VALLABA VIDYALAYA

CBSE AFFL. 1930722 | NIOS AI 190182

PROJECT REPORT ON

STUDENT DATA ANALYSIS

ACADEMIC YEAR - 2022-23

Roll. No: 00000218
Name: Shree Nandha A B
Class: XII–B
Subject: Informatics Practices
Subject Code: 065
Project Guide: Mrs. Madheena Banu (PGT - Comp.Sc)

Certificate

This is to certify that The Project/Dissertation entitled College data analysis. This a bonafide work done by Master Shree Nandha A B of class XII session 2021-22 in partial fulfillment of CBSE Examination and has been carried out under my direct supervision and guidance. This report or a similar report on the topic has not been submitted for any other examination and does not form a part of any other course undergone by the candidate.

Student (Shree Nandha A B)

Teacher (Madheena Banu)

Principal (Navaneetha Krishnan V)

Table of Contents

Acknowledgement

In the realm of project endeavors, the climax of success depends not just on my personal efforts but also significantly on the collective influence and astute guidance of many people. I would want to use this opportunity to express my sincere gratitude to everyone who contributed in some way to the successful completion of this project. I owe the Divine Providence a deep debt of gratitude for giving me the resolve necessary to see the project through to completion.

Unconditional thanks is expressed to my parents, whose unflinching support served as an unwavering basis for this endeavor.

The people whose work helped this project reach its current level of completion deserve the utmost gratitude. I am truly grateful for their unwavering support despite my inherent flaws. I'd want to express my sincere gratitude to the illustrious Principal of Vallaba Vidyalaya, a person who never ceases to inspire and lend a helping hand.

I would like to express my sincere gratitude to Mrs. Madheena Banu, the subject in-charge, for her thoughtful analysis of my project and for helping to overcome obstacles that arose during the implementation stage.

The direction and assistance given by every participant, past and present, who was instrumental in the project's growth, was crucial to its success. The consistency of their support acts as a foundation of appreciation for which I am indebted.

Project Synopsis

Title:

- Engineering Colleges in India

Problem Definition:

- Design a project to analyse the dataset of more than 5000 colleges and represent it graphically, etc.

Contribution/Team members:

- Shree Nandha A B
- Janardan M
- Naveen Sabari Rajhan R S

Team Detail:

- This Project is developed coordinately by us.
- It took approximately 2 days to develop this project, working 5-6 hours per day.
- All modules completed by us, as per hour view and knowledge.

Reason for choosing the Topic:

- The main aim is to make awareness for the students who are unaware of the college's in their locality so this project is to make sure everyone one gets the knowledge of each and every college.

"SOFTWARE IS LIKE SEX,
ITS BETTER WHEN ITS FREE"

- LINUS TORVALDS

Hardware Requirements

- OS:
  - Windows 7 or later,
  - Mac OS 10.10 or later,
  - Linux distros released after Debian (any).

- CPU:
  - Dual-core processor (Intel i3, AMD Rysen 3)
  - Quad-core processor or higher (recommended)

- Disk:
  - 4 GB RAM
  - 10 GB of unallocated space

Software Requirements

- Python 3.10.x or higher
- Required libraries/modules:

`matplotlib`	`3.7.2`
`mysql-connector`	`2.2.9`
`numpy`	`1.25.2`
`pandas`	`2.0.3`
`prettytable`	`3.8.0`
`plotly`	`5.16.1`
`prettytable`	`3.8.0`
`python-dateutil`	`2.8.2`
`SQLAlchemy`	`2.0.20`

Introduction

Welcome to a voyage into the redesigned world of data analysis. This project is more than just a collection of lines of code; it is a demonstration of the ability of software to simplify the process of manipulating complex data. We introduce a novel approach that is poised to revolutionize data analysis with the aim of reducing enormous jobs, saying goodbye to manual labor, and easing the burden on human shoulders.

The days of being buried beneath spreadsheets, choked by mountains of data, and attempting the Herculean job of making sense of confusion are long gone. This research ushers in a new era when sophisticated technology tames complex data sets. Visualize a future in which insights can be gleaned with a few clicks, trends can be seen in vibrant graphs, and laborious analysis is no longer necessary.

We set out on a mission to offer a seamless experience for data aficionados, analysts, and inquisitive minds alike by utilizing the possibilities of contemporary software. We want to develop a tool that not only enhances knowledge but also crunches data through the alchemy of code. This software aims to empower users to confidently dive into data, make defensible judgments, and uncover tales buried inside the digital maze. It is not just about bits and bytes.

Remember the revolutionary potential that software possesses as we delve into the details of this project. Efficiency, accuracy, and creativity are ready to take the place of the manual, the ordinary, and the monotonous. Prepare to see the emergence of a solution that transforms data analysis into an experience that empowers, enlightens, and enriches rather than just a task.

Objectives of the project

This project is primarily concerned with the field of data analysis. It responds to the requirement for automating time-consuming data processing processes, replacing manual labor, and ultimately reducing the related workload.

- **Utilizing Modern Software Tools:** The project encourages the utilization of contemporary software tools to create efficient and effective data analysis solutions.
- **Object-Oriented Programming Principles:** Students are tasked with proficiently incorporating OOP principles while developing solutions, ranging from smaller-scale projects to those of moderate complexity.
- **Procedural Problem Solving:** The project emphasizes the creation of procedural code that adeptly addresses varying degrees of complexity in problem-solving, catering to challenges both small and medium in scope.
- **Comprehensive Understanding of Computer Science:** Participants are expected to demonstrate a comprehensive grasp of computer science principles, spanning domains such as systems, theory, and software development.

Proposed System

- Exclusively relying on flawed human capabilities in today's fiercely competitive environment falls short of the mark.
- The proverb "to err is human" is no longer true because mistakes are no longer excused in the same way.
- A complete embrace of cutting-edge technology is required to thrive in today's competitive environment and get optimal results with accuracy.
- The competitive environment of today demands an adaptable response to the ebbing tides of time, leaving behind outmoded approaches.
- To meet the demands for both efficiency and exactitude in work operations, data management software is a crucial tool.
- The incorporation of software automation has developed into an essential component of a variety of enterprises, streamlining and accelerating their processes all at once.
- Numerous software products are available on the market that are made to meet the complex needs of various types of enterprises.
- Data administration is no longer associated with time-consuming manual tasks, complicated ledger maintenance, and mountains of paperwork.
- Automated processes considerably increase operational efficiency by saving a significant amount of time and resources.

Data Analysis

- **Extracting Insights for Informed Decisions**
    - Data analysis is a process that transforms raw data into actionable insights.
    - It involves cleaning, transforming, and refining data to ensure accuracy and reliability.
    - The process mitigates decision-making risks by uncovering patterns and trends.
    - Insights are often presented through visual representations like charts and graphs.
    - Everyday decisions involving data analysis mirror its essence by considering past events and future outcomes.

- **Benefits of Data Analysis**
    - **Accuracy Enhancement:** Raw data often contains errors and noise that data analysis rectifies.
    - **Risk Mitigation:** Analysis reveals patterns and anomalies, aiding better decision-making.
    - **Visual Representation:** Data insights are often presented visually, making complex relationships understandable.
    - **Evidence-Based Decision-Making:** Analyzing past data informs predictions for future scenarios.

- **Data Analysis in Daily Life**
    - **Umbrella Decision:** Analyzing weather data and forecasts before taking an umbrella.
    - **Core Concept:** Balancing historical patterns with future predictions for well-informed choices.
    - **Combining Retrospective and Prospective:** Data analysis uses historical trends to project future outcomes.

Graphical Representation of Data

- Graphical representation of data is an effective way to visually showcase numerical information, aiding in the analysis of quantitative data.

- Graphs are a type of chart where data is plotted as variables on a coordinate system.

- They allow for easy analysis of how changes in one variable relate to changes in others.

- Different types of graphs and charts are used to represent data visually, including lines, plots, and diagrams.

- Line graphs display trends over time by connecting data points with lines.

- Bar charts use bars to compare data in discrete categories.

- Scatter plots show relationships between two variables using data points.

- Histograms display the frequency distribution of continuous data in intervals.

- Heat-maps use color intensity to show patterns in large datasets.

- These representations enhance clarity, aid decision-making, and offer data-driven insights.

Why is Data Analysis Important?

Better Customer Targeting:

- Avoid wastage of time, resources, and money on irrelevant advertising campaigns.
- Data analysis guides where advertising efforts should be concentrated.

Understanding Target Customers:

- Track product and campaign performance within your target demographic.
- Gain insights into spending habits, disposable income, and areas of interest.
- Helps in pricing, campaign duration, and production projections.

Reducing Operational Costs:

- Identify resource-intensive areas and those with lower productivity.
- Optimize resource allocation, scaling back or eliminating inefficient areas.

Enhanced Problem-Solving:

- Informed decisions result in higher success rates.
- Data-driven insights support better choices and prevent costly mistakes.

Accurate Data Acquisition:

- Informed decisions require accurate data.
- Data analysis ensures relevant, precise information for marketing, planning, and realignment.

Data Analysis Methods

- **Qualitative Data Analysis:** Qualitative data analysis involves deriving insights from non-numeric data sources, such as words, symbols, pictures, and observations. Unlike quantitative methods, it doesn't rely on statistical calculations. Common qualitative analysis methods include:

- **Content Analysis**: This method focuses on analyzing behavioral and verbal data, often from sources like interviews, articles, or documents. It seeks to identify patterns, themes, and meanings within the data.

- **Narrative Analysis**: Narrative analysis involves working with data obtained from interviews, diaries, surveys, or any form of narrative. It aims to uncover the underlying stories and experiences that individuals share.

- **Hypothesis Testing**: This method assesses the validity of a given hypothesis or theory for a specific dataset or demographic. It involves comparing observed data with expected outcomes based on the hypothesis.

- **Mean (Average)**: The mean calculates the overall trend of a set of numbers by adding up all the values and dividing the sum by the number of items in the list. It provides an indicator of the central tendency of the data.

- **Sample Size Determination**: This method involves selecting a small sample from a larger group of people and analyzing it. The results obtained are then considered representative of the entire population. Careful sample size determination is crucial for drawing accurate conclusions.

Types of Graphical Representation

- **Line Graphs**: Depict trends and changes over time by connecting data points with lines.
- **Bar Charts**: Use rectangular bars to compare data across categories or time periods.
- **Pie Charts**: Show proportional distribution of categories within a whole, often expressed in percentages.
- **Scatter Plots**: Display relationships between two variables using individual data points on a grid.
- **Histograms**: Illustrate frequency distribution of continuous data by grouping it into intervals.
- **Area Charts**: Similar to line graphs, but the area beneath the line is shaded, useful for cumulative data.
- **Box Plots:** Represent data distribution and key statistics like median and quartiles.
- **Heatmaps**: Use color intensity to visualize patterns in large datasets or matrices.
- **Pictorial Graphs**: Use pictures or symbols to represent data, such as pictographs or icon arrays.
- **Stacked Bar Charts**: Display subcategories within larger categories as stacked bars.
- **Radar Charts**: Compare multiple variables using a radial arrangement of axes.

Source-file structure

```
.
├── connectors/
│   ├── config.py
│   ├── init.py
│   └── insert.py
│
├── datasets/
│   └── colleges.csv
│
├── src/
│   └── utils.py
│
├── main.py
└── requirements.txt
```

Source code

./connectors/config.py

```
""" Configurations to establish MySQL connection """
"""
 host - <string; | <...> >
 user - <string; | <...> >
 password - <string; | <...> >
 accentColor - <colorResolvable; | <...> >
 projectName - <string; | <...> >
"""
host = "localhost"
user = "nan"
password = "Almightynan012@3"
accentColor = "black"
fontColor = "white"
projectName = "project mayhem"
```

Source code

./connectors/init.py

```
import mysql.connector as msc
from src.utils import CustomLogger as logger

logger = logger()

class Initialize:
 """Queries sent to the database on startup"""

 def db(db):
 logger.log(
 "SUCCESS", "Handshake success, sending queries to create
database..."
 )
 # Create a database named 'library' if it doesn't exist
 db.execute("CREATE DATABASE IF NOT EXISTS library;")
 # Switch to the 'library' database for further operations
 db.execute("USE library;")
def movies(db):
 logger.log("DEBUG", "Creating table 'movies'...")
 try:
 # Create the 'movies' table if it doesn't exist, with
specific columns and data types
 db.execute(
 """
 CREATE TABLE IF NOT EXISTS movies (
 id INT(5) PRIMARY KEY,
 title VARCHAR(100) UNIQUE NOT NULL,
 overview TEXT,
 original_language VARCHAR(10) DEFAULT 'Unknown'
 vote_count int(5) DEFAULT 1,
 vote_average int(5) DEFAULT 1
 );
 """
 )
 logger.log(
  "SUCCESS", "Created table 'movies', switching to the next
statement..."
 )
  # Commit the changes to the database after the successful
table creation
 db.execute("COMMIT;")
 except msc.Error as err:
  # If the 'movies' table already exists, skip the table
creation
 if err.errno == msc.errorcode.ER_TABLE_EXISTS_ERROR:
   logger.log("WARNING", "Existing table 'movies' found,
skipping...")
```

Source code

./connectors/insert.py

```
import mysql.connector as msc
from src.utils import CustomLogger as logger
import pandas as pd
from unidecode import unidecode

# Initialize the logger
logger = logger()

class Insert:
 """Class to insert extracted values from a CSV file into the
SQL table"""

 @staticmethod
 def from_csv(db, csv_file_path):
 # Log debug information: Reading data from the CSV file
   logger.log("DEBUG", f"Reading data from CSV file:
{csv_file_path}")

 try:
 # Read the CSV file into a pandas DataFrame
 csv_data = pd.read_csv(csv_file_path)

 # Create the 'movies' table if it does not exist
 db.execute(
 """
 CREATE TABLE IF NOT EXISTS movies (
 id INT(5) PRIMARY KEY,
 title VARCHAR(100) UNIQUE NOT NULL,
 overview TEXT DEFAULT NULL,
 vote_count INT(5) DEFAULT 0,
 vote_average INT(5) DEFAULT 0
 );
 """
 )
 logger.log("SUCCESS", "Created table 'movies', switching to the
next statement...")
 db.execute("COMMIT;") # Commit the transaction after table
creation

 logger.log("DEBUG", "Inserting data into table 'movies'...")

 # SQL query to insert data into the 'movies' table
 insert_query = "INSERT INTO movies (id, title, overview,
original_language, vote_count, vote_average) VALUES (%s, %s, %s,
%s, %s, %s)"

 # Set 'cursor' as 'db', assuming that 'db' is a MySQL cursor
 cursor = db

 # Loop through each row in the CSV data and insert it into the
table
 for _, row in csv_data.iterrows():
 id = int(row["id"])

 # Preprocess the 'title', 'overview', and 'original_language'
fields
 title = str(row["title"])[:100]
 title = "".join(["*" if c in "!@#$%^&*" else c for c in title])
 title = unidecode(title)

 overview = str(row["overview"])
 overview = "".join(["*" if c in "!@#$%^&*" else c for c in
overview])
 overview = unidecode(overview)
```

```
values = (
 id,
 title,
 overview,
 original_language,
 vote_count,
 vote_average,
 )
 try:
 # Execute the insert query with the current row's values
 cursor.execute(insert_query, values)
  cursor.execute("COMMIT;") # Commit the transaction after
insertion
 except msc.Error as err:
 # If the row is a duplicate entry, log a warning and skip
 if err.errno == msc.errorcode.ER_DUP_ENTRY:
 logger.log(
 "WARNING",
   f"Duplicate entry: {values}. Skipping to prevent
duplicates...",
 )
 else:
 logger.log(
 "ERROR",
  f"An error occurred while inserting row: {values}. Error:
{err}",
 )
 db.execute("ROLLBACK;") # Rollback the transaction
 break # Exit the loop on the first error
  logger.log("SUCCESS", "Inserted data from CSV into the
'movies' table.")
 except msc.Error as err:
 # Log an error if an error occurs during the process
 logger.log("ERROR", f"An error occurred: {err}")
 db.execute("ROLLBACK;") # Rollback the transaction
```

Source code

./src/utils.py

```
import datetime
import os

class CustomLogger:
 """
 Coloured logging with timestamp and custom types.
 """

 os.system('')
 # Define ANSI escape codes for colored output
 RESET = "\033[0m"
 RED = "\033[91m"
 GREEN = "\033[92m"
 YELLOW = "\033[93m"
 BLUE = "\033[94;1m"
 MAGENTA = "\033[95m"
 CYAN = "\033[96m"

 def __init__(self):
 pass

 def get_timestamp(self):
 """Get the current timestamp in HH:MM:SS AM/PM format."""
 now = datetime.datetime.now().strftime("%I:%M:%S %p")
 return f"[{now}]"

 def log(self, level, message):
  """Print a colored log message to stdout based on the
levels."""
```

```
 log_levels = {
 "START": self.CYAN,
 "SUCCESS": self.GREEN,
 "INFO": self.CYAN,
 "ERROR": self.RED,
 "DEBUG": self.BLUE,
 "WARNING": self.YELLOW,
 }
 timestamp = self.get_timestamp()
 formatted_message = f"{timestamp} › {log_levels.get(level,
'')}{level: <8}{self.RESET} {message}"
 print(formatted_message)

if __name__ == "__main__":
 # Create an instance of the CustomLogger class
 logger = CustomLogger()
```

Source code

./main.py

```
import mysql.connector as msc
import connectors.config as config
from prettytable import PrettyTable
from src.utils import CustomLogger as logger
from connectors.init import Initialize
from connectors.insert import Insert
import tkinter as tk
from tkinter import messagebox, simpledialog
from tkinter.ttk import Button, Combobox
import threading
from tkinter.simpledialog import askinteger
import pandas as pd
from prettytable import PrettyTable
from tkinter import StringVar
from tkinter import ttk, font
import matplotlib.pyplot as plt
from tkinter import filedialog
from sqlalchemy import create_engine

def center_window(window):
 window.update_idletasks() # Make sure window size is updated
 width = window.winfo_width()
 height = window.winfo_height()

 screen_width = window.winfo_screenwidth()
 screen_height = window.winfo_screenheight()

 x = (screen_width - width) // 2
 y = (screen_height - height) // 2
 window.geometry(f"{width}x{height}+{x}+{y}")

def show_credits():
 title_label.config(text="Credits", font=("Montserrat", 16,
"bold"))
 label.config(
 text="<placeholder>",
 font=("Montserrat", 12),
 )
 enter_button.pack_forget()
 exit_button.pack_forget()
 credits_button.pack_forget()
 view_all_data.pack_forget()
 go_back_button.pack(
 side=tk.LEFT, padx=10, pady=10
 ) # Adding spacing on the left side and at the top


def go_back_to_main_menu():
 center_window(root)
 title_label.config(text="Welcome", font=("Montserrat", 16,
"bold"))
 label.config(
 text="Please choose an option below to initiate the desired
action.",
 font=("Montserrat", 12),
 )
 go_back_button.pack_forget()
 option1.pack_forget()
 option2.pack_forget()
 option3.pack_forget()
 option4.pack_forget()
 # credits_button.pack_forget()
 return_to_main_menu.pack_forget()
 enter_button.pack(side=tk.LEFT, padx=10)
 exit_button.pack(side=tk.RIGHT, padx=10)
 view_all_data.pack(side=tk.LEFT, padx=10)
 credits_button.pack(
 side=tk.LEFT, padx=10
 ) # Change this to left to align with other buttons

def show_head_tail():
 center_window(root)
 num_rows_head = askinteger(
 "Input", "Enter the number of rows to select from top:",
parent=root
 )
 num_rows_tail = askinteger(
 "Input", "Enter the number of rows to select from bottom:",
parent=root
 )

 if (
 num_rows_head is not None
 and num_rows_head > 0
 and num_rows_tail is not None
 and num_rows_tail > 0
 ):
 data = pd.read_csv("datasets/colleges.csv")
 top_rows = data.head(num_rows_head)
 bottom_rows = data.tail(num_rows_tail)

 # Trim columns that are longer than 300 characters
 columns_to_trim = ["Courses", "Facilities"]
 for col in columns_to_trim:
 top_rows.loc[:, col] = top_rows[col].apply(
 lambda x: x[:300] if isinstance(x, str) and len(x) > 300 else x
 )
 bottom_rows.loc[:, col] = bottom_rows[col].apply(
 lambda x: x[:300] if isinstance(x, str) and len(x) > 300 else x
 )
 def update_display():
 selected_col = selected_column.get()

 top_text = top_rows[selected_col].to_string(index=True)
 bottom_text = bottom_rows[selected_col].to_string(index=True)

 result_text.config(state=tk.NORMAL) # Enable the Text widget
 result_text.delete(1.0, tk.END)
 result_text.insert(
 tk.END,
 f"Top {num_rows_head} Rows:\n{top_text}\n\nBottom
{num_rows_tail} Rows:\n{bottom_text}",
 )
 result_text.config(state=tk.DISABLED) # Disable the Text widget
again

 result_window = tk.Toplevel(root)
 result_window.title("Head and Tail")
 result_window.geometry("800x600")
 center_window(root)

 # Create a dropdown menu for column selection
 selected_column = tk.StringVar()
selected_column.set(
 top_rows.columns[0]
 ) # Set default value to the first column
 column_dropdown = ttk.Combobox(
 result_window,
 textvariable=selected_column,
 values=top_rows.columns.tolist(),
 )
 column_dropdown.pack(pady=10)

 result_text = tk.Text(result_window, wrap=tk.WORD, font=
("Montserrat", 12))
 result_text.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
```

```
def on_exit():
 root.destroy()

animation_running = True
def on_enter_button_click():
 center_window(root)

 def update_text_animation(dot_count):
 global animation_running
 if animation_running:
 if dot_count == 0:
 label.config(
 text="Data Insertion in Progress, Kindly await completion.",
 font=("Montserrat", 12),
 )
 # Disable the buttons when the animation starts
 enter_button.config(state=tk.DISABLED)
 exit_button.config(state=tk.DISABLED)
 else:
 current_text = label.cget("text")
 label.config(text=current_text + ".", font=("Montserrat", 12))
 dot_count = (dot_count + 1) % 4
 root.after(1000, update_text_animation, dot_count)
 else:
 # Enable the buttons after the text is updated
 enter_button.config(state=tk.NORMAL)
 exit_button.config(state=tk.NORMAL)
 def finish_inserting_data():
 global animation_running
 animation_running = False
 title_label.config(text="Database Initialization Complete")
 label.config(
 text="Please utilize the provided buttons to commence the
process.",
 font=("Montserrat", 12),
 )
 # Enable the buttons after the text is updated
 view_all_data.pack_forget()
 enter_button.config(state=tk.NORMAL, command=search_dataset)
 exit_button.config(state=tk.NORMAL)

 def initialize_and_insert_data():
 connection = mydb.cursor()
 Initialize.db(connection)
 Insert.from_csv(connection, "./datasets/movies.csv")
 finish_inserting_data()

 # Start the animation thread
 animation_thread =
threading.Thread(target=update_text_animation, args=(0,))
 animation_thread.start()

 # Start the initialization thread
 initialize_thread =
threading.Thread(target=initialize_and_insert_data)
 initialize_thread.start()

 # Remove the "Credits" button when the "Enter" button is
clicked
 credits_button.pack_forget()
def show_csv_data():
 center_window(root)
 csv_file_path = "datasets/colleges.csv" # Update with your
actual CSV file path
 page_size = 50 # Number of rows per page

 def load_data(page_num):
 center_window(root)
 start_idx = page_num * page_size
 end_idx = start_idx + page_size
 data = pd.read_csv(csv_file_path)
 page_data = data.iloc[start_idx:end_idx]
 return page_data
 def show_page(page_num):
 center_window(root)
 nonlocal current_page # Use nonlocal to update the outer
current_page variable
 current_page = page_num # Update the current_page variable

 tree.delete(*tree.get_children()) # Clear existing data in the
Treeview
 page_data = load_data(page_num)

 for index, row in page_data.iterrows():
 tree.insert("", "end", values=row.tolist())

 result_window = tk.Toplevel(root)
 result_window.title("CSV Data")
 result_window.geometry("1000x800")
 center_window(root)
 data = pd.read_csv(csv_file_path) # Load the data here
 column_display_names = [
 "College Name",
 "Genders Accepted",
 "Campus Size",
 "Total Student Enrollments",
 "Total Faculty",
 "Established Year",
 "Rating",
 "University",
 "Courses",
 "Facilities",
 "City",
 "State",
 "Country",
 "College Type",
 "Average Fees",
 ]
 # tree = ttk.Treeview(result_window,
columns=column_display_names, show="headings")
 # tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

 style = ttk.Style()
 style.configure(
 "Treeview.Heading", font=("Helvetica", 12)
 ) # Change font and size as needed

 # Create the Treeview widget with the custom style
 tree = ttk.Treeview(
 result_window,
 columns=column_display_names,
 show="headings",
 style="Custom.Treeview",
 )
 tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

 for col in column_display_names:
 tree.heading(col, text=col) # Set heading text to column name
 tree.column(col, width=100) # Adjust column width as needed

 prev_page_button = ttk.Button(
 result_window,
 text="Previous Page",
 style="PrevPage.TButton",
 command=lambda: show_page(current_page - 1),
 )
 prev_page_button.pack(side=tk.LEFT, padx=10)

 prev_page_button_style = ttk.Style()
 prev_page_button_style.configure(
 "PrevPage.TButton", font=("Helvetica", 15)
 ) # Change font and size as needed
 next_page_button = ttk.Button(
 result_window,
 text="Next Page",
 style="NextPage.TButton",
 command=lambda: show_page(current_page + 1),
 )
 next_page_button.pack(side=tk.RIGHT, padx=10)

 prev_page_button_style = ttk.Style()
 prev_page_button_style.configure(
 "NextPage.TButton", font=("Helvetica", 15)
 ) # Change font and size as needed

 current_page = 0
 show_page(current_page)


def search_and_compare():
 search_query = search_entry.get().strip()
 if search_query:
 # Find the most related college name
 related_college = data[
 data["College Name"].str.contains(search_query, case=False,
na=False)
 ].iloc[0]
 college_name = related_college["College Name"]
 compare_colleges(college_name, selected_college)
 else:
 messagebox.showinfo("Search Error", "Please enter a search
query.")

def search_dataset():
 center_window(root)
 title_label.config(
 text="Please make a selection to proceed.", font=("Montserrat",
16, "bold"))
```

```
 option1.pack(side=tk.LEFT, padx=10)
 option2.pack(side=tk.LEFT, padx=10)
 option3.pack(side=tk.LEFT, padx=10)
 option4.pack(side=tk.LEFT, padx=10)
 credits_button.pack_forget()
 view_all_data.pack_forget()
 return_to_main_menu.pack(side=tk.LEFT, padx=10)
 exit_button.pack_forget()


def compare_colleges(college1, college2):
 try:
 # Load the data
 data = pd.read_csv("datasets/colleges.csv")

 # Filter data for the entered college names
 college1_data = data[data["College
Name"].str.contains(college1, case=False)]
 college2_data = data[data["College
Name"].str.contains(college2, case=False)]

 # Check if data is found for both colleges
 if college1_data.empty or college2_data.empty:
 messagebox.showinfo(
 "Error", "One or both college names not found in the data."
 )
 return

 # Columns to compare
 columns_to_compare = [
 "Campus Size",
 "Average Fees",
 "Total Student Enrollments",
 ]
 # Convert Campus Size values to int after handling different
cases
 college1_data.loc[:, "Campus Size"] = college1_data["Campus
Size"].apply(
 parse_campus_size
 )
 college2_data.loc[:, "Campus Size"] = college2_data["Campus
Size"].apply(
 parse_campus_size
 )
 # Handle NaN values by replacing them with 0
 college1_data[columns_to_compare] =
college1_data[columns_to_compare].fillna(0)
 college2_data[columns_to_compare] =
college2_data[columns_to_compare].fillna(0)
 # Convert values to float for comparison
 values_college1 =
college1_data[columns_to_compare].values[0].astype(float)
 values_college2 =
college2_data[columns_to_compare].values[0].astype(float)
 # Compare the length of courses
 courses_college1 = college1_data["Courses"].str.split(",
").dropna().iloc[0]
 courses_college2 = college2_data["Courses"].str.split(",
").dropna().iloc[0]
 courses_length_college1 = len(courses_college1)
 courses_length_college2 = len(courses_college2)

 courses_comparison = {
 college1: courses_length_college1,
 college2: courses_length_college2,
 }

 # Create bar graphs for comparison with custom colors
 plt.figure(figsize=(10, 5))
```

```
# Specify custom colors for the bars
colors = ["#050100", "#9000ff"]

plt.subplot(2, 2, 1)
plt.bar(
list(courses_comparison.keys()),
list(courses_comparison.values()),
color=colors,
)
plt.xlabel("Colleges")
plt.ylabel("Number of Courses")
plt.title("Number of Courses Comparison")
plt.xticks(rotation=45, ha="right")
# plt.annotate(
# college1,
# (0, courses_length_college1),
# textcoords="offset points",
# xytext=(0, 10),
# ha="center",
# )
# plt.annotate(
# college2,
# (1, courses_length_college2),
# textcoords="offset points",
# xytext=(0, 10),
# ha="center",
# )

plt.subplot(2, 2, 2)
plt.bar(
[college1, college2],
[values_college1[0], values_college2[0]],
color=["#050100", "#9000ff"],
)
```

```
 plt.xlabel("Colleges")
 plt.ylabel("Campus Size")
 plt.title("Campus Size Comparison")
 plt.xticks(rotation=45, ha="right")

 plt.subplot(2, 2, 3)
 avg_fees_college1 = college1_data["Average Fees"].values[0]
 avg_fees_college2 = college2_data["Average Fees"].values[0]

 plt.bar(
 [college1, college2], [avg_fees_college1, avg_fees_college2],
color=colors
 )
 plt.xlabel("Colleges")
 plt.ylabel("Average Fees")
 plt.title("Average Fees Comparison")
 plt.xticks(rotation=45, ha="right")

 plt.subplot(2, 2, 4)
 enrollments_college1 = college1_data["Total Student
Enrollments"].values[0]
 enrollments_college2 = college2_data["Total Student
Enrollments"].values[0]

 plt.bar(
 [college1, college2],
 [enrollments_college1, enrollments_college2],
 color=colors,
 )
 plt.xlabel("Colleges")
 plt.ylabel("Total Student Enrollments")
 plt.title("Total Student Enrollments Comparison")
 plt.xticks(rotation=45, ha="right")

 plt.tight_layout()
 plt.show()
 except Exception as e:
 messagebox.showinfo("Error", f"An error occurred: {e}")
 print(e)


def parse_campus_size(value):
 try:
 # Handle cases where value is not in the expected format
 if isinstance(value, str) and "Acre" in value:
 return float(value.replace(" Acres", ""))
 elif isinstance(value, str) and value.isdigit():
 return float(value)
 else:
 return 0.0
 except:
 return 0.0


def export_csv_data():
 global data # Make sure data is a global variable accessible in
this function
 data = pd.read_csv(
 "datasets/colleges.csv"
 ) # Load your CSV data here

 def load_data(page_num):
 start_idx = page_num * page_size
 end_idx = start_idx + page_size

 return data.iloc[start_idx:end_idx]

 def show_page(page_num):
 nonlocal current_page # Use nonlocal to update the outer
current_page variable
 current_page = page_num
 tree.delete(*tree.get_children()) # Clear existing data in the
Treeview
 page_data = load_data(page_num)

 for index, row in page_data.iterrows():
 tree.insert("", "end", values=row.tolist())

 def prev_page():
 if current_page > 0:
 show_page(current_page - 1)

 def next_page():
 last_page = (len(data) - 1) // page_size
 if current_page < last_page:
 show_page(current_page + 1)

 def save_changes():
 updated_data = []
 for item in tree.get_children():
 values = tree.item(item, "values")
 updated_data.append(values)

 # Convert the updated data back to a DataFrame
 updated_df = pd.DataFrame(updated_data, columns=data.columns)

 # Define supported file extensions
 supported_extensions = ["csv", "xlsx", "json", "html"]

 # Create a Combobox for selecting the file extension
 file_extension_combo = Combobox(
 root, values=supported_extensions, state="readonly"
 )
 file_extension_combo.set("csv")
 file_extension_combo.pack()
 def save_with_extension():
 user_extension = file_extension_combo.get().lower()

 if user_extension not in supported_extensions:
 messagebox.showerror("Invalid Extension", "Unsupported file
extension.")
 return
 # Prompt the user to select a file location to save the updated
data
 file_path = filedialog.asksaveasfilename(
 defaultextension=f".{user_extension}",
 filetypes=[
 ("CSV Files", "*.csv"),
 ("Excel Files", "*.xlsx"),
 ("JSON Files", "*.json"),
 ("HTML Files", "*.html"),
 ],
 )
 if file_path:
 if user_extension == "csv":
 updated_df.to_csv(file_path, index=False)
 elif user_extension == "xlsx":
 updated_df.to_excel(file_path, index=False)
 elif user_extension == "json":
 updated_df.to_json(file_path, orient="records")
 elif user_extension == "html":
 updated_df.to_html(file_path, index=False)

 messagebox.showinfo(
 "Save Successful", f"Changes saved to '{file_path}'."
 )
 # # Create a button to trigger the save process
 # save_button = tk.Button(root, text="Save Changes",
command=save_with_extension)
 # save_button.pack()
 edit_window = tk.Toplevel(root)
 edit_window.title("Edit CSV Data")
 edit_window.geometry("1000x800")
 center_window(edit_window)

 page_size = 10 # Number of rows per page
 current_page = 0

 column_display_names = data.columns

 # Create the Treeview widget with the custom style
 tree = ttk.Treeview(
 edit_window,
 columns=column_display_names,
 show="headings",
 style="Custom.Treeview",
 )
 tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

 for col_index, col in enumerate(column_display_names):
 tree.heading(col_index, text=col) # Set the heading text using
the index
 tree.column(col_index, width=100) # Adjust column width as
needed

 entry_widgets = {} # Store entry widgets for each row
 for index, row in load_data(current_page).iterrows():
 item = tree.insert("", "end", values=row.tolist())
 entry_widgets[item] = []

 for col_index, col_value in enumerate(row):
 entry = tk.Entry(tree, justify="center", font=("Helvetica",
12))
 entry.insert(0, col_value)
 tree.window_create(item, col_index, window=entry)
 entry_widgets[item].append(entry)
 prev_page_button = ttk.Button(
 edit_window, text="Previous Page", command=prev_page
 )
 prev_page_button.pack(side=tk.LEFT, padx=10)

 next_page_button = ttk.Button(edit_window, text="Next Page",
command=next_page)
 next_page_button.pack(side=tk.RIGHT, padx=10)

 save_button = ttk.Button(
 edit_window,
 text="Save this data to a CSV",
 command=save_changes,
 style="Montserrat.TButton",
 )
 save_button.pack(pady=10)

 # Define a custom style for the buttons to use Montserrat font
 style = ttk.Style()
 style.configure(
 "Custom.Treeview", font=("Helvetica", 12)
 ) # Change font and size as needed
 style.configure("Montserrat.TButton", font=("Montserrat", 12))

 def finish_editing(event, item, row, col):
 new_value = entry_var.get()
 tree.item(item, values=("", new_value, ""))
 entry.place_forget()
 entry.unbind("<FocusOut>")

 def save_changes():
 updated_data = []
 for item in tree.get_children():
 values = tree.item(item, "values")
 updated_data.append(values)
 # Convert the updated data back to a DataFrame
 updated_df = pd.DataFrame(updated_data, columns=data.columns)

 # Define supported file extensions
 supported_extensions = ["csv", "xlsx", "json", "html"]

 def save_with_extension():
 user_extension = file_extension_combo.get().lower()

 if user_extension not in supported_extensions:
 messagebox.showerror("Invalid Extension", "Unsupported file
extension.")
 return

 # Prompt the user to select a file location to save the updated
data
 file_path = filedialog.asksaveasfilename(
 defaultextension=f".{user_extension}",
 )

 if file_path:
 if user_extension == "csv":
 updated_df.to_csv(file_path, index=False)
 elif user_extension == "xlsx":
 updated_df.to_excel(file_path, index=False)
 elif user_extension == "json":
 updated_df.to_json(file_path, orient="records")
 elif user_extension == "html":
 updated_df.to_html(file_path, index=False)

 messagebox.showinfo(
 "Save Successful", f"Changes saved to '{file_path}'."
 )
 extension_button.pack_forget()
 def open_extension_selector():
 extension_window = tk.Toplevel(root)
 extension_window.title("Select File Extension")
 # Create a Combobox for selecting the file extension
 file_extension_combo = ttk.Combobox(
 extension_window, values=supported_extensions, state="readonly"
 )
 file_extension_combo.set("csv")
 file_extension_combo.pack(padx=10, pady=10)

 # Create a button to trigger the save process with the selected
extension
 save_button = tk.Button(
 extension_window, text="Save Changes",
command=save_with_extension
 )
 save_button.pack(pady=10)
 extension_button.pack_forget()

 # Create a button to open the extension selector window
 extension_button = tk.Button(
 root, text="Select File Extension",
command=open_extension_selector
 )
 extension_button.pack(pady=10)
 extension_button.pack_forget()
 # Create a new Toplevel window for the file extension selection
 extension_window = tk.Toplevel(edit_window)
 extension_window.title("Select an extension")
 extension_window.geometry("600x300")
 center_window(extension_window)

 # Create a Combobox for selecting the file extension
 file_extension_combo = ttk.Combobox(
 extension_window, values=supported_extensions, state="readonly"
 )
```

```
file_extension_combo.set("csv")
 file_extension_combo.pack(padx=10, pady=10)

 # Create a button to trigger the save process with the selected
extension
 save_button = tk.Button(
 extension_window, text="Save Changes",
command=save_with_extension
 )
 save_button.pack(pady=10)

 # Create a button to open the extension selector window
 extension_button = tk.Button(
 root, text="Select File Extension",
command=open_extension_selector
 )
 extension_button.pack(pady=10)
 extension_button.pack_forget()

 edit_window = tk.Toplevel(root)
 edit_window.title("Edit CSV Data")
 edit_window.geometry("1000x800")
 center_window(edit_window)

 page_size = 50 # Number of rows per page
 current_page = 0
 column_display_names = data.columns

 # Create the Treeview widget with the custom style
 tree = ttk.Treeview(
 edit_window,
 columns=column_display_names,
 show="headings",
 style="Custom.Treeview",
 )
 tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
 tree.heading("#1", text="Column A")
 tree.heading("#2", text="Column B")
 tree.heading("#3", text="Column C")

 tree.insert("", "end", values=("Value 1", "Value 2", "Value
3"))

 entry_var = tk.StringVar()
 entry = tk.Entry(root, textvariable=entry_var)

 entry_var = tk.StringVar()
 entry = tk.Entry(edit_window, textvariable=entry_var)

 # Set the heading text for columns based on
column_display_names
 for col_index, col in enumerate(column_display_names):
 tree.heading(col_index, text=col) # Set the heading text using
the index
 tree.column(col_index, width=100) # Adjust column width as
needed

 prev_page_button = ttk.Button(edit_window, text="Previous
Page", command=prev_page)
 prev_page_button.pack(side=tk.LEFT, padx=10)

 next_page_button = ttk.Button(edit_window, text="Next Page",
command=next_page)
 next_page_button.pack(side=tk.RIGHT, padx=10)

 save_button = ttk.Button(
 edit_window,
 text="Save this page in a custom extension",
 command=save_changes,
 style="Montserrat.TButton",
 )
 save_button.pack(pady=10)
 # Define a custom style for the buttons to use Montserrat font
 style = ttk.Style()
 style.configure(
 "Custom.Treeview", font=("Helvetica", 12)
 ) # Change font and size as needed
 style.configure("Montserrat.TButton", font=("Montserrat", 12))

 # Initially show the first page
 show_page(current_page)


def compare_colleges_window():
 def compare():
 college1 = college1_var.get().strip()
 college2 = college2_var.get().strip()

 if college1 and college2:
 compare_colleges(college1, college2)
 else:
 messagebox.showinfo("Input Error", "Please enter both college
names.")

 compare_window = tk.Toplevel(root)
 compare_window.title("Compare Colleges")
 compare_window.geometry("400x250")
 center_window(compare_window)

 college1_label = tk.Label(compare_window, text="Enter College
1:")
 college1_label.pack(pady=10)
 college1_var = tk.StringVar()
 college1_entry = tk.Entry(compare_window,
textvariable=college1_var)
 college1_entry.pack()
 college2_label = tk.Label(compare_window, text="Enter College
2:")
 college2_label.pack(pady=10)
 college2_var = tk.StringVar()
 college2_entry = tk.Entry(compare_window,
textvariable=college2_var)
 college2_entry.pack()

 compare_button = tk.Button(compare_window, text="Compare",
command=compare)
 compare_button.pack(pady=10)


def search_college():
 college_name = simpledialog.askstring("Search College", "Enter
College Name:")
 if college_name:
 # Use a case-insensitive search for college names
 data = pd.read_csv("datasets/colleges.csv")
 relevant_data = data[
 data["College Name"].str.contains(college_name, case=False)
 ]
 if not relevant_data.empty:
 display_results_window(relevant_data.iloc[0])
 else:
 messagebox.showinfo("No Results", "No matching data found.")


def display_results_window(result):
 results_window = tk.Toplevel()
 results_window.title("Search Results")
 results_window.geometry("1100x900")
 center_window(results_window)
 text_widget = tk.Text(results_window, font=("Montserrat", 14))
 text_widget.pack(fill="both", expand=True)
 text_widget.tag_configure("bold", font=("Montserrat", 18,
"bold"))

 text_widget.insert("end", f"College Name: {result['College
Name']}\n", "bold")
 text_widget.insert("end", f"Genders Accepted: {result['Genders
Accepted']}\n")
 text_widget.insert("end", f"Campus Size: {result['Campus
Size']}\n")
 text_widget.insert(
 "end", f"Total Student Enrollments: {result['Total Student
Enrollments']}\n"
 )
 text_widget.insert("end", f"Total Faculty: {result['Total
Faculty']}\n")
 text_widget.insert("end", f"Established Year:
{result['Established Year']}\n")
 text_widget.insert("end", f"Rating: {result['Rating']}\n")
 text_widget.insert("end", f"University:
{result['University']}\n")

 # Append courses with "-"
 courses = result["Courses"].split(",")
 formatted_courses = "\n".join([f"- {course.strip()}" for course
in courses])
 text_widget.insert("end", f"Courses:
...\n{formatted_courses}\n")

 text_widget.insert("end", f"Facilities:
{result['Facilities']}\n")
 text_widget.insert("end", f"City: {result['City']}\n")
 text_widget.insert("end", f"State: {result['State']}\n")
 text_widget.insert("end", f"Country: {result['Country']}\n")
 text_widget.insert("end", f"College Type: {result['College
Type']}\n")
```

```
text_widget.insert("end", f"Average Fees: {result['Average
Fees']}\n")

 text_widget.configure(state="disabled")

def exit_application():
 root.destroy()

def navigate_to_number_options():
 title_label.config(text="Welcome")
 label.config(
 text="Option 1 - View top x and bottom y rows.\nOption 2 -
Export certain rows from dataset.\nOption 3 - Compare data
between 2 colleges.\nOption 4 - View a brief info about a
college.",
 )
 enter_button.pack(side=tk.LEFT, padx=10)
 credits_button.pack(side=tk.LEFT, padx=10)
 exit_button.pack(side=tk.LEFT, padx=10)
 view_all_data.pack(side=tk.LEFT, padx=10)

# Initialize the logger
logger = logger()

# Connect to the MySQL database using the provided configuration
# mydb = msc.connect(
# host=config.host,
# user=config.user,
# password=config.password,
# )

# Log information: Sending initial connection to the database,
awaiting response.
logger.log("INFO", "Sending initial connection to the database,
awaiting response.")
```

```
try:
 if True:
 # Open tkinter box after database operations are completed
 root = tk.Tk()
 root.title(config.projectName)
 root.geometry("700x500")
 center_window(root)
 root.config(bg=config.accentColor)

 title_label = tk.Label(
 root,
 text="Welcome",
 fg=config.fontColor,
 bg=config.accentColor,
 font=("Montserrat", 16, "bold"),
 )
 title_label.pack(pady=20)

 label = tk.Label(
 root,
 text="Select a button to get started.",
 fg=config.fontColor,
 bg=config.accentColor,
 font=("Montserrat", 12),
 )
 label.pack(pady=10)

 button_frame = tk.Frame(root, bg=config.accentColor)
 button_frame.pack(pady=10)

 enter_button = Button(
 button_frame,
 text="Enter",
 command=on_enter_button_click,
 style="Montserrat.TButton",
 )
```

```
enter_button.pack(side=tk.LEFT, padx=10)

 exit_button = Button(
 button_frame, text="Exit", command=on_exit,
style="Montserrat.TButton"
 )
 exit_button.pack(side=tk.RIGHT, padx=10)

 credits_button = Button(
 button_frame,
 text="Credits",
 command=show_credits,
 style="Montserrat.TButton",
 )
 credits_button.pack(
 side=tk.LEFT, padx=10
 ) # Change this to left to align with other buttons

 go_back_button = Button(
 button_frame,
 text="Go back to main menu",
 command=go_back_to_main_menu,
 style="Montserrat.TButton",
 )
 go_back_button.pack_forget()

 view_all_data = Button(
 button_frame,
 text="View all data",
 command=show_csv_data,
 style="Montserrat.TButton",
 )
 view_all_data.pack(side=tk.LEFT, padx=10)
```

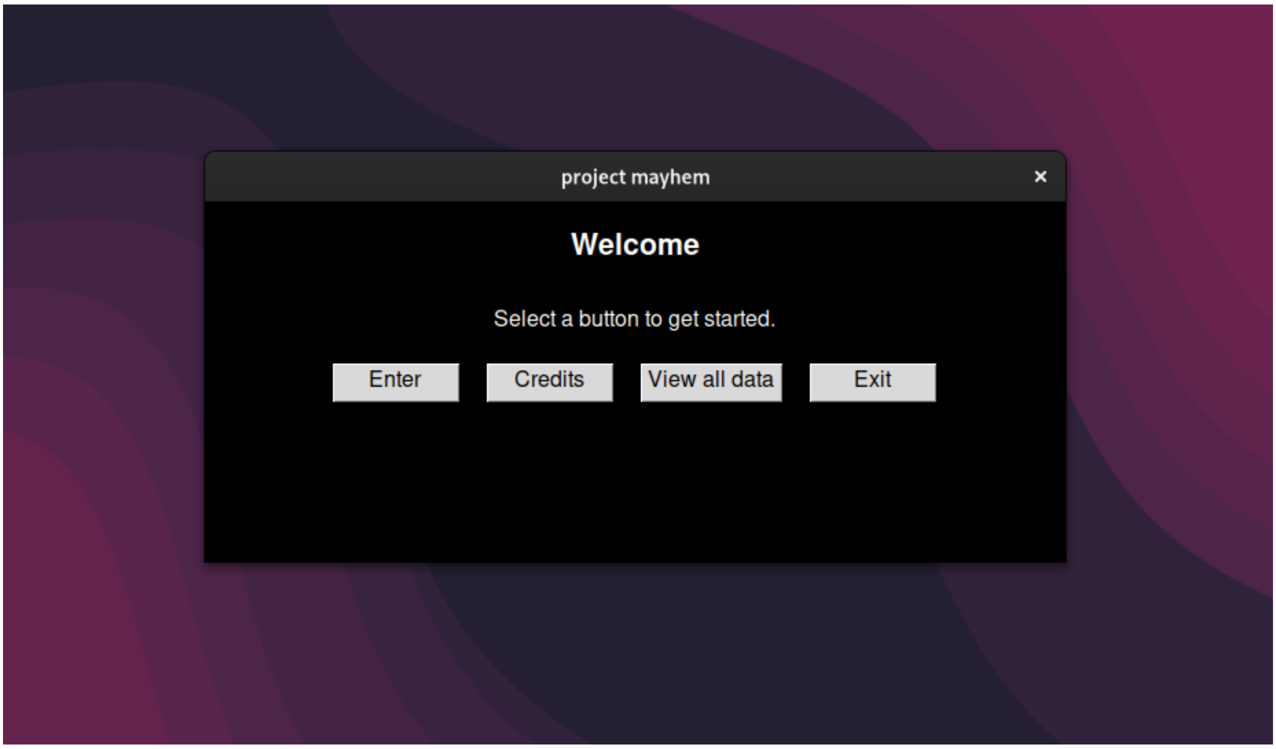
```
option1 = Button(
 button_frame, text="1", command=show_head_tail,
style="Montserrat.TButton"
 )
 option1.pack_forget()
 option2 = Button(
 button_frame, text="2", command=export_csv_data,
style="Montserrat.TButton"
 )
 option2.pack_forget()
 option3 = Button(
 button_frame,
 text="3",
 command=lambda: compare_colleges_window(),
 style="Montserrat.TButton",
 )
 option3.pack_forget()
 option4 = ttk.Button(
 button_frame,
 text="4",
 command=search_college,
 style="Montserrat.TButton",
 )
 option4.pack_forget()
 return_to_main_menu = Button(
 button_frame,
 text="Return to main menu",
 command=go_back_to_main_menu,
 style="Montserrat.TButton",
 )
 # Define a custom style for the buttons to use Montserrat font
 style = tk.ttk.Style()
 style.configure("Montserrat.TButton", font=("Montserrat", 12))

 root.mainloop()
 # Log success information: Database connected successfully. Use
the CLI or GUI to start querying.
 logger.log(
 "SUCCESS",
 "Database connected successfully, use the CLI or GUI to start
querying.",
 )

except Exception as e:
 # Log error: State the error name and print the error stack
 logger.log(
 "ERROR", "Client request was sent, but the server rejected the
handshake."
 )
 logger.log("ERROR", f"An error occurred: {e}")
```

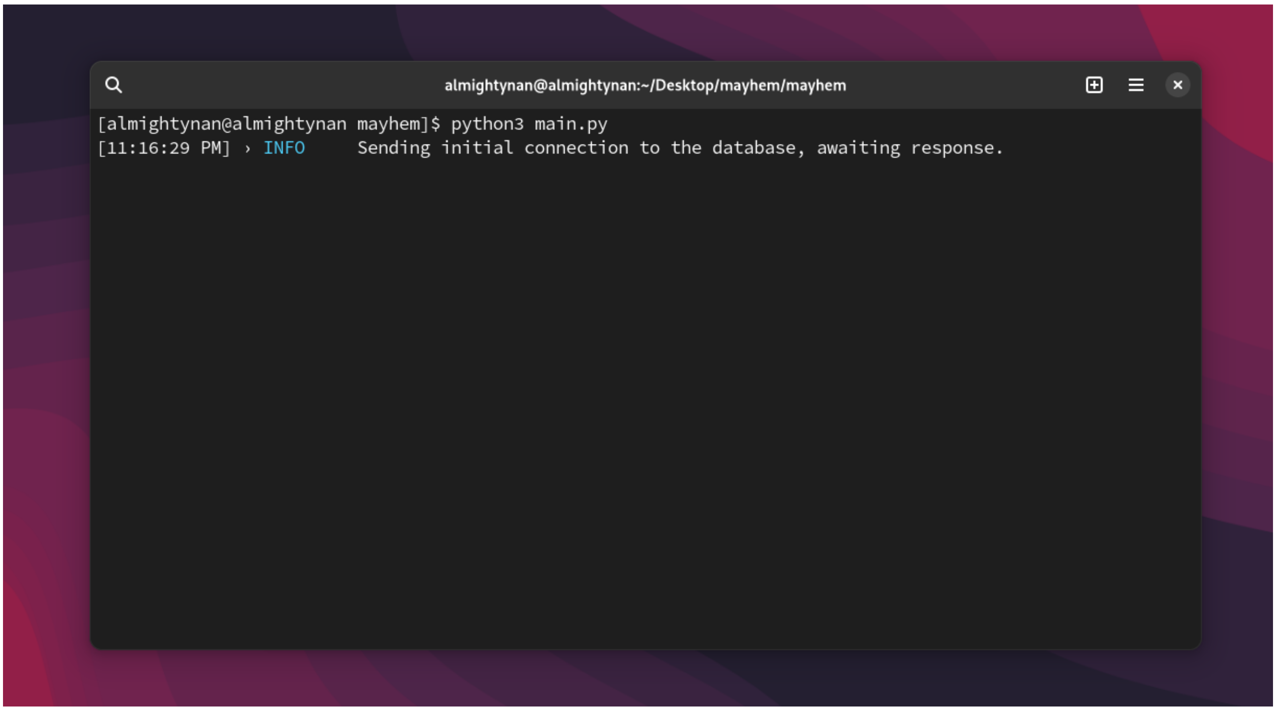
Source code

./requirements.txt

```
matplotlib
mysql-connector-python
numpy
pandas
prettytable
plotly
prettytable
python-dateutil
SQLAlchemy
```

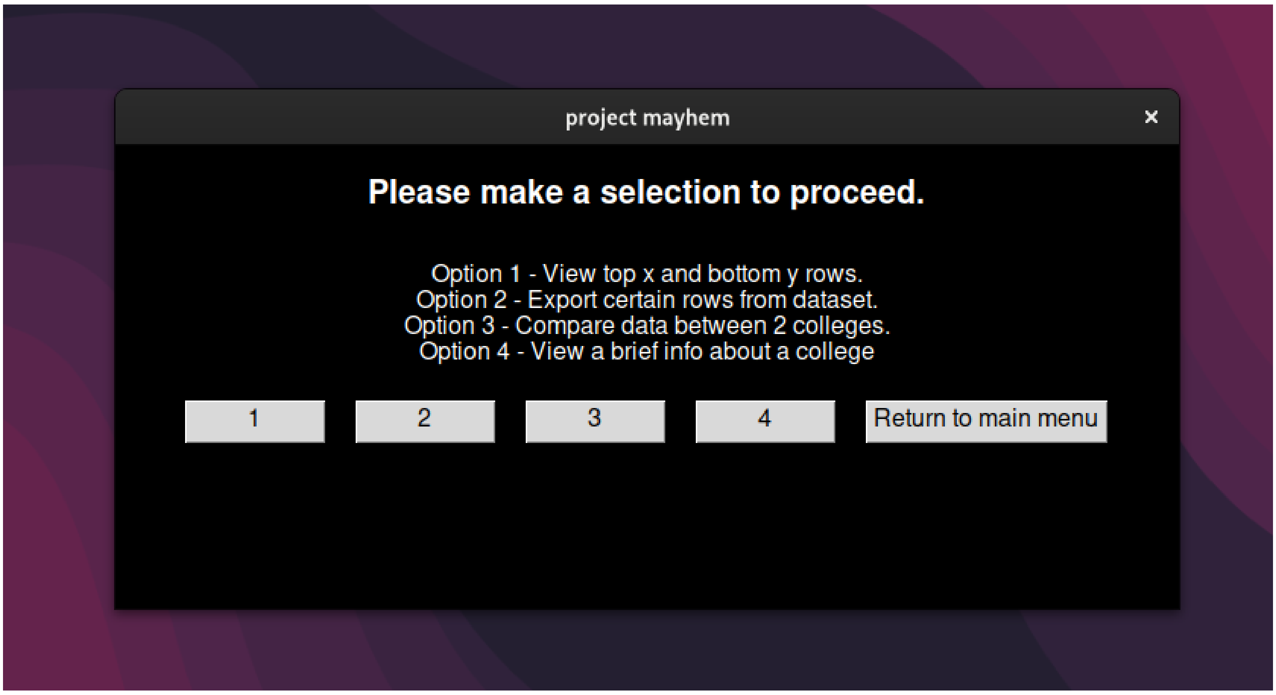
Output





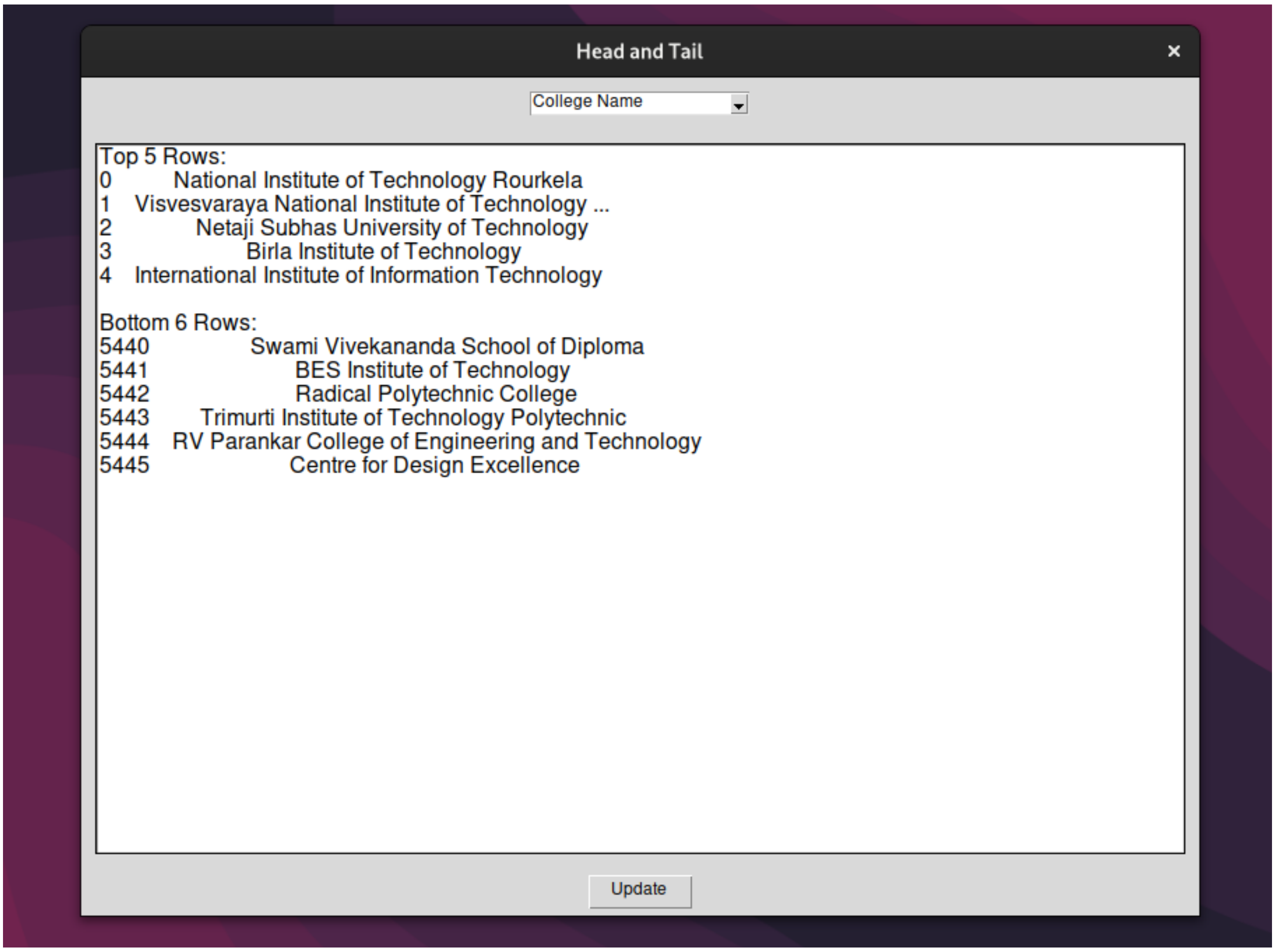
```
[almightynan@almightynan mayhem]$ python3 main.py
[11:16:29 PM] › INFO     Sending initial connection to the database, awaiting response.
```

Output



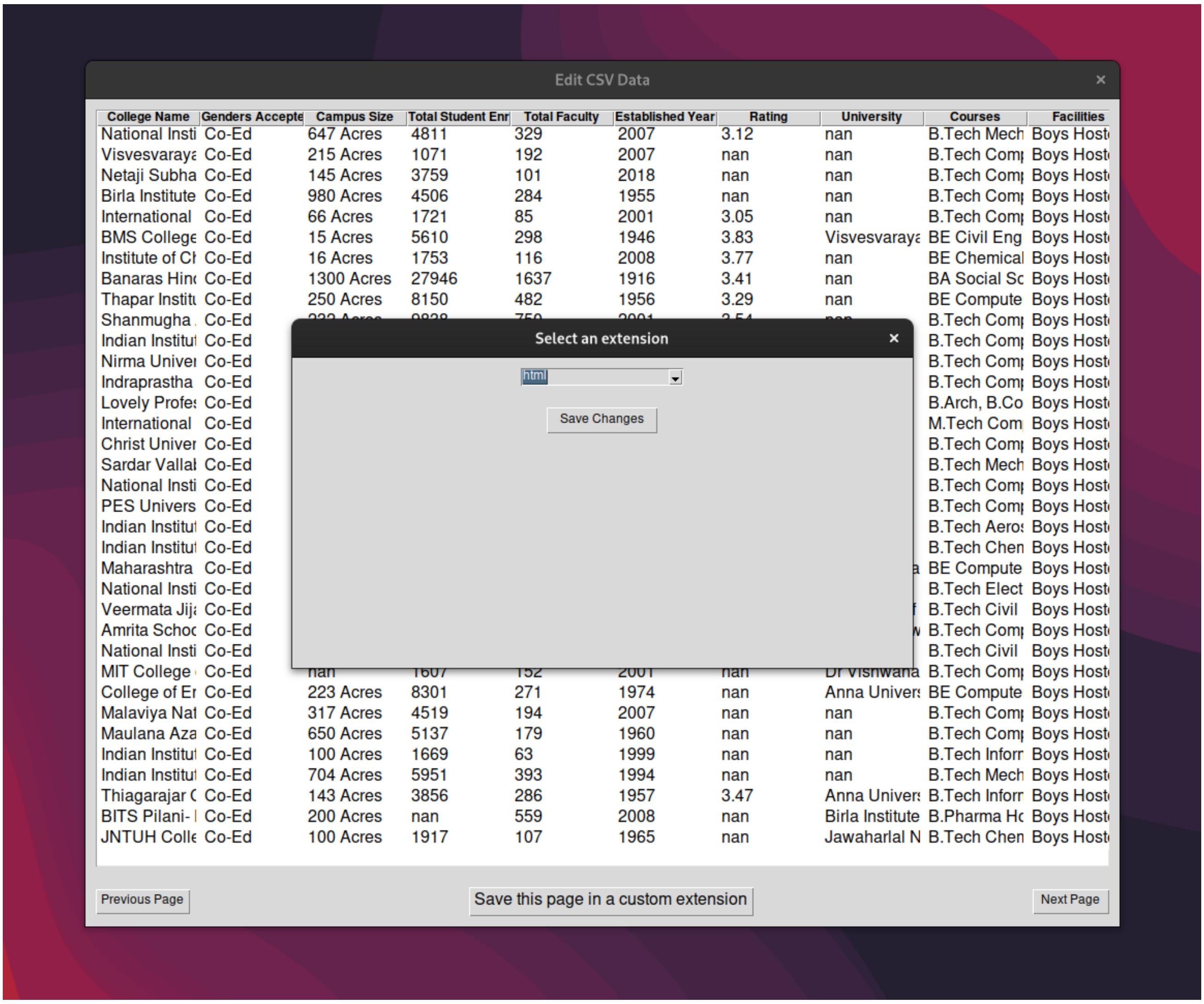
Please make a selection to proceed.

Option 1 - View top x and bottom y rows.
Option 2 - Export certain rows from dataset.
Option 3 - Compare data between 2 colleges.
Option 4 - View a brief info about a college



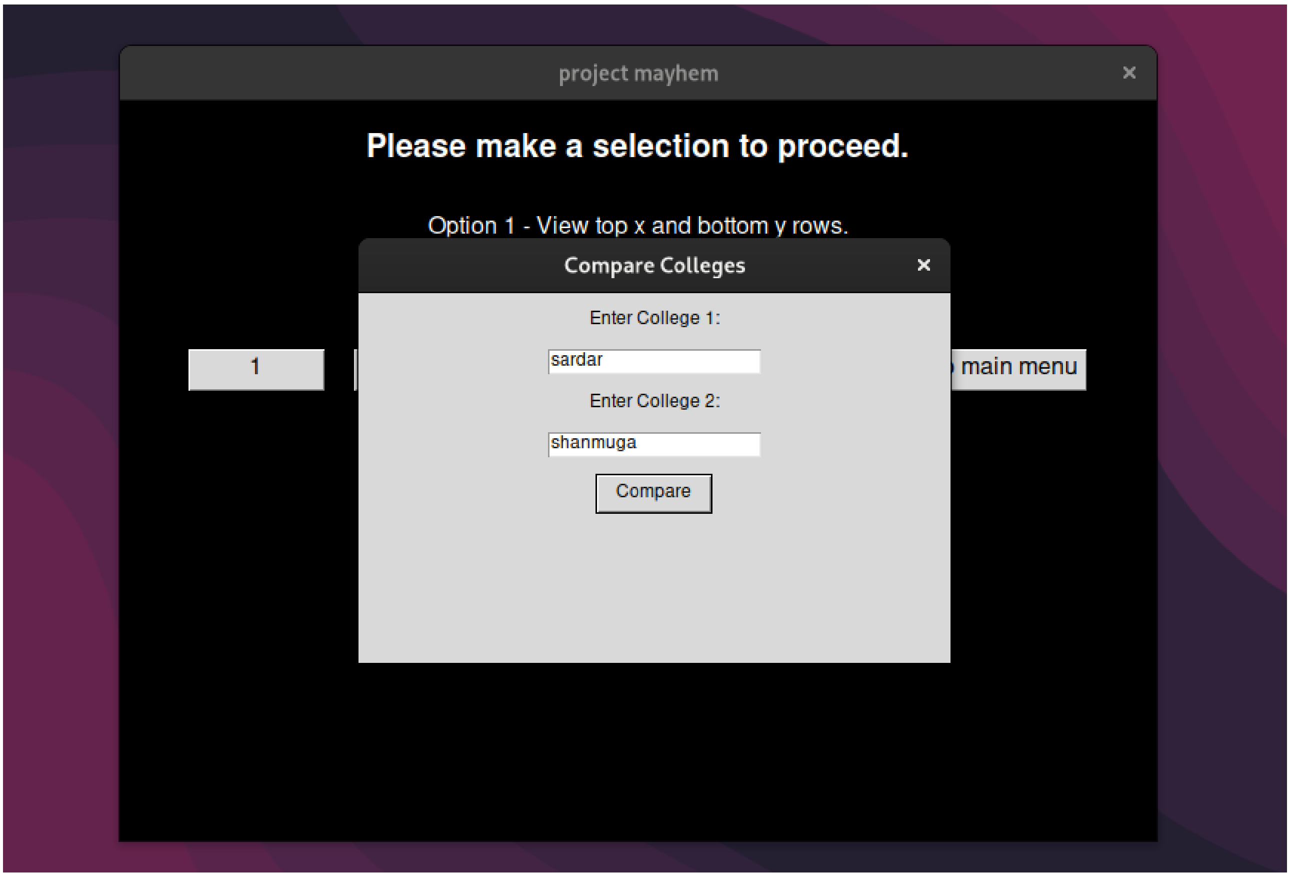
```
Top 5 Rows:
0        National Institute of Technology Rourkela
1    Visvesvaraya National Institute of Technology ...
2            Netaji Subhas University of Technology
3                  Birla Institute of Technology
4    International Institute of Information Technology

Bottom 6 Rows:
5440              Swami Vivekananda School of Diploma
5441                   BES Institute of Technology
5442                   Radical Polytechnic College
5443        Trimurti Institute of Technology Polytechnic
5444    RV Parankar College of Engineering and Technology
5445                  Centre for Design Excellence
```

Output



College Name	Genders Accepte	Campus Size	Total Student Enr	Total Faculty	Established Year	Rating	University	Courses	Facilities
National Insti	Co-Ed	647 Acres	4811	329	2007	3.12	nan	B.Tech Mech	Boys Host
Visvesvaraya	Co-Ed	215 Acres	1071	192	2007	nan	nan	B.Tech Comp	Boys Host
Netaji Subha	Co-Ed	145 Acres	3759	101	2018	nan	nan	B.Tech Comp	Boys Host
Birla Institute	Co-Ed	980 Acres	4506	284	1955	nan	nan	B.Tech Comp	Boys Host
International	Co-Ed	66 Acres	1721	85	2001	3.05	nan	B.Tech Comp	Boys Host
BMS College	Co-Ed	15 Acres	5610	298	1946	3.83	Visvesvaraya	BE Civil Eng	Boys Host
Institute of Ch	Co-Ed	16 Acres	1753	116	2008	3.77	nan	BE Chemical	Boys Host
Banaras Hinc	Co-Ed	1300 Acres	27946	1637	1916	3.41	nan	BA Social Sc	Boys Host
Thapar Institu	Co-Ed	250 Acres	8150	482	1956	3.29	nan	BE Compute	Boys Host
Shanmugha	Co-Ed	222 Acres	9828	750	2001	2.54	nan	B.Tech Comp	Boys Host
Indian Institut	Co-Ed							B.Tech Comp	Boys Host
Nirma Univer	Co-Ed							B.Tech Comp	Boys Host
Indraprastha	Co-Ed							B.Tech Comp	Boys Host
Lovely Profes	Co-Ed							B.Arch, B.Co	Boys Host
International	Co-Ed							M.Tech Com	Boys Host
Christ Univer	Co-Ed							B.Tech Comp	Boys Host
Sardar Vallal	Co-Ed							B.Tech Mech	Boys Host
National Insti	Co-Ed							B.Tech Comp	Boys Host
PES Univers	Co-Ed							B.Tech Comp	Boys Host
Indian Institut	Co-Ed							B.Tech Aeros	Boys Host
Indian Institut	Co-Ed							B.Tech Chen	Boys Host
Maharashtra	Co-Ed						a	BE Compute	Boys Host
National Insti	Co-Ed							B.Tech Elect	Boys Host
Veermata Jija	Co-Ed						f	B.Tech Civil	Boys Host
Amrita Schoc	Co-Ed						w	B.Tech Comp	Boys Host
National Insti	Co-Ed							B.Tech Civil	Boys Host
MIT College	Co-Ed	nan	1607	152	2001	nan	Dr Vishwana	B.Tech Comp	Boys Host
College of En	Co-Ed	223 Acres	8301	271	1974	nan	Anna Univers	BE Compute	Boys Host
Malaviya Nat	Co-Ed	317 Acres	4519	194	2007	nan	nan	B.Tech Comp	Boys Host
Maulana Aza	Co-Ed	650 Acres	5137	179	1960	nan	nan	B.Tech Comp	Boys Host
Indian Institut	Co-Ed	100 Acres	1669	63	1999	nan	nan	B.Tech Inforr	Boys Host
Indian Institut	Co-Ed	704 Acres	5951	393	1994	nan	nan	B.Tech Mech	Boys Host
Thiagarajar C	Co-Ed	143 Acres	3856	286	1957	3.47	Anna Univers	B.Tech Inforr	Boys Host
BITS Pilani- I	Co-Ed	200 Acres	nan	559	2008	nan	Birla Institute	B.Pharma Hc	Boys Host
JNTUH Colle	Co-Ed	100 Acres	1917	107	1965	nan	Jawaharlal N	B.Tech Chen	Boys Host

Output

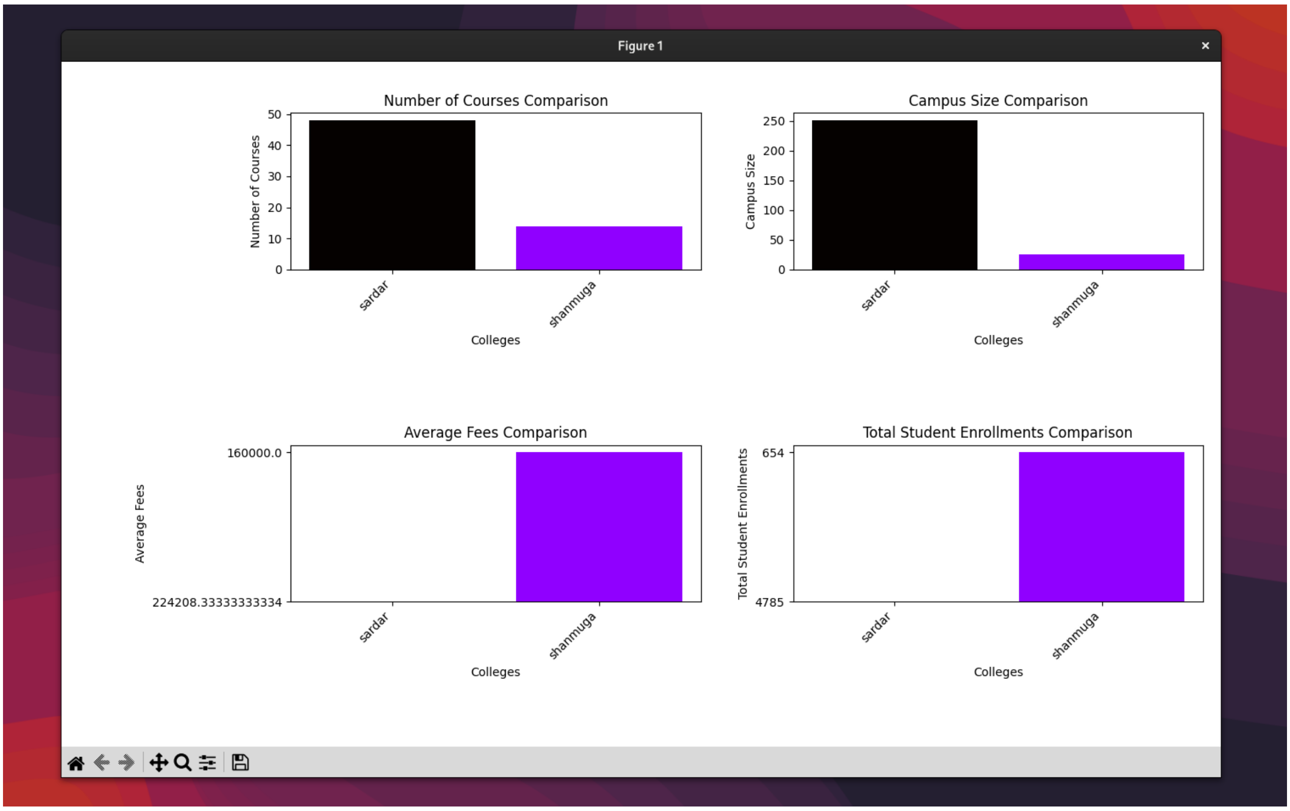


Please make a selection to proceed.

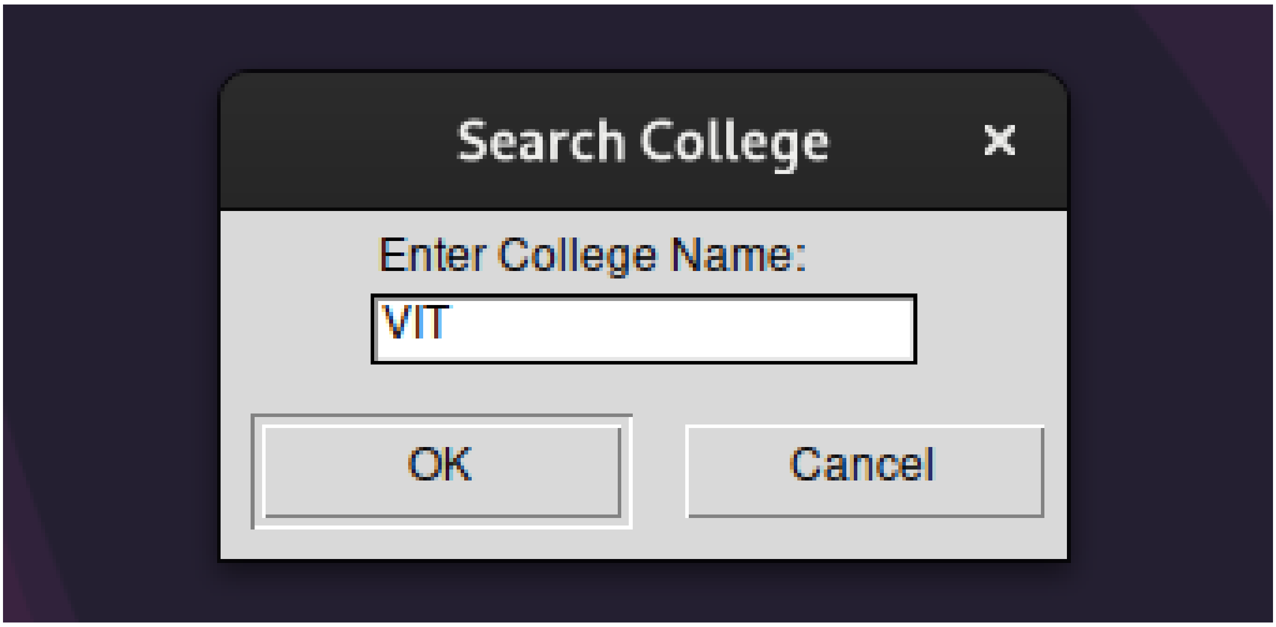
Option 1 - View top x and bottom y rows.

Enter College 1: sardar

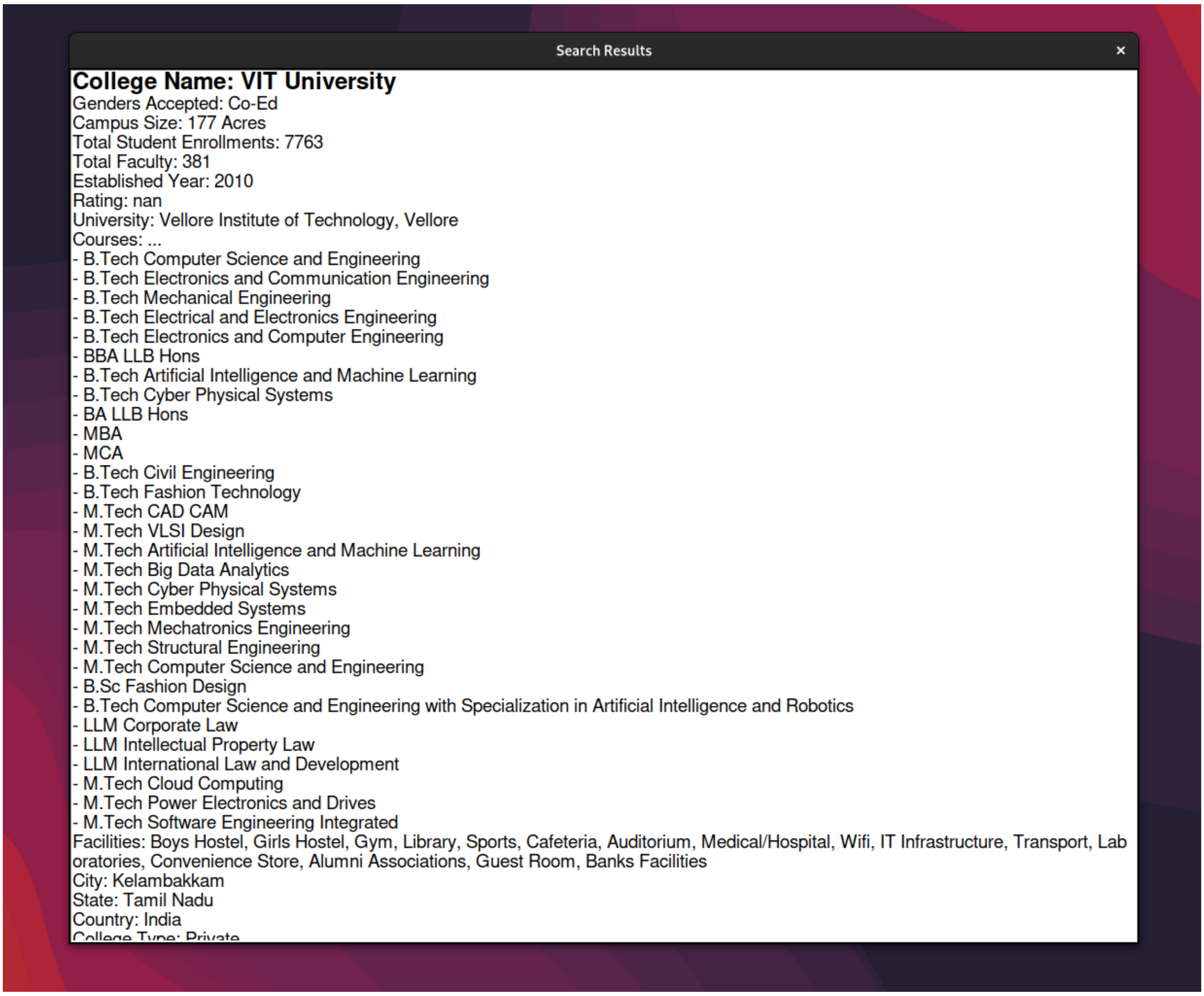
Enter College 2: shanmuga



Output



Enter College Name: VIT



```
College Name: VIT University
Genders Accepted: Co-Ed
Campus Size: 177 Acres
Total Student Enrollments: 7763
Total Faculty: 381
Established Year: 2010
Rating: nan
University: Vellore Institute of Technology, Vellore
Courses: ...
- B.Tech Computer Science and Engineering
- B.Tech Electronics and Communication Engineering
- B.Tech Mechanical Engineering
- B.Tech Electrical and Electronics Engineering
- B.Tech Electronics and Computer Engineering
- BBA LLB Hons
- B.Tech Artificial Intelligence and Machine Learning
- B.Tech Cyber Physical Systems
- BA LLB Hons
- MBA
- MCA
- B.Tech Civil Engineering
- B.Tech Fashion Technology
- M.Tech CAD CAM
- M.Tech VLSI Design
- M.Tech Artificial Intelligence and Machine Learning
- M.Tech Big Data Analytics
- M.Tech Cyber Physical Systems
- M.Tech Embedded Systems
- M.Tech Mechatronics Engineering
- M.Tech Structural Engineering
- M.Tech Computer Science and Engineering
- B.Sc Fashion Design
- B.Tech Computer Science and Engineering with Specialization in Artificial Intelligence and Robotics
- LLM Corporate Law
- LLM Intellectual Property Law
- LLM International Law and Development
- M.Tech Cloud Computing
- M.Tech Power Electronics and Drives
- M.Tech Software Engineering Integrated
Facilities: Boys Hostel, Girls Hostel, Gym, Library, Sports, Cafeteria, Auditorium, Medical/Hospital, Wifi, IT Infrastructure, Transport, Lab
oratories, Convenience Store, Alumni Associations, Guest Room, Banks Facilities
City: Kelambakkam
State: Tamil Nadu
Country: India
College Type: Private
```

Limitations

- The code isn't productive if scaled to handle larger datasets or more complex operations.
- Depending on the project's scale, tkinter might not be the most suitable GUI library for complex applications.
- The code assumes a certain level of compatibility with the user's environment, which could be a limitation on different platforms.
- Needs customization & functionality to fulfill the need of users.

References/Bibliography

- Python docs: https://docs.python.org/3/
- Tkinter docs: https://docs.python.org/3/library/tk.html
- pandas docs: https://pandas.pydata.org/docs/
- mysql docs: https://dev.mysql.com/doc/
- matplotlib docs: https://matplotlib.org/stable/api
- Text book: Class XII - Informatics Practices
- Guidance from teacher: Mrs. Madheena Banu