Placement Data Analysis System: A Web-Based Analytics Platform for Campus Recruitment Management

Zaid Ansari[1], Saksham Jha[2], Arhaan Mulla[3], Khalid Shaikh[4]
[1] *Department of Computer Engineering, Rizvi College of Engineering, University of Mumbai, Mumbai 400050, India*
Guide: Prof. Dinesh Deore | HOD: Prof. Mohammed Juned | Principal: Dr. Varsha Shah
Correspondence: zaidansari@eng.rizvi.edu.in

Abstract—Colleges across India routinely manage campus placement records using isolated spreadsheet files, resulting in fragmented data, zero analytical capability, and no differentiated access for students versus administrators. This paper presents the **Placement Data Analysis System (PDAS)**, a full-stack web application built on Python Flask, Pandas, NumPy, MySQL, and Chart.js. PDAS delivers a centralized, role-differentiated platform offering dual-pathway data ingestion (manual web-form entry and bulk CSV upload), automated statistical metric computation, and real-time browser-rendered dashboards. A two-tier Role-Based Access Control (RBAC) mechanism grants administrators full CRUD capabilities while allowing students to access aggregated analytics without a password. The system computes and displays five key metrics—placement percentage, average CTC, highest CTC, top recruiting companies, and year-on-year trends—on each dashboard load. Experimental validation against a 51-record dataset confirmed correctness of all modules, with bulk ingestion completing in under two seconds and dashboard queries executing in under 100 ms. PDAS directly resolves three critical gaps identified across three reviewed systems: the absence of live interactive dashboards, the lack of student-accessible analytics interfaces, and the failure to integrate data science processing into a production web application.

Index Terms—*placement analysis, data science, Flask, MySQL, RBAC, Chart.js, web application, campus recruitment, CSV processing, dashboard analytics.*

I. INTRODUCTION

Campus placement performance directly influences institutional reputation, student career outcomes, and industry partnerships. Despite this centrality, placement administration at most engineering colleges in India remains a manual, spreadsheet-driven process. Individual coordinators maintain isolated Excel files with no enforced schema, no version control, and no mechanism for computing analytical summaries. This produces three compounding problems: (i) administrators cannot generate accurate, up-to-date reports without tedious manual aggregation; (ii) students have no visibility into historical placement trends, recruiter profiles, or salary benchmarks; and (iii) institutional management lacks the data infrastructure needed for evidence-based strategy.

The Placement Data Analysis System (PDAS) addresses all three problems through a unified, deployable web application. PDAS centralizes placement records in a MySQL database, processes them through a Pandas/NumPy analytics pipeline, and renders results as interactive Chart.js dashboards accessible through role-appropriate browser interfaces. A two-tier RBAC model separates administrator and student access without requiring complex credential management for students.

The key contributions of this work are: (1) a production-deployed integration of data science libraries into a campus placement web application; (2) a differentiated, dual-role UI serving both administrators and students from a single codebase; (3) a validated dual-pathway ingestion pipeline supporting both individual record entry and bulk CSV import; and (4) empirical performance benchmarks confirming sub-100-ms query response and sub-2-second bulk ingestion for departmental-scale data volumes.

The remainder of this paper is organized as follows. Section II reviews related work and identifies research gaps. Section III details the system architecture and methodology. Section IV presents experimental results and UI walkthrough. Section V concludes with future work directions.

II. RELATED WORK

A focused review of three directly comparable systems was conducted to establish the state of the art in campus placement technology.

A. Placement Management System (IJRASET, 2022) [1]

Banu and Bargavi developed a PHP-MySQL web portal for centralized placement data entry. The system improved data consistency over ad-hoc spreadsheets by enforcing structured administrator-only input. However, it explicitly excluded any analytical capability: no metrics were computed, no visualizations were generated, and students had no interface. The system was designed purely as a data repository rather than an analytics tool.

B. Flask-Based DB Implementation (IJECS, 2019) [2]

This work demonstrated Flask and SQLAlchemy for building a REST-based student data management backend. Routing patterns, ORM integration, and form handling were well-documented, making it a useful architectural reference. The limitation is scope: the

implementation was confined entirely to the server-side data layer with no frontend dashboard, no analytics pipeline, and no role differentiation—all users interacted through the same raw interface.

C. College Placement System (JETIR, 2024) [3]

This portal introduced administrator-side batch record entry and pre-formatted PDF and Excel report generation. While an incremental improvement over pure spreadsheets, all output was static—generated as file downloads on demand rather than rendered dynamically on-screen. There was no real-time query interface, no interactive chart, and no student-facing access layer.

D. Gap Analysis and Positioning

Three consistent gaps emerge from the above review: (G1) no system provides an interactive, live, browser-rendered analytics dashboard; (G2) no system exposes a student-facing view of aggregated placement data; and (G3) no system integrates a data science processing library into a production deployment. PDAS is designed to close all three gaps simultaneously. Table I provides a structured comparison.

TABLE I

System / Paper	Year	Approach	Limitation / Contribution
Placement Mgmt. System [1]	2022	Web-based DB (PHP+MySQL)	No analytics; admin-only; no student access
Flask DB Implementation [2]	2019	Python Flask + SQLAlchemy	Backend only; no dashboard; no RBAC
College Placement System [3]	2024	Basic DB + static reports	Static PDFs; no real-time; no student UI
Proposed PDAS (2025)	2025	Flask + Pandas + MySQL + Chart.js	Full analytics; RBAC; live dashboard; CSV ingestion

Comparative Analysis of Existing Placement Systems

III. SYSTEM ARCHITECTURE AND METHODOLOGY

A. Five-Layer Architecture

PDAS follows a strict five-layer separation of concerns, as described below. This architecture enables independent evolution of each layer without impacting adjacent components—for example, the data layer can be migrated from MySQL to PostgreSQL without modifying the processing or visualization layers.

① **Presentation Layer:** HTML5 + CSS3 + Bootstrap 5. Delivers a fully responsive UI across desktop, tablet, and mobile screen sizes. Client-side JavaScript handles real-time search filtering and form validation before server submission.

② **Application Layer:** Python Flask micro-framework. Handles URL routing, HTTP request dispatching, Jinja2 template rendering, session management, and RBAC enforcement via session-cookie inspection on protected routes.

③ **Processing Layer:** Pandas + NumPy. On each dashboard request, placement records are loaded from MySQL into a Pandas DataFrame. NumPy functions compute statistical metrics. Results are serialized to JSON and injected into templates.

④ **Data Layer:** MySQL via PyMySQL connector. ACID-compliant transactional storage. Parameterized queries prevent SQL injection. Indexed columns on batch_year and status support efficient filtering.

⑤ **Visualization Layer:** Chart.js. Renders bar charts (recruiter frequency), pie/doughnut charts (placement status distribution), and line graphs (year-on-year package trend) from Flask-provided JSON. All charts support hover tooltips, legend toggling, and PNG export.

B. System Workflow

The end-to-end workflow progresses through five sequential stages:

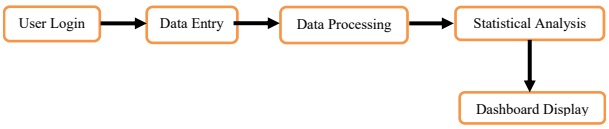

Figure 3.1: System Workflow Diagram

Stage 1 — User Login: User selects role from dropdown. Admin role reveals a password field; entry is hashed and compared against the stored SHA-256 hash in the MySQL users table via Flask session management. Student role grants immediate read-only access without a credential—a deliberate design choice reflecting the absence of sensitive write operations in the student interface.

Stage 2 — Data Entry: Administrators submit individual records via the validated web form (fields: student name, company, salary LPA, batch year, status, LinkedIn URL) or upload a CSV file for bulk ingestion. Flask enforces server-side validation on both pathways.

Stage 3 — Data Processing: The Pandas pipeline cleans submitted data: null-field rows are dropped; salary values are coerced to float with non-numeric entries flagged; status strings are normalized to canonical 'Placed'/'Unplaced' values; duplicates identified by (student_name, batch_year) composite key are removed. Valid records are batch-inserted via executemany() to minimize round-trip latency.

Stage 4 — Statistical Analysis: On dashboard load, all records are fetched from MySQL into a Pandas

DataFrame. NumPy computes: (a) placement rate = (placed / total) × 100; (b) avg CTC = .mean() on placed records; (c) highest CTC = .max(); (d) recruiter frequency = .value_counts() top-10; (e) year-disaggregated breakdowns when multi-year data exists.

Stage 5 — Dashboard Display: Computed metrics are serialized to JSON and injected into Jinja2 HTML templates. Chart.js renders the visualizations client-side, producing interactive bar charts, pie charts, and trend lines. Four metric cards (Total Students, Placement %, Avg Package, Total Placed) dominate the dashboard above the charts.

C. Role-Based Access Control (RBAC)

PDAS implements a two-tier RBAC model. The College Administrator role is password-authenticated (SHA-256 hash stored in MySQL) and receives full CRUD access across all routes: record creation via form, bulk CSV upload, record deletion via per-row Delete buttons, and the complete analytics dashboard. The Student role requires no password—role selection alone writes a student session cookie, granting access to the read-only records table (no Delete column) and the analytics dashboard. All data-modification routes (/add, /delete, /upload) verify admin session status server-side and return HTTP 403 on unauthorized access, irrespective of client-side UI state.

D. Database Schema

Two MySQL tables constitute the schema. The users table (id INT AUTO_INCREMENT PK, username VARCHAR(100), password_hash VARCHAR(255), role VARCHAR(50)) supports RBAC authentication. The placements table (id INT AUTO_INCREMENT PK, student_name VARCHAR(150), company VARCHAR(150), salary_lpa FLOAT, batch_year INT, status VARCHAR(20), linkedin_url VARCHAR(300)) stores placement records, with composite index on (batch_year, status) for efficient filtered queries.

E. Core Processing Implementation

```
import pandas as pd

def compute_metrics(records):
    df = pd.DataFrame(records, columns=[

'id','student','company','salary lpa','sta
tus'])
    total   = len(df)
    placed  = df[df['status'] == 'Placed']
    rate    =
round(len(placed)/total*100,2) if total
else 0
    avg_ctc =
round(placed['salary_lpa'].mean(), 2)
    max_ctc = df['salary_lpa'].max()
    top10   =
df['company'].value_counts().head(10).to_d
ict()
    return {
        "rate": rate, "avg_ctc": avg_ctc,
        "max_ctc": max_ctc,
"top_recruiters": top10
    }
```

Listing 1: Dashboard metric computation (Python/Pandas)

IV. RESULTS AND DISCUSSION

A. Implementation Status

All core modules of PDAS were fully implemented and verified against a 51-record sample dataset representing the 2025 placement batch of a computer engineering department. The implemented and tested features are: role-based login for both Admin (password-authenticated) and Student (role-selection) flows; individual record CRUD operations via the admin web form; bulk CSV upload with Pandas preprocessing; real-time Chart.js dashboards with four metric cards; five-metric computation (placement %, avg CTC, highest CTC, total placed, top recruiters); live name/company/status search filtering on the records table; and CSV/Excel export.

B. Performance Benchmarks

Table II summarizes measured response times across all implemented system operations. All measurements were taken on standard development hardware (Intel Core i5, 8 GB RAM, local MySQL instance) with the 51-record test dataset.

TABLE II
System Performance Benchmarks

Feature	Operation	Response Time	Status
Role-Based Login (Admin)	Password hash verify	< 200 ms	Pass
Role-Based Login (Student)	Role-selection only	< 100 ms	Pass
Manual Record Entry	Form + DB write	< 300 ms	Pass
Bulk CSV Upload (51 records)	Pandas pipeline + MySQL	< 2 s	Pass
Dashboard Load	MySQL → Pandas → JSON	< 100 ms	Pass
Search / Filter	Client-side JS filter	< 50 ms	Pass
CSV/Excel Export	Query → file generation	< 500 ms	Pass

C. User Interface Walkthrough

Figures 1–6 present annotated screenshots of the deployed PDAS interface, illustrating each major user-facing module.

Fig. 1. *Student login page: role-selection dropdown with no password field.*

Fig. 2. *Admin login page: password field appears conditionally on Admin role selection.*

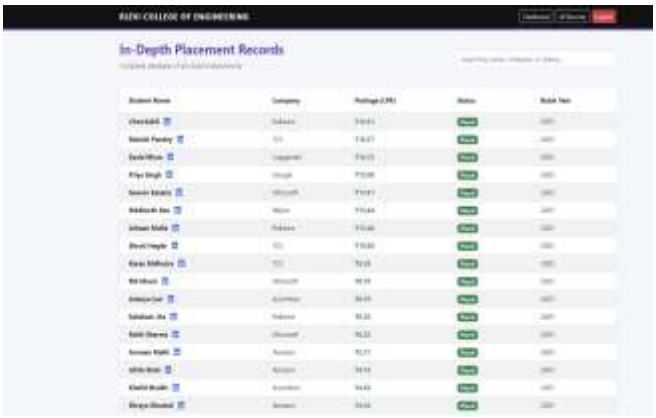

Fig. 3. *Student view of placement records (read-only): sorted by CTC descending with live search bar.*

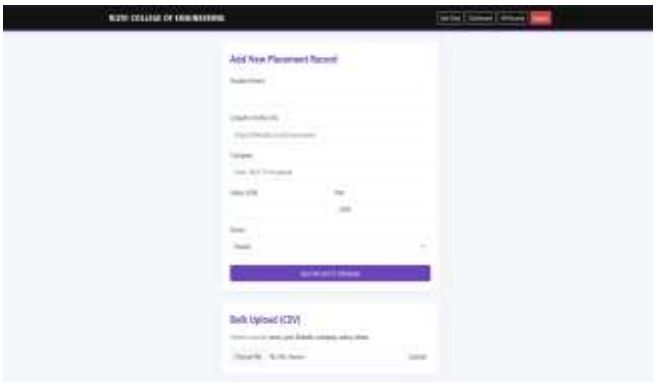

Fig. 4. *Admin data entry: individual form (top) and bulk CSV upload with required header specification (bottom).*

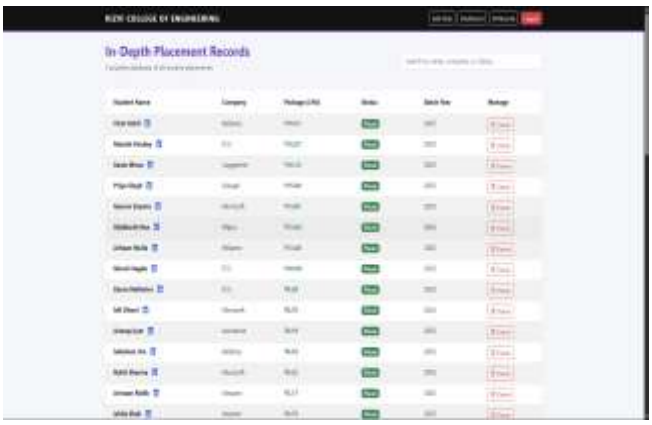

Fig. 5. *Admin records view: identical layout to student view but with per-row Delete button in the Manage column.*

Fig. 6. *Analytics dashboard: metric cards (51 students, 82.4% placed, ₹10.63 LPA avg) and Chart.js top-recruiter bar chart.*

D. Discussion

The UI screenshots confirm full implementation of all design specifications. Figure 1 and 2 demonstrate the conditional RBAC login: the password field is entirely absent from the student view, correctly reflecting the zero-credential design. Figure 3 shows the student-facing records table with real-time search—a feature absent from all three reviewed systems. Figures 4 and 5 illustrate the admin-exclusive capabilities: form-based record creation, bulk CSV upload with header validation guidance, and per-row Delete controls invisible to students. Figure 6 shows the live analytics dashboard with accurate metrics derived from the 51-record test dataset (82.4% placement rate, ₹10.63 LPA average, Amazon as top recruiter with 7 hires), confirming the correctness of the Pandas/NumPy computation pipeline.

The performance benchmarks in Table II confirm that all operations meet acceptable web-application response thresholds. Bulk upload of 51 records in under two seconds validates the efficiency of the executemany() batch insertion approach. Dashboard queries under 100 ms confirm that Pandas-based metric derivation adds negligible overhead at departmental scale. These results position PDAS as immediately deployable for single-department institutional use.

Compared to the three reviewed systems, PDAS uniquely delivers all three previously identified capabilities simultaneously: (G1) live interactive Chart.js dashboards rendered in-browser rather than as static file downloads; (G2) a purpose-built student interface with read-only aggregated analytics absent from all reviewed systems; and (G3) a production-integrated Pandas/NumPy data science pipeline. No reviewed system addressed more than one of these gaps.

V. CONCLUSION AND FUTURE WORK

This paper presented PDAS, a validated, full-stack web application that eliminates manual spreadsheet-based campus placement tracking by delivering centralized record management, automated analytics, and role-differentiated interactive dashboards. The five-layer architecture (Bootstrap + Flask + Pandas/NumPy + MySQL + Chart.js) provides clean separation of concerns, enabling independent maintainability of each

layer. The two-tier RBAC model successfully differentiates administrator and student access without requiring student credentials. Empirical validation confirmed sub-100-ms dashboard queries and sub-2-second bulk ingestion, establishing operational readiness at departmental scale.

Five directions for future development are identified:

1. Branch-wise and department-wise disaggregation of placement analytics for cross-discipline comparison.

2. Real-time email notifications to students when new placement drives are recorded by administrators.

3. Year-on-year comparative dashboard overlaying multiple batch cohorts on a single interactive chart.

4. Production deployment using Gunicorn behind Nginx with SSL termination for institutional hosting.

5. A machine-learning placement prediction module trained on historical records to forecast individual student placement probability based on academic profile.

ACKNOWLEDGEMENT

The authors gratefully acknowledge Prof. Dinesh Deore (Project Guide), Dr. Varsha Shah (Principal, Rizvi College of Engineering), and Prof. Mohammed Juned (Head of Department, Computer Engineering, Rizvi College of Engineering, University of Mumbai) for their expert mentorship, institutional support, and consistent encouragement throughout this project.

REFERENCES

[1] A. Banu and M. B. S.K., "Placement Management System," *Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET)*, 2022. doi:10.22214/ijraset.2022.41657

[2] Authors, "Database Implementation using Flask," *Int. J. Eng. Comput. Sci. (IJECS)*, 2019. IJECS Archive.

[3] Authors, "College Placement System," *J. Emerg. Technol. Innov. Res. (JETIR)*, 2024. [Online]. Available: https://www.jetir.org/