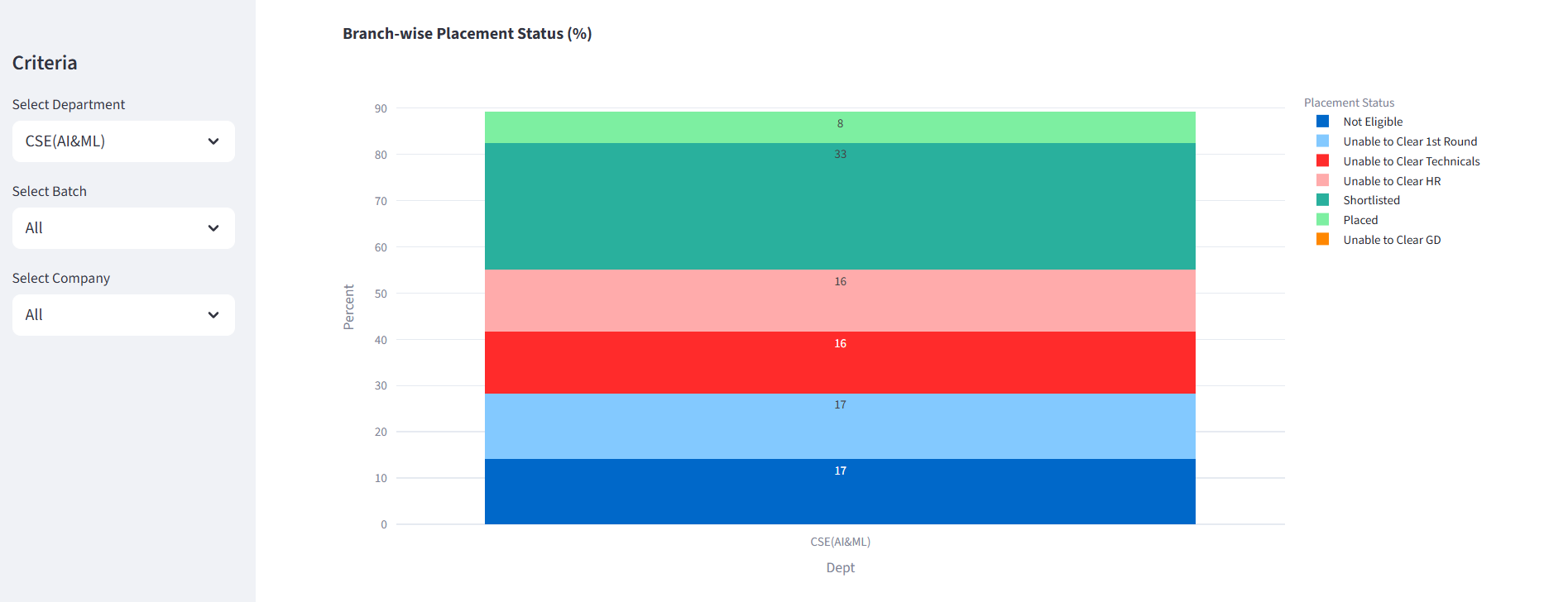
**Overall Placement Status (Bar Graph)** – Shows distribution of students across different placement statuses (e.g., Placed, Shortlisted, Not Eligible).



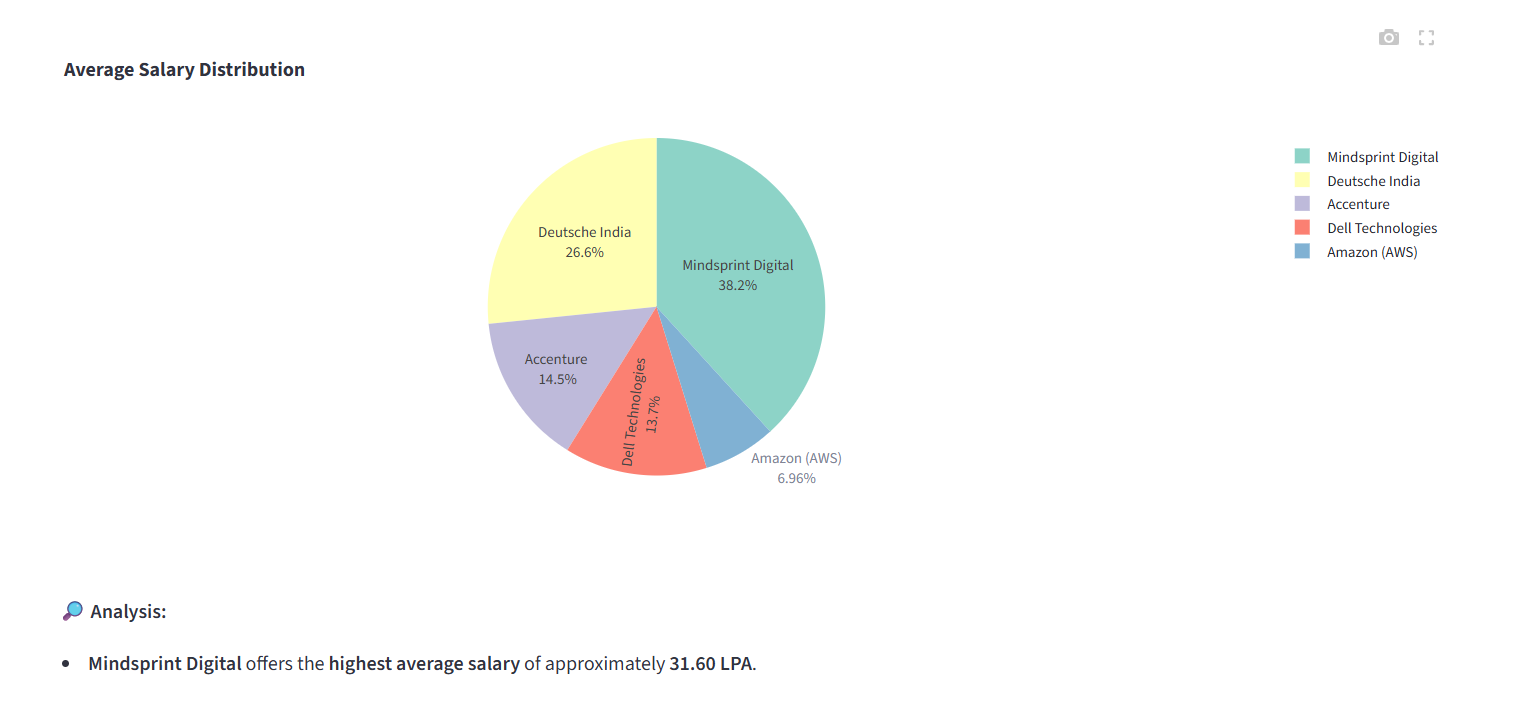
**Top Recruiters Chart** – Ranks companies based on the total number of hires (Placed + Shortlisted).



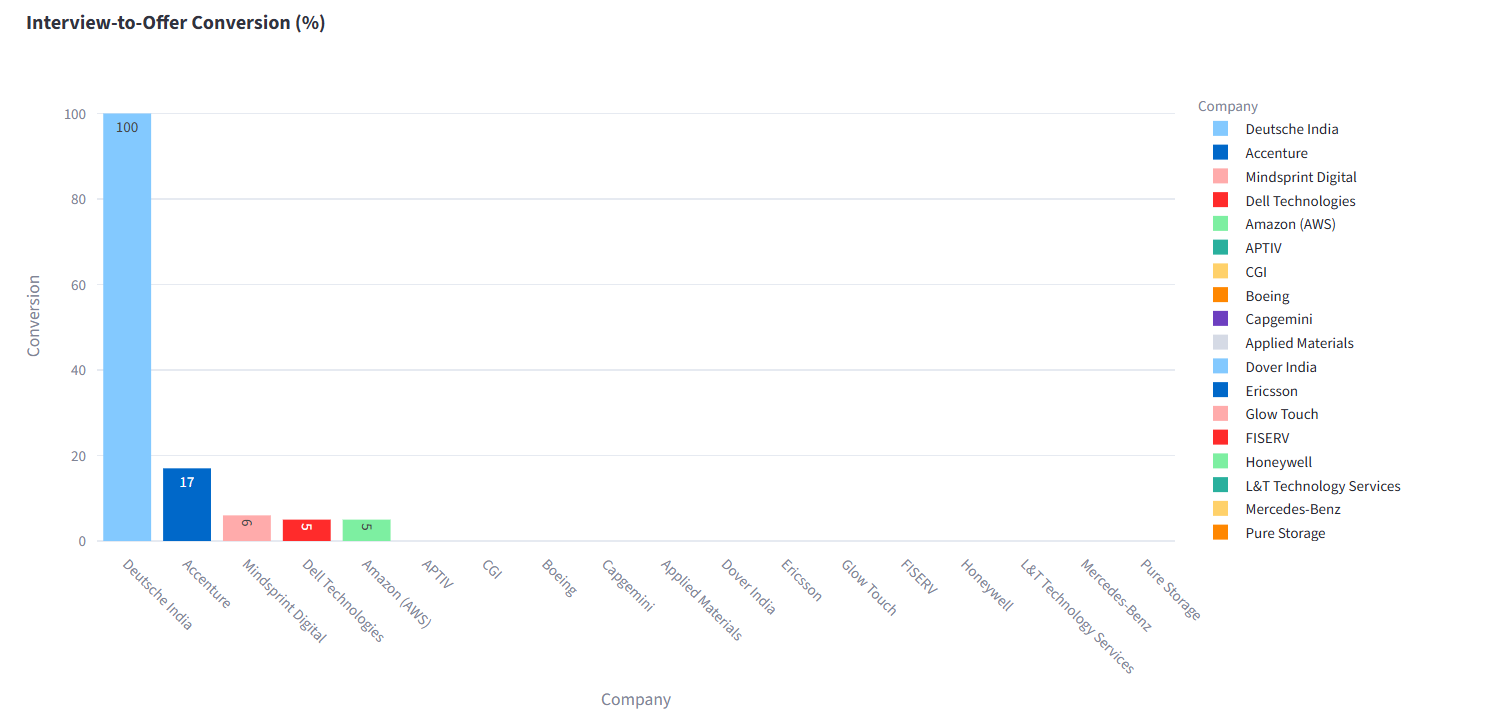
**Batch wise Analysis**



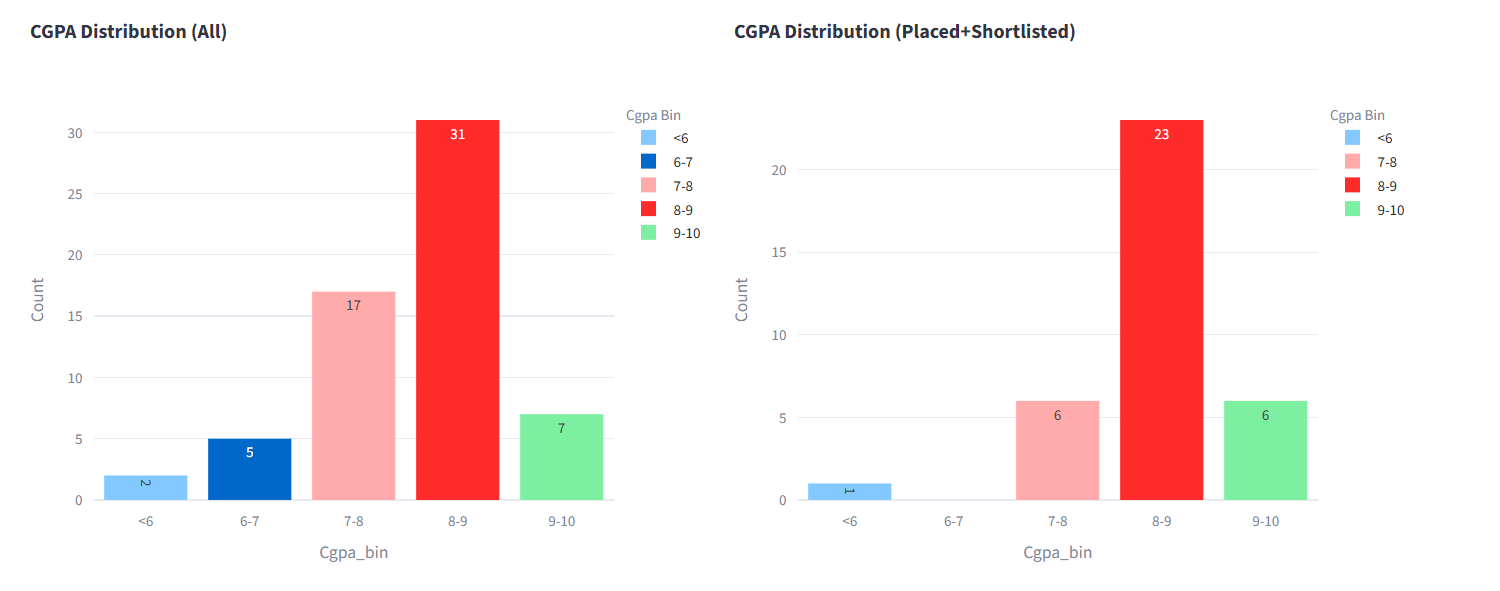
**Branch wise Analysis**



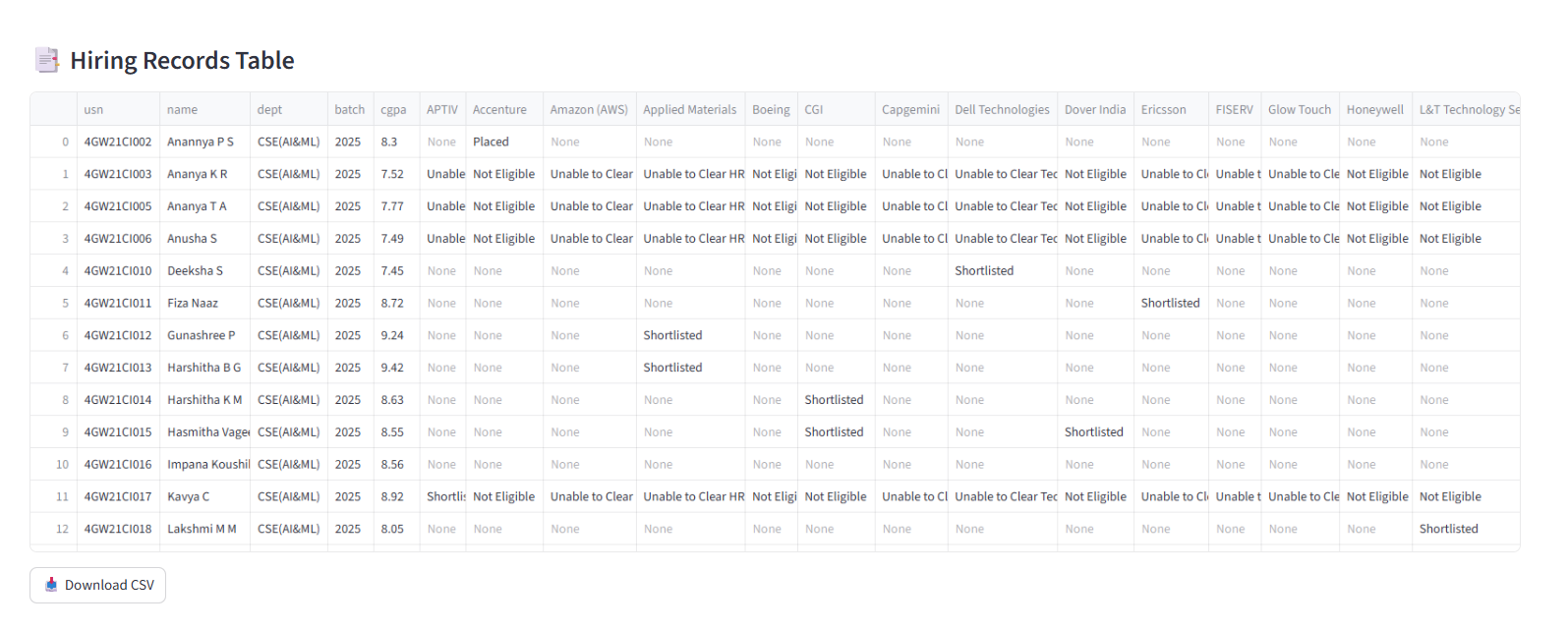
**Salary Distribution (Pie Chart)** – Visualizes average salary packages offered by different companies.



**Interview-to-Offer Conversion** – Displays recruiter efficiency in converting interviews into final offers.



**CGPA Distribution (All vs Placed/Shortlisted)** – Compares overall CGPA ranges with those of successfully placed/shortlisted students.



**Hiring Records Table** – Provides detailed student-wise hiring records with placement outcomes and downloadable CSV option.

# 9. Conclusion :

The **Placement Analysis Dashboard** provides a centralized, interactive, and visual tool for analyzing placement outcomes.

**Key outcomes include:**

* Identified recruiters with the **highest hiring counts and best conversion rates**.
* Highlighted **departments and batches** with the strongest placement performance.
* Demonstrated the correlation between **CGPA and placement success**, helping students understand the impact of academics on recruitment outcomes.
* Enabled administrators to **download hiring records** for official documentation and reporting purposes.

This solution effectively bridges the gap between **raw placement data** and **actionable insights**, making the placement process more **transparent, efficient, and data-driven**.

# 10. Closure :

The Placement Analysis Dashboard is a **scalable and extendable system** that empowers placement officers, faculty, and students with meaningful insights.

**Current Achievements:**

* Real-time **filtering** by batch, department, and company.
* Interactive **graphs and dashboards** for easy interpretation.
* Exportable hiring records for institutional use.

**Future Enhancements may include:**

* **Predictive modeling** to estimate student placement probability.
* Automated **PDF report generation** for each department or batch.
* Integration with **resume parsing** tools and **Applicant Tracking Systems (ATS)** to further streamline placement management.

This ensures the system will continue to grow into a comprehensive **decision-support platform** for placement analytics.

# 11. Bibliography :

* **Streamlit Documentation**: [https://docs.streamlit.io/](https://docs.streamlit.io/?utm_source=chatgpt.com)
* **Pandas Documentation**: [https://pandas.pydata.org/](https://pandas.pydata.org/?utm_source=chatgpt.com)
* **Plotly Express**: [https://plotly.com/python/plotly-express/](https://plotly.com/python/plotly-express/?utm_source=chatgpt.com)
* **MySQL Documentation**: [https://dev.mysql.com/doc/](https://dev.mysql.com/doc/?utm_source=chatgpt.com)
* **Python Documentation**: [https://docs.python.org/](https://docs.python.org/?utm_source=chatgpt.com)