# BACSE101 Problem Solving using Python

PROJECT REPORT

on

**UNIVERSITY PORTAL SYSTEM**

*Prepared by:*

**Ansruta Ghosh - 25BCE2167**
**P Sai Santhosh - 25BCE2148**
**Vardaan Agrawal- 25BCE2146**

*Under the supervision of:*

**Goutam Majumder**

## TABLE OF CONTENTS

## ABSTRACT

The *University Portal System* is a Python-based application designed to simplify and automate the management of academic data such as students, teachers, subjects, and marks. It integrates **MySQL** as the backend database for reliable data storage and retrieval, while modules like **hashlib** ensure secure password hashing, and **getpass** conceals sensitive user input for privacy. Data validation is handled using **re (regular expressions)**, whereas **NumPy** and **Pandas** are employed for analytical computations like highest, lowest, and average marks per subject. The **tabulate** module formats data neatly in console tables, enhancing readability and usability.

The system supports two user roles — *Admin* and *Student*. Admins can log in to add, view, update, or remove student, teacher, and subject records, manage marks, and generate subject-level statistics. Students can securely log in to view their marks, compare performance across subjects, and update personal information. Additionally, features like **sorting** and **binary search** enhance data accessibility and demonstrate algorithmic application within a real-world system. Overall, the portal provides a practical example of how Python can combine data management, security, and computational analysis to build efficient educational tools.

## 1. INTRODUCTION

The *University Portal System* is a Python-based console application designed to manage student, teacher, and academic data efficiently using MySQL as the backend database. It allows administrators to perform key operations like adding, viewing, sorting, and deleting records for students, teachers, and subjects, as well as entering and analyzing marks. Students, on the other hand, can log in to view their performance, update their details, and change passwords. The system focuses on automation, data security, and simplicity, ensuring that academic records are maintained systematically and with minimal human error.

### 1.1 Domain Information

Educational Management Systems (EMS) are widely used in schools and universities to manage academic data and streamline administrative processes. This project falls under that domain, offering a command-line interface version of a basic EMS. It demonstrates the use of structured databases, data integrity through relational design, and user authentication — all essential components of a real-world educational software solution. By integrating database management and analytical computation, the project provides a complete backend-driven academic record management system suitable for small-scale institutional use.

### 1.2 Software Libraries Used

The project leverages a variety of Python modules to enhance functionality, security, and data presentation:

- **mysql.connector** – for connecting Python to the MySQL database and performing CRUD operations.

- **hashlib** – to securely store user passwords by converting them into SHA-256 hash strings.

- **getpass** – for hidden password input during login and password changes, ensuring privacy.

- **re (Regular Expressions)** – used to validate email formats and maintain input accuracy.

- **NumPy** – to calculate statistical metrics like maximum, minimum, and average marks.

- **Pandas** – for structured data manipulation and displaying tabular student or mark records.

- **tabulate** – to display formatted tables in the console, improving readability.

- **sys** – used for program exit handling and system-level operations.

Each of these libraries contributes to making the system robust, secure, and user-friendly while demonstrating how external packages extend Python's core functionality for professional applications.

**1.3 Challenges Faced**

During development, several challenges arose that required both logical and technical problem-solving:

1. **Database Connectivity** – establishing and maintaining a stable connection between Python and MySQL, especially when handling multiple operations and constraints.

2. **Secure Authentication** – ensuring passwords are encrypted and validated correctly without storing them in plain text.

3. **Data Validation** – managing invalid or incomplete user inputs and preventing system crashes.

4. **Data Integrity** – maintaining referential consistency between tables (students, teachers, subjects, and marks).

5. **User Interface Clarity** – designing a console-based flow that remains intuitive and easy to navigate for both admin and student roles.

6. **Statistical Computation** – integrating NumPy and Pandas to calculate and present analytics in a readable, well-formatted way.

## 2. Problem Statement and Objectives

### 2.1 Problem Statement

In many educational institutions, academic record management is still performed manually or with minimal automation. This often leads to data redundancy, inconsistencies, and security risks. Tasks such as registering students, assigning teachers, managing subjects, or recording marks require considerable administrative effort and are prone to human error.
The lack of an integrated system for both administrators and students results in inefficient workflows and unreliable data tracking. Moreover, without analytical tools, comparing student performance or computing statistics like average or highest marks becomes tedious and time-consuming.

To address these issues, the *University Portal System* aims to create a secure, centralized, and automated solution. It enables institutions to store, update, and retrieve student-related information while providing analytical insights into academic performance through Python's computational libraries.

### 2.2 Objectives

The main objectives of the *University Portal System* are derived from the code's modular structure and implemented features. These objectives include:

1. **Database Integration:**
   To develop a MySQL-backed system that persistently stores information about students, teachers, subjects, and marks, ensuring structured and reliable data management.

2. **User Authentication and Security:**
   To implement secure login systems for both admin and student users using the `hashlib` module for password hashing and `getpass` for concealed input, ensuring that sensitive credentials are never exposed.

3. **Administrative Control (CRUD Operations):**
   To allow administrators to perform core operations such as adding, viewing, deleting, and sorting students, teachers, and subjects, as well as updating marks,

through a structured and menu-driven interface.

4. **Student Portal Functionality:**
   To provide a personalized portal for students where they can log in to view marks, check subject statistics, update their personal details, and change their passwords.

5. **Analytical Features:**
   To use **NumPy** and **Pandas** for computing and displaying statistical information (highest, lowest, and average marks) that helps analyze academic performance efficiently.

6. **Search and Sorting Algorithms:**
   To demonstrate algorithmic concepts by implementing **binary search** for quick student lookup and **sorting** functions to organize records by roll number, name, or class.

7. **Data Validation and Integrity:**
   To employ regular expressions (`re`) for validating email inputs and enforce relational constraints in MySQL tables to maintain consistency across data entities.

8. **Demonstration and Usability:**
   To create demo data automatically for testing the system, ensuring the project can be easily run and evaluated in different environments without extensive setup.

## 3. IMPLEMENTATION

```
import mysql.connector

import hashlib

import getpass

import sys

import numpy as np

import pandas as pd

import re

from tabulate import tabulate


DB_CONFIG =
{"host":"localhost","user":"root","password":"root","database":"university
_portal"}


def hash_pass(p):

    return hashlib.sha256(p.encode()).hexdigest()


def connect_db():

    conn = mysql.connector.connect(host=DB_CONFIG["host"],
user=DB_CONFIG["user"], password=DB_CONFIG["password"])

    cur = conn.cursor()

    cur.execute("CREATE DATABASE IF NOT EXISTS university_portal")

    conn.commit()

    conn.close()
```

```
conn = mysql.connector.connect(**DB_CONFIG)

cur = conn.cursor()

cur.execute("""

CREATE TABLE IF NOT EXISTS admins(

    id INT AUTO_INCREMENT PRIMARY KEY,

    username VARCHAR(100) UNIQUE,

    passhash VARCHAR(256)

)""")

cur.execute("""

CREATE TABLE IF NOT EXISTS teachers(

    id INT AUTO_INCREMENT PRIMARY KEY,

    teacher_id VARCHAR(50) UNIQUE,

    name VARCHAR(100),

    email VARCHAR(100),

    phone VARCHAR(20)

)""")

cur.execute("""

CREATE TABLE IF NOT EXISTS students(

    id INT AUTO_INCREMENT PRIMARY KEY,

    roll VARCHAR(50) UNIQUE,

    name VARCHAR(100),

    email VARCHAR(100),

    phone VARCHAR(20),

    class VARCHAR(50),
        passhash VARCHAR(256)

    )""")

    cur.execute("""

    CREATE TABLE IF NOT EXISTS subjects(

        id INT AUTO_INCREMENT PRIMARY KEY,

        code VARCHAR(50) UNIQUE,

        name VARCHAR(100),

        teacher_id VARCHAR(50)

    )""")

    cur.execute("""

    CREATE TABLE IF NOT EXISTS marks(

        id INT AUTO_INCREMENT PRIMARY KEY,

        student_roll VARCHAR(50),

        subject_code VARCHAR(50),

        marks FLOAT,

        FOREIGN KEY (student_roll) REFERENCES students(roll) ON DELETE
CASCADE

    )""")

    conn.commit()

    cur.execute("SELECT COUNT(*) FROM admins")

    if cur.fetchone()[0] == 0:

        cur.execute("INSERT INTO admins(username,passhash)
VALUES(%s,%s)",("admin",hash_pass("admin")))

    conn.commit()
    return conn


conn = connect_db()

cursor = conn.cursor()


def validate_email(e):

    return re.match(r"[^@]+@[^@]+\.[^@]+", e)


def admin_login():

    uname = input("Enter admin username: ").strip()

    pw = getpass.getpass("Enter admin password: ")

    cursor.execute("SELECT passhash FROM admins WHERE 
username=%s",(uname,))

    r = cursor.fetchone()

    if r and r[0]==hash_pass(pw):

        print("Login Successful! Welcome Admin.")

        admin_menu()

    else:

        print("Invalid credentials.")


def student_login():

    roll = input("Enter your roll: ").strip()

    pw = getpass.getpass("Enter your password: ")

    cursor.execute("SELECT passhash FROM students WHERE roll=%s",(roll,))
    r = cursor.fetchone()
    if r and r[0]==hash_pass(pw):
        print(f"Welcome {roll}")
        student_menu(roll)
    else:
        print("Invalid credentials or student not found.")


def change_admin_password():
    uname = input("Admin username to change: ").strip()
    pw = getpass.getpass("Old password: ")
    cursor.execute("SELECT passhash FROM admins WHERE 
username=%s",(uname,))
    r = cursor.fetchone()
    if not r or r[0]!=hash_pass(pw):
        print("Authentication failed.")
        return
    newp = getpass.getpass("New password: ")
    newp2 = getpass.getpass("Confirm new password: ")
    if newp!=newp2:
        print("Mismatch.")
        return
    cursor.execute("UPDATE admins SET passhash=%s WHERE 
username=%s",(hash_pass(newp),uname))
    conn.commit()
    print("Password updated.")


def add_teacher():
    tid = input("Teacher ID: ").strip()
    name = input("Name: ").strip()
    email = input("Email: ").strip()
    phone = input("Phone: ").strip()
    if not validate_email(email):
        print("Invalid email.")
        return
    try:
        cursor.execute("INSERT INTO teachers(teacher_id,name,email,phone)
VALUES(%s,%s,%s,%s)",(tid,name,email,phone))
        conn.commit()
        print("Teacher added.")
    except mysql.connector.IntegrityError:
        print("Teacher ID already exists.")


def view_teachers():
    cursor.execute("SELECT teacher_id,name,email,phone FROM teachers")
    data = cursor.fetchall()
    print(tabulate(data, headers=["ID","Name","Email","Phone"],
tablefmt="psql"))
```

```
def remove_teacher():

    tid = input("Teacher ID to remove: ").strip()

    cursor.execute("DELETE FROM teachers WHERE teacher_id=%s",(tid,))

    conn.commit()

    print("Removed if existed.")


def add_student():

    roll = input("Roll: ").strip()

    name = input("Name: ").strip()

    email = input("Email: ").strip()

    phone = input("Phone: ").strip()

    cls = input("Class: ").strip()

    pw = getpass.getpass("Set password for student: ")

    if not validate_email(email):

        print("Invalid email.")

        return

    try:

        cursor.execute("INSERT INTO
students(roll,name,email,phone,class,passhash) VALUES(%s,%s,%s,%s,%s,%s)",

                       (roll,name,email,phone,cls,hash_pass(pw)))

        conn.commit()

        print("Student added.")

    except mysql.connector.IntegrityError:

        print("Roll already exists.")
```

```python
def view_students_df():
    cursor.execute("SELECT roll,name,email,phone,class FROM students")
    rows = cursor.fetchall()
    df = pd.DataFrame(rows,
columns=["roll","name","email","phone","class"])
    if df.empty:
        print("No students.")
    else:
        print(df.to_string(index=False))


def remove_student():
    roll = input("Roll to remove: ").strip()
    cursor.execute("DELETE FROM students WHERE roll=%s",(roll,))
    conn.commit()
    print("Removed if existed.")


def add_subject():
    code = input("Subject code: ").strip()
    name = input("Subject name: ").strip()
    teacher_id = input("Assigned teacher id (optional): ").strip() or None
    try:
        cursor.execute("INSERT INTO subjects(code,name,teacher_id)
VALUES(%s,%s,%s)",(code,name,teacher_id))
```

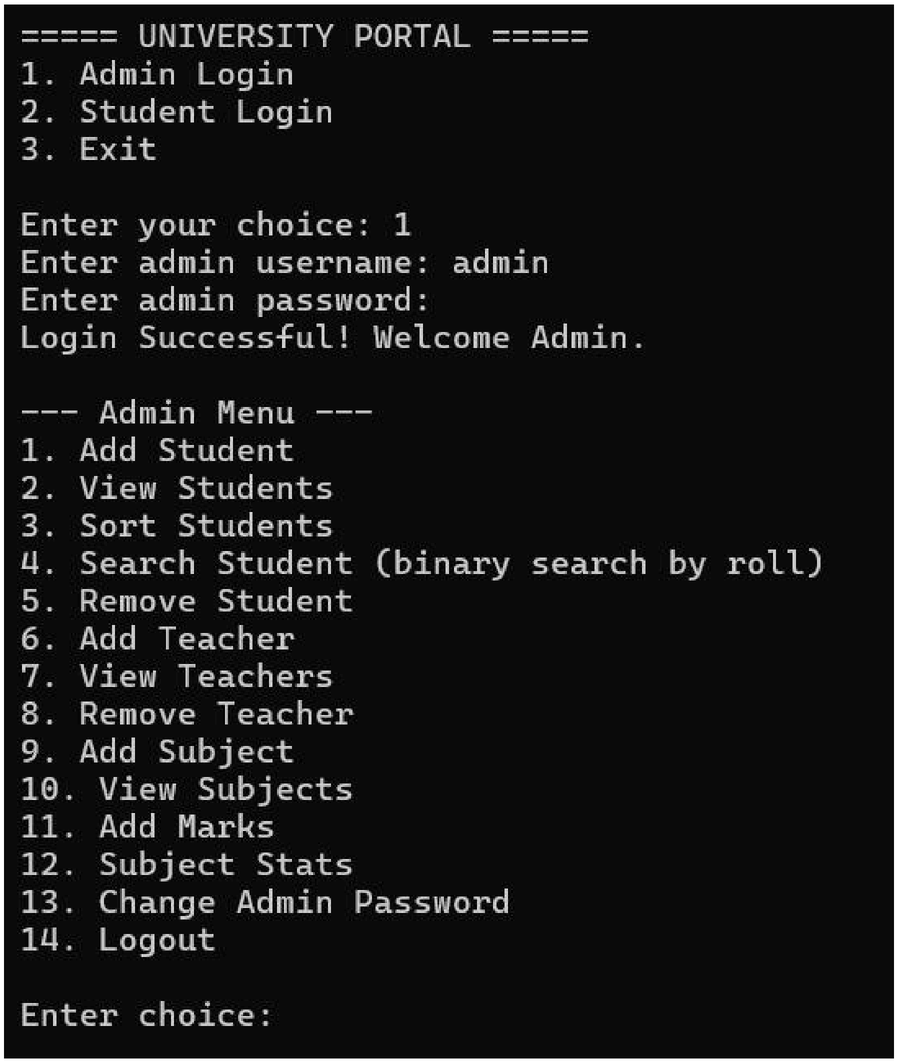
```
        conn.commit()
        print("Subject added.")
    except mysql.connector.IntegrityError:
        print("Subject code exists.")


def view_subjects():
    cursor.execute("SELECT code,name,teacher_id FROM subjects")
    rows = cursor.fetchall()
    print(tabulate(rows, headers=["Code","Name","TeacherID"],
tablefmt="psql"))


def add_marks():
    roll = input("Student roll: ").strip()
    subj = input("Subject code: ").strip()
    try:
        m = float(input("Marks: ").strip())
    except:
        print("Invalid marks.")
        return
    cursor.execute("SELECT 1 FROM students WHERE roll=%s",(roll,))
    if not cursor.fetchone():
        print("Student not found.")
        return
    cursor.execute("SELECT 1 FROM subjects WHERE code=%s",(subj,))
    if not cursor.fetchone():
        print("Subject not found.")
        return
    cursor.execute("INSERT INTO marks(student_roll,subject_code,marks)
VALUES(%s,%s,%s)",(roll,subj,m))
    conn.commit()
    print("Marks added.")

def view_marks_for_student(roll):
    cursor.execute("SELECT subject_code,marks FROM marks WHERE
student_roll=%s",(roll,))
    rows = cursor.fetchall()
    if not rows:
        print("No marks.")
        return
    df = pd.DataFrame(rows, columns=["subject","marks"])
    grouped = df.groupby("subject")["marks"].agg(list).reset_index()
    stats = []
    for _, r in grouped.iterrows():
        arr = np.array(r["marks"],dtype=float)
        stats.append((r["subject"], arr.max(), arr.mean(), arr.min()))
    print(tabulate(stats,
headers=["Subject","Highest","Average","Lowest"], tablefmt="psql"))
    print("\nYour marks:")
    print(tabulate(rows, headers=["Subject","Marks"], tablefmt="psql"))
```

```python
def stats_for_subject(subj):

    cursor.execute("SELECT marks FROM marks WHERE
subject_code=%s",(subj,))

    rows = cursor.fetchall()

    if not rows:

        print("No marks for this subject.")

        return

    arr = np.array([r[0] for r in rows],dtype=float)

    print(f"Subject {subj} -> Highest: {arr.max():.2f}, Average:
{arr.mean():.2f}, Lowest: {arr.min():.2f}")


def binary_search_students(sorted_list, key):

    lo = 0

    hi = len(sorted_list)-1

    while lo<=hi:

        mid = (lo+hi)//2

        if sorted_list[mid][0]==key:

            return sorted_list[mid]

        elif sorted_list[mid][0]<key:

            lo = mid+1

        else:

            hi = mid-1

    return None
```

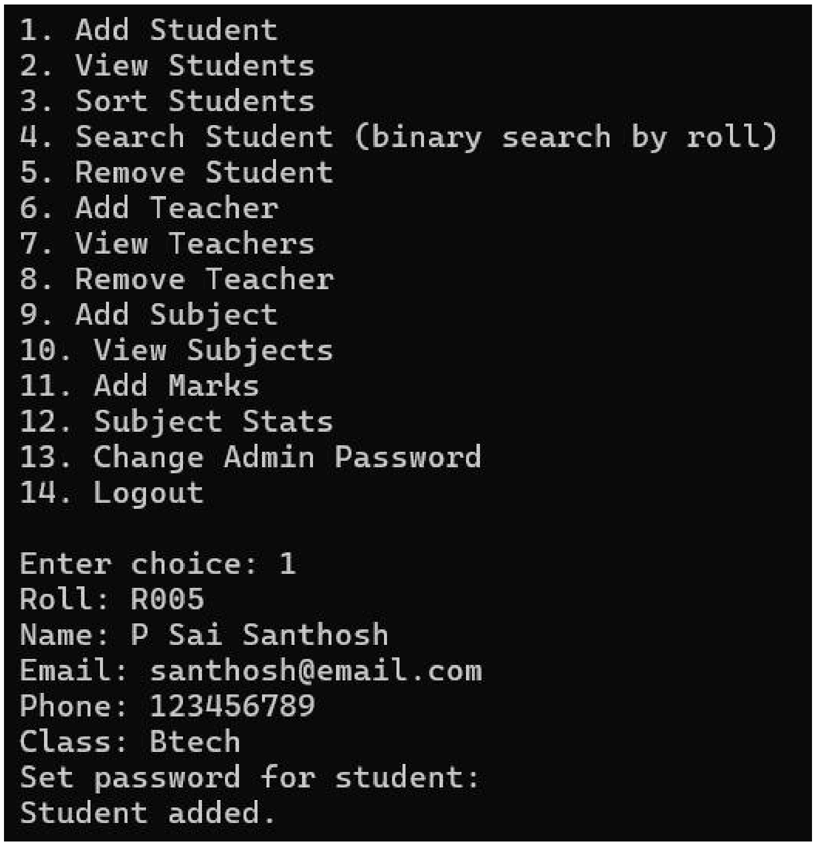
```
def sort_students_in_db(by="roll"):
    cursor.execute("SELECT roll,name,email,phone,class FROM students")
    rows = cursor.fetchall()
    if not rows:
        print("No students.")
        return
    if by not in ["roll","name","class"]:
        by="roll"
    idx = {"roll":0,"name":1,"class":4}[by]
    rows_sorted = sorted(rows, key=lambda x: x[idx])
    print(tabulate(rows_sorted,
headers=["roll","name","email","phone","class"], tablefmt="psql"))
    return rows_sorted


def search_student_binary():
    rows_sorted = sort_students_in_db(by="roll")
    if not rows_sorted:
        return
    key = input("Enter roll to search: ").strip()
    res = binary_search_students([(r[0],r[1],r[2],r[3],r[4]) for r in
rows_sorted], key)
    if res:
        print("Found:", res)
    else:

        print("Not found.")


def change_student_password(roll=None):

    if not roll:

        roll = input("Student roll: ").strip()

    old = getpass.getpass("Old password: ")

    cursor.execute("SELECT passhash FROM students WHERE roll=%s",(roll,))

    r = cursor.fetchone()

    if not r or r[0]!=hash_pass(old):

        print("Auth failed.")

        return

    new = getpass.getpass("New password: ")

    new2 = getpass.getpass("Confirm new password: ")

    if new!=new2:

        print("Mismatch.")

        return

    cursor.execute("UPDATE students SET passhash=%s WHERE
roll=%s",(hash_pass(new),roll))

    conn.commit()

    print("Password updated.")


def admin_menu():

    while True:
        print("""
--- Admin Menu ---
1. Add Student
2. View Students
3. Sort Students
4. Search Student (binary search by roll)
5. Remove Student
6. Add Teacher
7. View Teachers
8. Remove Teacher
9. Add Subject
10. View Subjects
11. Add Marks
12. Subject Stats
13. Change Admin Password
14. Logout
""")
        c = input("Enter choice: ").strip()
        if c=="1":
            add_student()
        elif c=="2":
            view_students_df()
        elif c=="3":
            key = input("Sort by (roll/name/class): ").strip()
            sort_students_in_db(by=key)
        elif c=="4":
            search_student_binary()
        elif c=="5":
            remove_student()
        elif c=="6":
            add_teacher()
        elif c=="7":
            view_teachers()
        elif c=="8":
            remove_teacher()
        elif c=="9":
            add_subject()
        elif c=="10":
            view_subjects()
        elif c=="11":
            add_marks()
        elif c=="12":
            subj = input("Subject code for stats: ").strip()
            stats_for_subject(subj)
        elif c=="13":
            change_admin_password()
        elif c=="14":
            break
        else:

            print("Invalid choice.")


def student_update_info(roll):

    print("Leave blank to keep current.")

    cursor.execute("SELECT name,email,phone,class FROM students WHERE 
roll=%s",(roll,))

    r = cursor.fetchone()

    if not r:

        print("Not found.")

        return

    name = input(f"Name [{r[0]}]: ").strip() or r[0]

    email = input(f"Email [{r[1]}]: ").strip() or r[1]

    phone = input(f"Phone [{r[2]}]: ").strip() or r[2]

    cls = input(f"Class [{r[3]}]: ").strip() or r[3]

    if not validate_email(email):

        print("Invalid email.")

        return

    cursor.execute("UPDATE students SET name=%s,email=%s,phone=%s,class=%s 
WHERE roll=%s",(name,email,phone,cls,roll))

    conn.commit()

    print("Updated.")


def student_view_teachers(roll):
    cursor.execute("SELECT s.code,s.name,t.name FROM subjects s LEFT JOIN 
teachers t ON s.teacher_id=t.teacher_id")

    rows = cursor.fetchall()

    print(tabulate(rows, headers=["Subject Code","Subject 
Name","Teacher"], tablefmt="psql"))


def student_view_performance(roll):

    cursor.execute("SELECT subject_code,marks FROM marks WHERE 
student_roll=%s",(roll,))

    rows = cursor.fetchall()

    if not rows:

        print("No marks.")

        return

    df = pd.DataFrame(rows, columns=["subject","marks"])

    subj_list = df['subject'].unique().tolist()

    out=[]

    for subj in subj_list:

        cursor.execute("SELECT marks FROM marks WHERE 
subject_code=%s",(subj,))

        all_marks = [r[0] for r in cursor.fetchall()]

        arr = np.array(all_marks,dtype=float)

        highest = arr.max()

        average = arr.mean()

        lowest = arr.min()

        your = df[df.subject==subj]["marks"].iloc[0]
        out.append((subj,your,highest,average,lowest))

    print(tabulate(out, headers=["Subject","Your
Marks","Highest","Average","Lowest"], tablefmt="psql"))


def student_menu(roll):

    while True:

        print(f"""

--- Student Menu ({roll}) ---

1. Update Personal Information

2. View Teachers and Subjects

3. View Marks & Relative Performance

4. Sort Subjects (by code)

5. Change Password

6. Logout

""")

        c = input("Enter choice: ").strip()

        if c=="1":

            student_update_info(roll)

        elif c=="2":

            student_view_teachers(roll)

        elif c=="3":

            student_view_performance(roll)

        elif c=="4":

            cursor.execute("SELECT code,name FROM subjects")
            rows = sorted(cursor.fetchall(), key=lambda x: x[0])
            print(tabulate(rows, headers=["Code","Name"],
tablefmt="psql"))
        elif c=="5":
            change_student_password(roll)
        elif c=="6":
            break
        else:
            print("Invalid choice.")


def create_demo_data():
    cursor.execute("SELECT COUNT(*) FROM students")
    if cursor.fetchone()[0]==0:
        sample = [

("R001","Alice","alice@example.com","9999999991","BSc",hash_pass("pass1"))
,

("R002","Bob","bob@example.com","9999999992","BSc",hash_pass("pass2")),

("R003","Charlie","charlie@example.com","9999999993","BSc",hash_pass("pass
3"))
        ]
        cursor.executemany("INSERT INTO
students(roll,name,email,phone,class,passhash)
VALUES(%s,%s,%s,%s,%s,%s)",sample)
```

```
    cursor.execute("SELECT COUNT(*) FROM subjects")

    if cursor.fetchone()[0]==0:

        subs =
[("MATH101","Calculus","T1"),("PHY101","Physics","T2"),("CS101","Programmi
ng","T3")]

        cursor.executemany("INSERT INTO subjects(code,name,teacher_id)
VALUES(%s,%s,%s)",subs)

    cursor.execute("SELECT COUNT(*) FROM teachers")

    if cursor.fetchone()[0]==0:

        t = [("T1","Dr. Euler","euler@uni.edu","9000000001"),("T2","Dr.
Newton","newton@uni.edu","9000000002"),("T3","Dr.
Turing","turing@uni.edu","9000000003")]

        cursor.executemany("INSERT INTO
teachers(teacher_id,name,email,phone) VALUES(%s,%s,%s,%s)",t)

    cursor.execute("SELECT COUNT(*) FROM marks")

    if cursor.fetchone()[0]==0:

        m = [


("R001","MATH101",85.0),("R001","PHY101",78.0),("R001","CS101",92.0),


("R002","MATH101",65.0),("R002","PHY101",72.0),("R002","CS101",80.0),


("R003","MATH101",90.0),("R003","PHY101",88.0),("R003","CS101",85.0)

        ]

        cursor.executemany("INSERT INTO
marks(student_roll,subject_code,marks) VALUES(%s,%s,%s)",m)

    conn.commit()
    print("Demo data created/ensured.")


def main_menu():
    create_demo = input("Create demo data? (y/n) [y]: ").strip().lower()
or "y"
    if create_demo=="y":
        create_demo_data()
    while True:
        print("""
===== UNIVERSITY PORTAL =====
1. Admin Login
2. Student Login
3. Exit
""")
        ch = input("Enter your choice: ").strip()
        if ch=="1":
            admin_login()
        elif ch=="2":
            student_login()
        elif ch=="3":
            print("Goodbye.")
            conn.close()
            sys.exit(0)
        else:
            print("Invalid choice.")


if __name__=="__main__":
    main_menu()
```

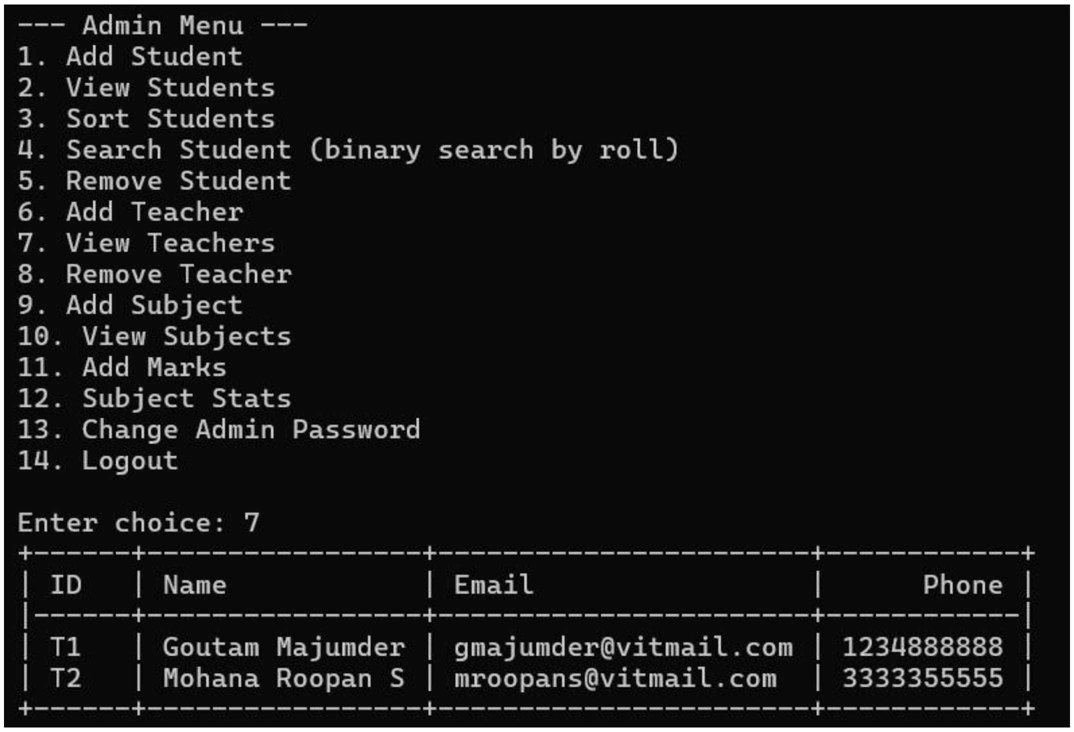
## 4. DEMO SCREENSHOTS



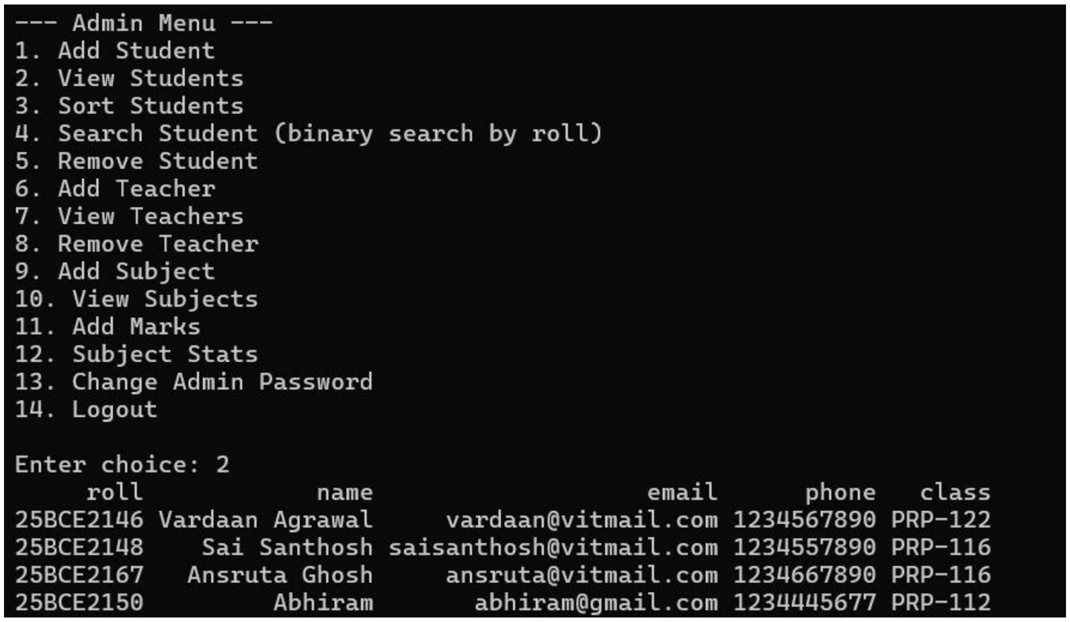
```
===== UNIVERSITY PORTAL =====
1. Admin Login
2. Student Login
3. Exit

Enter your choice: 1
Enter admin username: admin
Enter admin password:
Login Successful! Welcome Admin.

--- Admin Menu ---
1. Add Student
2. View Students
3. Sort Students
4. Search Student (binary search by roll)
5. Remove Student
6. Add Teacher
7. View Teachers
8. Remove Teacher
9. Add Subject
10. View Subjects
11. Add Marks
12. Subject Stats
13. Change Admin Password
14. Logout

Enter choice:
```



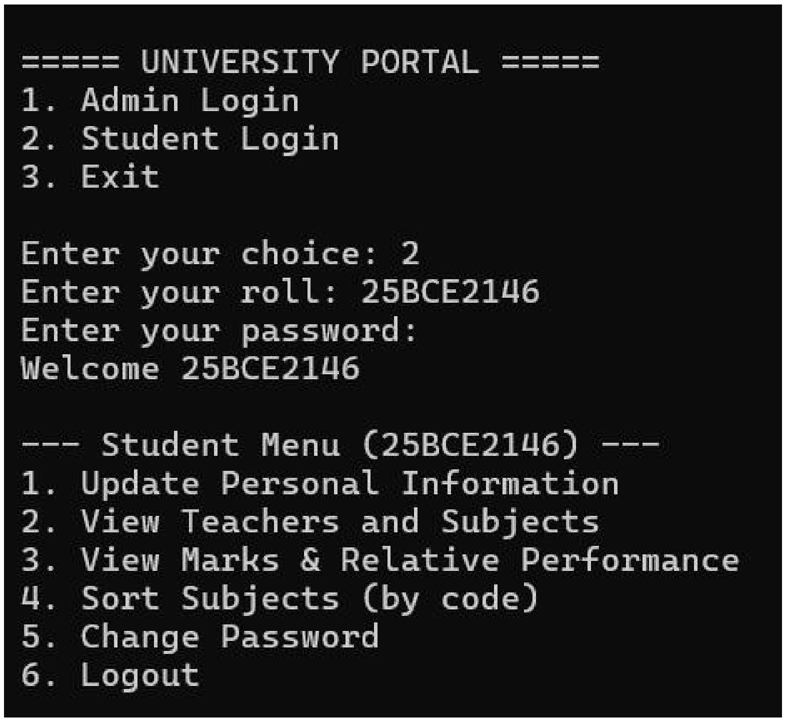
```
1. Add Student
2. View Students
3. Sort Students
4. Search Student (binary search by roll)
5. Remove Student
6. Add Teacher
7. View Teachers
8. Remove Teacher
9. Add Subject
10. View Subjects
11. Add Marks
12. Subject Stats
13. Change Admin Password
14. Logout

Enter choice: 1
Roll: R005
Name: P Sai Santhosh
Email: santhosh@email.com
Phone: 123456789
Class: Btech
Set password for student:
Student added.
```



```
--- Admin Menu ---
1. Add Student
2. View Students
3. Sort Students
4. Search Student (binary search by roll)
5. Remove Student
6. Add Teacher
7. View Teachers
8. Remove Teacher
9. Add Subject
10. View Subjects
11. Add Marks
12. Subject Stats
13. Change Admin Password
14. Logout

Enter choice: 7
+------+-----------------+-----------------------+------------+
| ID   | Name            | Email                 |      Phone |
|------+-----------------+-----------------------+------------|
| T1   | Goutam Majumder | gmajumder@vitmail.com | 1234888888 |
| T2   | Mohana Roopan S | mroopans@vitmail.com  | 3333355555 |
+------+-----------------+-----------------------+------------+
```



```
--- Admin Menu ---
1. Add Student
2. View Students
3. Sort Students
4. Search Student (binary search by roll)
5. Remove Student
6. Add Teacher
7. View Teachers
8. Remove Teacher
9. Add Subject
10. View Subjects
11. Add Marks
12. Subject Stats
13. Change Admin Password
14. Logout

Enter choice: 2
     roll            name                   email      phone   class
25BCE2146 Vardaan Agrawal     vardaan@vitmail.com 1234567890 PRP-122
25BCE2148    Sai Santhosh saisanthosh@vitmail.com 1234557890 PRP-116
25BCE2167   Ansruta Ghosh     ansruta@vitmail.com 1234667890 PRP-116
25BCE2150         Abhiram       abhiram@gmail.com 1234445677 PRP-112
```



```
===== UNIVERSITY PORTAL =====
1. Admin Login
2. Student Login
3. Exit

Enter your choice: 2
Enter your roll: 25BCE2146
Enter your password:
Welcome 25BCE2146

--- Student Menu (25BCE2146) ---
1. Update Personal Information
2. View Teachers and Subjects
3. View Marks & Relative Performance
4. Sort Subjects (by code)
5. Change Password
6. Logout
```

## 5. CONCLUSION

The *University Portal System* successfully demonstrates how Python can be used to build an efficient, secure, and scalable academic management solution. By integrating **MySQL** for persistent data storage and using libraries like **NumPy**, **Pandas**, and **Tabulate**, the system effectively handles both database management and analytical computations within a simple command-line interface.

The project achieves its core goal of automating student and teacher data handling while emphasizing data integrity, validation, and security through password hashing and structured input management. The inclusion of sorting and binary search functions showcases algorithmic understanding and practical application of computational logic in a real-world context.

From the administrator's perspective, the portal streamlines record management and performance analysis; for students, it provides an accessible platform to view marks and track academic progress. Beyond its immediate functionality, the project serves as a strong foundational model that can be extended into a full-fledged web or GUI-based application with additional modules for attendance tracking, grading systems, and automated reporting.

Overall, the *University Portal System* highlights how a well-structured Python program can integrate multiple concepts — database design, security, algorithms, and analytics — into a cohesive, real-world software solution for educational management.