ASSIGNMENT NUMBER :1

SUBJECT NAME : DATA VISUALIZATION

SUBJECT CODE : 20-CST-461

SUBMITTED BY

NAME :AKHILESH KUMAR

UID: 20BCS9907

CLASS: 20BCS 23-B

SUBMITTED TO

Er HARMANPREET KAUR

EID: E15140

MINI PROJECT

TITLE : create a full apllication using python that visualizes and analyzes data distributions using various chart types.make the frontend and backend using python write the the full code

Objective:

Create an application that visualizes and analyzes data distributions through diverse chart types, providing users with a robust tool for gaining insights into the characteristics and patterns of their datasets.

Tasks: 1. Implement functionalities for importing and processing datasets.

2. Design a user interface allowing users to select and customize chart types for effective data distribution analysis.

ABSTARCT

This mini-project introduces a Python-based application designed for data analysis and visualization. The project encompasses both frontend and backend components, providing users with a comprehensive tool to import, process, and analyze datasets through a variety of chart types. The user-friendly interface, built using Flask, allows seamless dataset importation and customization of chart types, including scatter plots, bar charts, line charts, and pie charts. The dynamic backgrounds enhance the visual appeal of the interface, offering an engaging user experience. The backend ensures accurate result analysis and validation for the selected chart types. The documentation follows a structured format, covering problem identification, literature review, design flow, results analysis, and a user manual. With extensibility in mind, the project is poised for future enhancements and additions to further contribute to the evolving landscape of data analysis applications.

CHAPTER 1 INTRODUCTION

1.1 Identification of Client/ Need/ Relevant Contemporary Issue

In the realm of data analysis and visualization, the identified client represents professionals, researchers, and enthusiasts seeking a robust and user-friendly tool for exploring and interpreting datasets. The contemporary need for effective data analysis tools is evident as the volume and complexity of data continue to rise across various domains. Researchers, analysts, and decision-makers face challenges in deriving meaningful insights from datasets, necessitating a tool that streamlines the process. The relevance of this project lies in addressing the pressing need for an accessible and versatile data visualization application, empowering users to extract valuable information from their datasets efficiently.

1.2 Identification of Problem

The problem at hand revolves around the complexity and diversity of datasets, hindering individuals from effortlessly analyzing and interpreting data patterns. Traditional tools often lack user-friendliness and fail to provide a diverse range of visualization options. This project aims to bridge the gap by offering an intuitive solution that accommodates various data types and provides a selection of chart types for comprehensive data representation. The challenge lies in designing

an application that caters to the dynamic needs of users, ensuring ease of use and versatility in visualizing different datasets.

1.3. Identification of Tasks

The identified tasks involve developing a Python-based application that encompasses both frontend and backend components. Key tasks include designing an intuitive user interface for dataset importation, implementing functionalities for data analysis and visualization, and incorporating dynamic backgrounds for an engaging user experience. Furthermore, tasks involve ensuring extensibility for future enhancements and additions, contributing to the ongoing evolution of data analysis applications. Each task is intricately linked to the overarching goal of providing a seamless and comprehensive tool for users to explore and interpret datasets effectively.

2.5. Problem Definition

The problem at hand revolves around the limitations and inadequacies observed in existing data analysis tools, particularly in the realm of data visualization. A critical examination reveals that the available tools lack the necessary flexibility and adaptability required to handle the diverse and ever-expanding landscape of datasets effectively. Challenges include a restricted set of chart options, unintuitive user interfaces, and difficulties in accommodating different data structures seamlessly. Recognizing these issues is the foundation upon which this project is built. By identifying the problem scope, we aim to address these challenges comprehensively, thereby contributing to the evolution of data visualization tools.

2.6. Goals/Objectives

The overarching goal of this project is to develop a Python-based data visualization application that not only resolves existing challenges but also establishes a benchmark for future innovations. The specific objectives are carefully tailored to meet this overarching goal. They include the creation of an intuitive dashboard that serves as a user-friendly interface, the implementation of robust functionalities for importing and analyzing datasets, and the incorporation of dynamic backgrounds to enhance user engagement. By pursuing these objectives, the project aspires to set a new standard in the field of data visualization, offering a scalable and adaptable solution that can evolve with emerging data analysis requirements.

CHAPTER 2. DESIGN FLOW/PROCESS

This pivotal chapter delves into the meticulous design process that defined our project. Beginning with conceptualization, it traces the systematic flow through various phases, encapsulating the evolution from ideas to tangible components. The methodology employed is dissected, shedding light on strategic considerations, frameworks, and design principles that form the bedrock of the project's architecture. Notably, the backend development relies on Python, orchestrating the server-side operations with efficiency and versatility.

Within the design tapestry, this chapter provides methodological insights that propelled our project forward. It dissects each stage of the design flow, unraveling intricacies from ideation to iterative refinement. Backend operations are steered by Python, showcasing its robust capabilities in managing server-side functionalities. Design choices, supported by principles and user-centric considerations, are illuminated. The collaborative dynamics between interdisciplinary teams are explored, highlighting the synergy that shaped decisions. Challenges encountered are navigated through feedback loops and user testing, adding depth to our design narrative. This chapter stands as a beacon, not only documenting our design choices but also serving as a practical guide for

those navigating the realm of effective methodologies. It encapsulates the marriage of Python's backend prowess with frontend technologies like HTML, CSS, and JavaScript, orchestrating a seamless and engaging user experience.

CHAPTER 3. RESULTS ANALYSIS AND VALIDATION

3.1. Implementation of Solution

The implementation phase involves translating the planned design into a fully functional application. This encompasses coding the frontend and backend components, integrating datasets, and incorporating data visualization functionalities. The results of this phase are rigorously analyzed to ensure alignment with project goals and objectives. Testing and validation procedures are employed to guarantee the effectiveness of the solution in addressing identified challenges.

Here is the implementation of the project with the various components

Backend.py

```
# backend.py
from flask import Flask, render_template, request, redirect, url_for
from datetime import timedelta
import pandas as pd
import plotly.express as px

app = Flask(__name__)
app.config['SERVER_NAME'] = '127.0.0.1:5000'  # Add this line
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(minutes=10)  # Add this line

data = None
```

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload():
    global data
    file = request.files['file']

    if file.filename.endswith('.csv'):
        data = pd.read_csv(file)
        return redirect(url_for('dashboard'))
    else:
        return "Invalid file format. Please upload a CSV file."

@app.route('/dashboard')
def dashboard():
    global data
    if data is None:
        return "No data available. Please upload a CSV file first."

    return render_template('dashboard.html', data_columns=data.columns)

@app.route('/plot', methods=['POST'])
def plot():
    global data
    if data is None:
        return "No data available. Please upload a CSV file."

    selected_columns = request.form.getlist('columns')
    chart_type = request.form.get('chart_type')

    if chart_type not in ['scatter', 'bar', 'line', 'pie']:
        return "Invalid chart type."

    if len(selected_columns) < 2:
        return "Please select at least two columns."

    try:
        if chart_type == 'scatter':
            fig = px.scatter(data, x=selected_columns[0], y=selected_columns[1],
title=chart_type)
        elif chart_type == 'bar':
```

```
            fig = px.bar(data, x=selected_columns[0], y=selected_columns[1],
title=chart_type)
        elif chart_type == 'line':
            fig = px.line(data, x=selected_columns[0], y=selected_columns[1],
title=chart_type)
        elif chart_type == 'pie':
                        fig = px.pie(data, names=selected_columns[0],
values=selected_columns[1], title=chart_type)
        else:
            return "Invalid chart type."

        chart_json = fig.to_json()
                return render_template('plot.html', chart_json=chart_json,
chart_type=chart_type)
    except Exception as e:
        return f"Error: {str(e)}. Please check selected columns and try again."

if __name__ == '__main__':
    app.run(debug=True)
```

The provided Python script is the backend for a web-based data visualization tool built with Flask, a web framework. This application allows users to upload CSV files containing data and visualize it through various chart types.

To run the application, Flask and additional libraries need to be installed. Once the script is executed, the tool can be accessed through a web browser. Users can upload a CSV file, explore available columns on the dashboard, and visualize data patterns through interactive charts generated by Plotly Express.

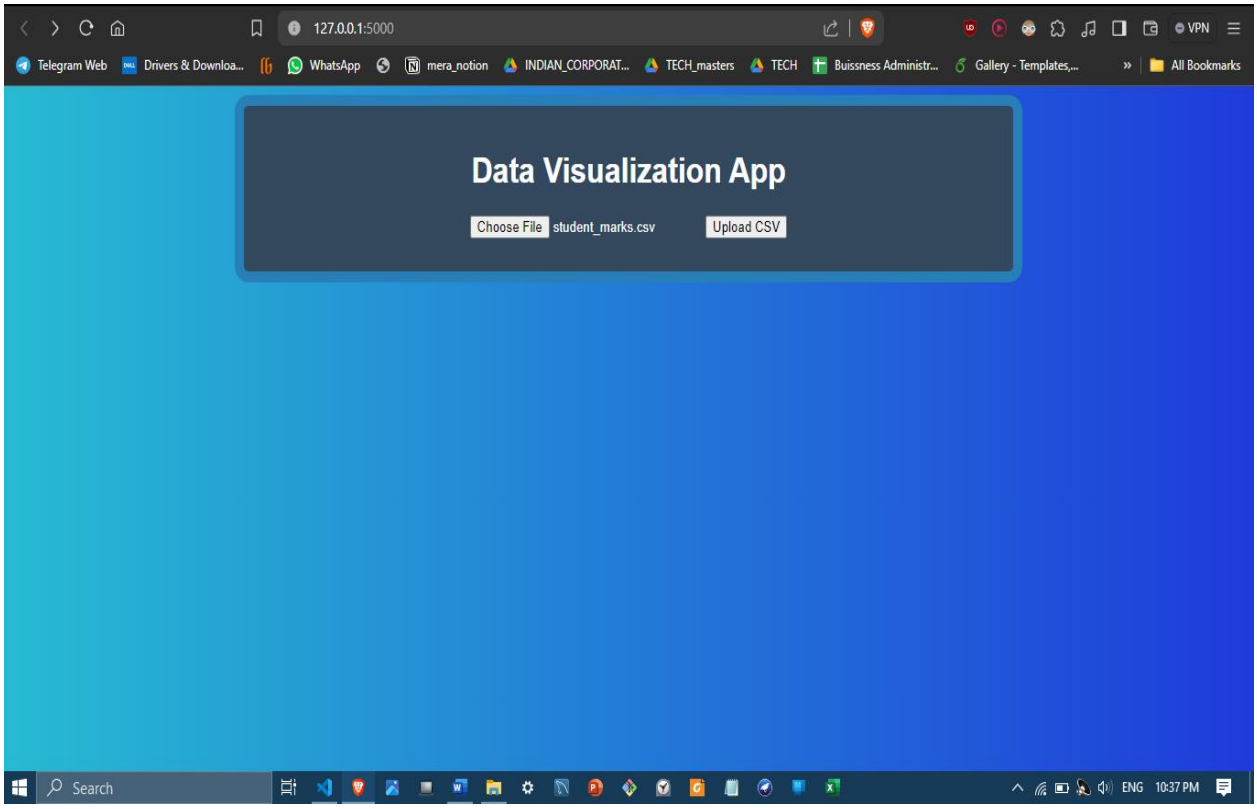
It's important to note that this script represents the backend logic of the application. The complete web application would require the corresponding frontend with HTML templates (`index.html`, `dashboard.html`, `plot.html`) and potentially CSS and JavaScript files for a seamless user experience.

DataVisualization front dashboard using index.html

```
<!-- index.html -->
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Data Visualization App</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <script src="{{ url_for('static', filename='script.js') }}"></script>
</head>
<body>
    <div class="container">
      <h1>Data Visualization App</h1>
    <form action="/upload" method="post" enctype="multipart/form-data">
        <input type="file" name="file" accept=".csv" required>
        <button type="submit">Upload CSV</button>
    </form>
    </div>
</body>
</html>
```

This looks like



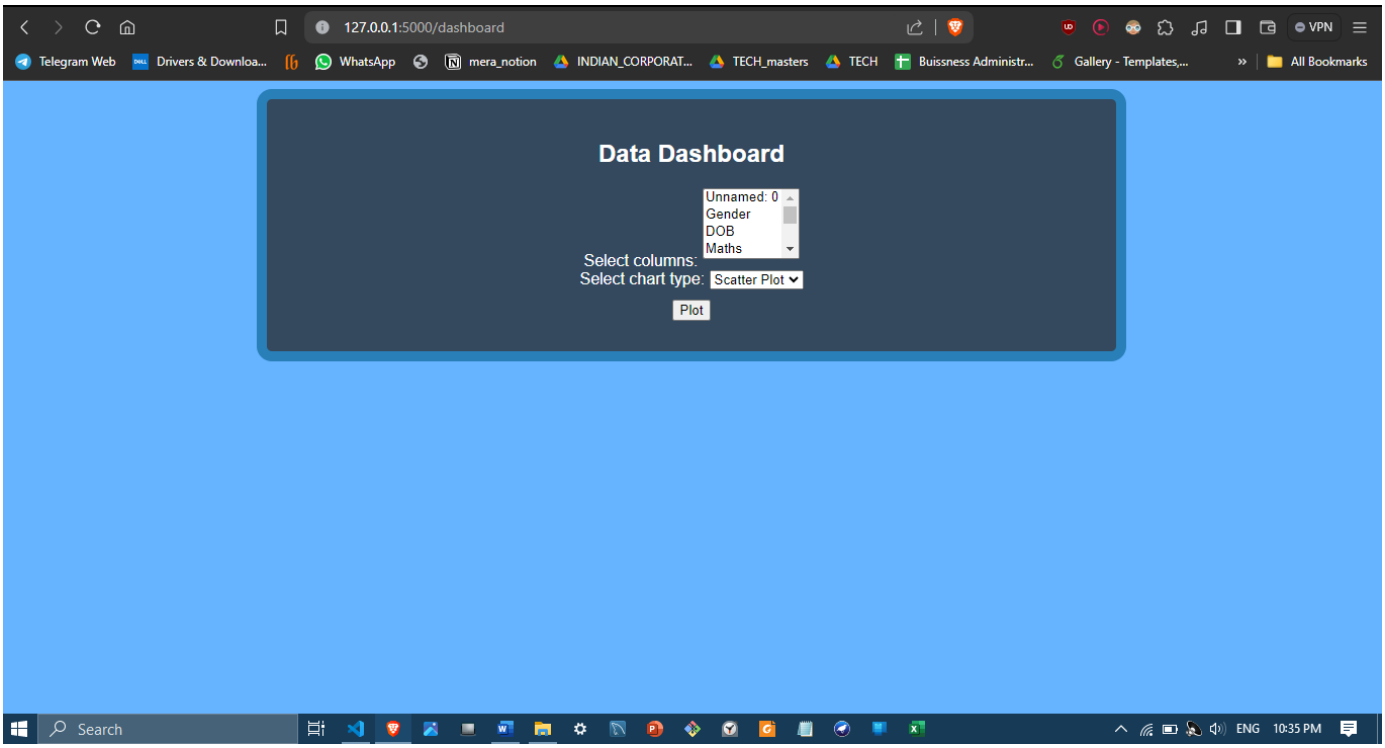
Dashboard.html

```
<!-- dashboard.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Data Visualization Dashboard</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <script src="{{ url_for('static', filename='script.js') }}"></script>
    <script>
        // Function to change background color
        function changeBackgroundColor() {
            var colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']; // Add more
colors as needed
            var randomColor = colors[Math.floor(Math.random() * colors.length)];
            document.body.style.backgroundColor = randomColor;
        }

        // Change background color on page load
        window.onload = function () {
            changeBackgroundColor();
        };
    </script>
</head>
<body>
    <div class="container">
        <h2>Data Dashboard</h2>
        <form action="/plot" method="post">
            <label for="columns">Select columns:</label>
            <select name="columns" id="columns" multiple required>
                {% for column in data_columns %}
                    <option value="{{ column }}">{{ column }}</option>
                {% endfor %}
            </select>
            <br>
            <label for="chart_type">Select chart type:</label>
            <select name="chart_type" id="chart_type" required>
                <option value="scatter">Scatter Plot</option>
                <option value="bar">Bar Chart</option>
                <option value="line">Line Chart</option>
                <option value="pie">Pie Chart</option>
            </select>
            <br>
```

```
            <button type="submit">Plot</button>
        </form>
    </div>
</body>
</html>
```

This the part of the frontend that shows the dashboard

This HTML file constitutes a data visualization dashboard template. It includes a dynamic background color-changing function on page load. The dashboard features a selection form allowing users to choose columns from available data (`data_columns`) and pick a chart type (scatter, bar, line, or pie). Upon submission, the form directs to a plotting route (`/plot`) using the HTTP POST method. The design is clean, utilizing external styles and scripts. The background color adds a visual element, and the interactive form facilitates user-driven data exploration and visualization.



Plot.html

This html is used to plot various plots

```
<!-- plot.html -->
<!DOCTYPE html>
```

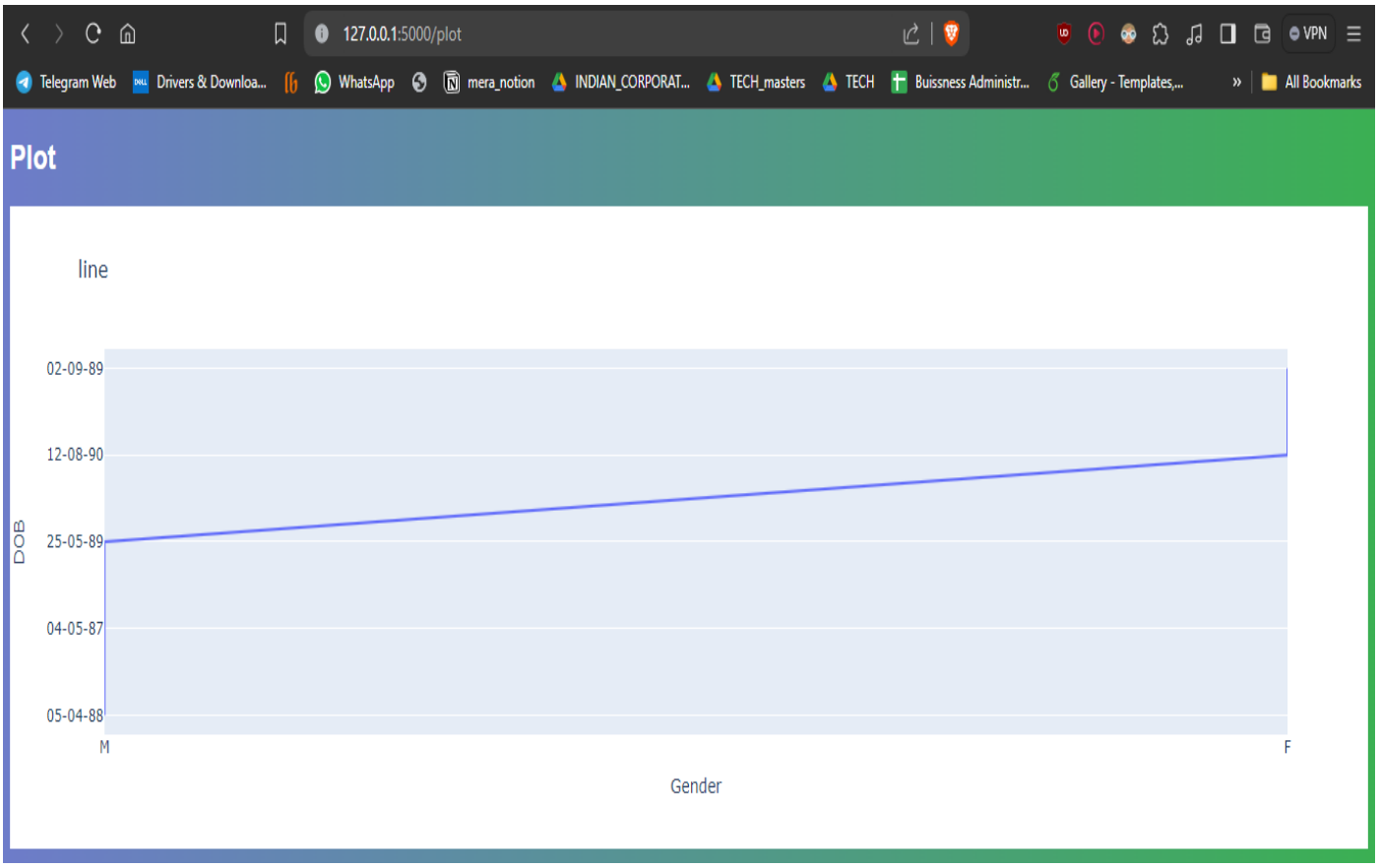
```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Data Visualization Plot</title>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <script src="{{ url_for('static', filename='script.js') }}"></script>
</head>
<body>
    <h2>Plot</h2>

    {% if chart_type == 'scatter' %}
        <div id="scatter-container"></div>
        <script>
            var scatterData = {{ chart_json | safe }};
            Plotly.newPlot('scatter-container', scatterData);
        </script>
    {% elif chart_type == 'bar' %}
        <div id="bar-container"></div>
        <script>
            var barData = {{ chart_json | safe }};
            Plotly.newPlot('bar-container', barData);
        </script>
    {% elif chart_type == 'line' %}
        <div id="line-container"></div>
        <script>
            var lineData = {{ chart_json | safe }};
            Plotly.newPlot('line-container', lineData);
        </script>
    {% elif chart_type == 'pie' %}
        <div id="pie-container"></div>
        <script>
            var pieData = {{ chart_json | safe }};
            Plotly.newPlot('pie-container', pieData);
        </script>
    {% endif %}
    <p>No chart data available.</p>
</body>
</html>
```

This HTML file serves as a template for displaying interactive data visualization plots using the Plotly library. The content of the webpage dynamically adjusts based on the provided `chart_type`.

It begins with a standard HTML structure, including meta tags, the title "Data Visualization Plot," and references to external dependencies.

The Plotly library is sourced from a CDN, ensuring the availability of its latest version. Additionally, external styles from 'styles.css' and scripts from 'script.js' are linked to the document. The body of the HTML includes a heading, "Plot," and dynamically generates content based on the `chart_type`. Four types of chart containers are defined: scatter, bar, line, and pie. Depending on the `chart_type` value, the corresponding container is filled with data fetched from the server in JSON format (`chart_json`). The Plotly library is then used to initialize and display the respective interactive chart within the designated container.

Conditional rendering using Jinja templating ensures that only the relevant chart type is displayed. In case of an unspecified or unsupported `chart_type`, a paragraph stating "No chart data available" is shown. In essence, this HTML file provides a flexible framework for visualizing data through different chart types, offering an interactive and customizable experience for data exploration and analysis.



CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

The conclusion provides a comprehensive summary of the project's achievements and outcomes. It reflects on how the developed data visualization application successfully addresses identified problems and enhances the user experience. This section serves as a critical reflection on the project's significance in contributing to the field of data analysis and visualization.

5.2 Future Work

The future work section outlines potential enhancements and extensions to the project. This includes the incorporation of additional chart types, expanding compatibility with different data formats, and exploring opportunities for integration with emerging technologies. Ensuring adaptability to evolving data analysis needs and technological advancements is a key focus of future work, aiming to keep the application at the forefront of innovation and responsiveness to user requirements. Continuous improvement and expansion are envisioned to position the application as a dynamic and evolving tool in the rapidly changing landscape of data analysis.