Internship

Explanation of the Project Code

Your project is a **web application for visualizing financial data** using **Flask** and **Plotly**. Here's a breakdown of each file:

---

1. `app.py` (The Backend)

- **Purpose:** This is the main backend file that runs the Flask web application.
- **Key Components:**
  - Uses **Flask** to create a web server.
  - Allows users to **upload a CSV file** containing financial data.
  - Reads the uploaded file and generates **various charts** (line, bar, histogram, and pie charts) using **Plotly**.
  - Displays the charts on a webpage.
- **How it Works:**
  1. The user visits the homepage (`/`).
  2. They **upload a CSV file**.
  3. The file is **saved** in the `uploads` folder.
  4. The app reads the file and extracts **the first two columns**.
  5. Generates **interactive charts** and displays them on the visualization page (`/visualize/filename`).

---

2. `style.css` (Styling)

- **Purpose:** Defines the visual style of the web application.
- **Key Features:**
  - **Center-aligns content**.
  - **Uses Arial font** for readability.
  - **Colors:**
    - **Headings:** Dark gray (`#333`).
    - **Button:** Blue background with white text.
    - **Background (set inside `upload.html`)**: Mint green (`#A9FFD4`).

---

3. `upload.html` (Upload Page)

- **Purpose:** This is the **home page** where users upload CSV files.
- **Key Features:**
  - Contains a **file upload form**.
  - Provides a **sample CSV file download link**.
  - Uses **CSS styling** from `style.css` to look nice.

---

4. `visualization_plotly.html` (Visualization Page)

- **Purpose:** Displays **interactive charts** after a file is uploaded.
- **Key Features:**
  - Loads **Plotly.js** (a JavaScript library for data visualization).
  - Shows **4 types of charts**:
    - **Line Chart** (Financial Trend Analysis)
    - **Bar Chart** (Comparative Analysis)

    - **Histogram** (Data Distribution)
    - **Pie Chart** (Proportion Representation)
- The charts are generated using **JSON data** passed from `app.py`.

How to Present This Project in Class

Step-by-Step Explanation for Your Presentation

1. Introduction

- "Today, I am presenting a **web application** for **visualizing financial data** using **Flask and Plotly**."
- "This app allows users to upload a **CSV file** and instantly see **interactive charts**."

2. Show the `upload.html` Page

- "This is the **home page** where users can upload their financial data in CSV format."
- "**A sample file is available for download** to test the app."

3. Show How the File Upload Works

- Upload a sample CSV file and click the "Upload" button.

4. Explain How `app.py` Processes the File

- "The file is **saved in the `uploads` folder**`."
- "The program reads the **first two columns** of the CSV and **creates charts**."

5. Show the `visualization_plotly.html` Page

- "After uploading, we see **four different types of charts**:"
    - **Line Chart:** "Shows trends in financial data."
    - **Bar Chart:** "Compares different categories."
    - **Histogram:** "Shows data distribution."
    - **Pie Chart:** "Represents proportion data."

6. Conclusion

- "This project is useful for **quick financial data analysis**."
- "It is built with **Flask (backend), HTML/CSS (frontend), and Plotly (visualization).**"
- "Thank you! Any questions?"

---

app.py

Explanation of `app.py` (Step-by-Step)

This Python script **creates a web application** using **Flask** that allows users to **upload a CSV file** and visualize the data using **interactive charts**.

---

1. Importing Required Libraries

```
from flask import Flask, render_template, request, redirect, url_for
import os
import pandas as pd
import plotly.express as px
import plotly
import json
```

- `Flask`: The web framework used to build the application.
- `render_template`: Loads HTML templates.
- `request`: Handles file uploads from users.
- `redirect, url_for`: Redirects users after uploading files.
- `os`: Handles file operations (saving uploaded files).
- `pandas`: Reads and processes CSV files.
- `plotly.express`: Creates interactive charts.
- `json`: Converts charts into JSON format for display.

---

2. Flask Application Setup

```
app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
```

- `app = Flask(__name__)`: Creates a Flask web application.
- `UPLOAD_FOLDER = 'uploads'`: Defines a folder to store uploaded CSV files.
- `os.makedirs(UPLOAD_FOLDER, exist_ok=True)`: Creates the folder if it does not exist.

---

3. Homepage – Uploading a File

```
@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files['file']
        if file.filename == '':
            return redirect(request.url)
        if file:
            filename = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
            file.save(filename)
            return redirect(url_for('visualize', filename=file.filename))
    return render_template('upload.html')
```

- **What This Does:**
  1. If the user **opens the homepage (`/`)**, it shows the **upload form** (`upload.html`).
  2. When the user **submits a file**, it:
     - Checks if a file is selected.
     - Saves it in the `uploads` folder.
     - Redirects the user to `/visualize/filename` to see charts.

---

4. Visualizing Data

```
@app.route('/visualize/<filename>')
def visualize(filename):
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    try:
        df = pd.read_csv(filepath)

        # Ensure the CSV has at least two columns
        if len(df.columns) >= 2:
            x_col = df.columns[0]
            y_col = df.columns[1]
```

- Loads the uploaded **CSV file** using `pandas`.
- **Ensures** the file has at least **two columns** for visualization.

---

5. Creating Charts

```
# Line Chart
line_fig = px.line(df, x=x_col, y=y_col, title="Financial Trend Analysis")
line_graphJSON = json.dumps(line_fig, cls=plotly.utils.PlotlyJSONEncoder)

# Bar Chart
bar_fig = px.bar(df, x=x_col, y=y_col, title="Comparative Bar Chart")
bar_graphJSON = json.dumps(bar_fig, cls=plotly.utils.PlotlyJSONEncoder)

# Histogram
hist_fig = px.histogram(df, x=y_col, title="Histogram of Financial Data")
hist_graphJSON = json.dumps(hist_fig, cls=plotly.utils.PlotlyJSONEncoder)

# Pie Chart
pie_fig = px.pie(df, names=x_col, values=y_col, title="Pie Chart of Financial Data")
pie_graphJSON = json.dumps(pie_fig, cls=plotly.utils.PlotlyJSONEncoder)
```

- **Creates four different types of charts using Plotly**:
  1. **Line Chart**: Shows trends in data.
  2. **Bar Chart**: Compares different categories.
  3. **Histogram**: Displays frequency distribution.
  4. **Pie Chart**: Represents proportions.

---

6. Sending Data to the Visualization Page

```
return render_template(
    'visualization_plotly.html',
    line_graphJSON=line_graphJSON,
    bar_graphJSON=bar_graphJSON,
    hist_graphJSON=hist_graphJSON,
    pie_graphJSON=pie_graphJSON,
```

```
    filename=filename
)
```

- Sends **JSON data of charts** to `visualization_plotly.html`, which will display them using **Plotly.js**.

---

7. Error Handling

```
else:
    return "CSV needs at least two columns for financial visualization.", 400
except Exception as e:
    return f"Error processing file: {str(e)}", 500
```

- If the CSV file **does not have at least two columns**, it shows an error.
- If **any error** occurs, it displays an error message.

---

8. Running the Application

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

- Runs the application **in debug mode**, allowing easy troubleshooting.

---

style.css

Detailed Explanation of `style.css`

This file is responsible for **styling the web pages** of your Flask application. It controls the **layout, colors, fonts, and appearance** of elements like buttons, text, and forms.

---

1. Styling the Entire Page (`body`)

```
body {
    font-family: Arial, sans-serif;
    margin: 20px;
    text-align: center;
}
```

- `font-family: Arial, sans-serif;` → Uses **Arial** font (or another sans-serif font if Arial isn't available).
- `margin: 20px;` → Adds **space around the page** so elements aren't stuck to the edge.
- `text-align: center;` → **Centers all text** and elements inside the page.

---

2. Styling Headings (`h1`)

```
h1 {
    color: #333;
}
```

- `color: #333;` → Makes the heading text a **dark gray color** (`#333` is almost black).

---

3. Styling Forms (`form`)

```
form {
    margin: 20px;
}
```

- `margin: 20px;` → Adds **space around the form** so it's not too close to other elements.

---

4. Styling Input Fields (`input`)

```
input {
    padding: 10px;
    margin: 10px;
}
```

- `padding: 10px;` → **Increases the size** of the text box for easier typing.
- `margin: 10px;` → **Adds space** around the input fields so they don't touch each other.

---

5. Styling Buttons (`button`)

```css
button {
    padding: 10px 20px;
    background-color: blue;
    color: white;
    border: none;
    cursor: pointer;
}
```

- `padding: 10px 20px;` → Makes the **button bigger** for easier clicking.
- `background-color: blue;` → The **button color is blue**.
- `color: white;` → The **text inside the button is white**.
- `border: none;` → Removes the **default button border**.
- `cursor: pointer;` → Changes the **mouse pointer to a hand** when hovering over the button.

---

upload.html

Detailed Explanation of `upload.html`

This file is the **home page** of the web application, where users can **upload a CSV file**. It contains HTML and CSS to structure and style the page.

---

1. HTML Boilerplate (`<html>`, `<head>`, `<body>`)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Data Visualization</title>
```

What This Does

- `<!DOCTYPE html>`: Declares the file as an HTML5 document.
- `<html lang="en">`: Sets the language to English.
- `<head>`: Contains meta information like **character encoding** and **page title**.
- `<meta charset="UTF-8">`: Ensures special characters (like ₹, €, etc.) are displayed correctly.
- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Makes the page responsive for mobile users.
- `<title>Data Visualization</title>`: Sets the browser tab title.

---

2. Linking External CSS (`style.css`)

```
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
```

- Loads the **external CSS file (`style.css`)** from the `static/` folder.
- This file styles elements like buttons, text, and form fields.

---

3. Inline CSS for Additional Styling

```
<style>
    body {
        background-color: #A9FFD4; /* Soft Mint Green */
        color: #333; /* Dark Gray text for contrast */
        font-family: Arial, sans-serif;
        margin: 20px;
        text-align: center;
    }
</style>
```

- `background-color: #A9FFD4;` → Sets a **mint green background**.
- `color: #333;` → Makes the text **dark gray** for readability.
- `font-family: Arial, sans-serif;` → Uses Arial or similar fonts.
- `margin: 20px;` → Adds spacing around the page.
- `text-align: center;` → **Centers all elements**.

4. Page Heading (`<h1>`)

```
<h1>Upload Financial Data CSV</h1>
```

- Displays a **heading** on the page.
- The text is styled using **CSS** (`h1 { color: #333; }`).

---

5. File Upload Form

```
<form action="/" method="post" enctype="multipart/form-data">
    <input type="file" name="file">
    <button type="submit">Upload</button>
</form>
```

What This Does

- `<form action="/">`: When the user submits a file, it is sent to the **Flask backend (`app.py`)**.
- `method="post"`: Uses **POST method** to send the file (instead of just loading a page).
- `enctype="multipart/form-data"`: **Enables file uploads**.
- `<input type="file" name="file">`: **File selection field** for the user.
- `<button type="submit">Upload</button>`: **Submit button** to upload the file.

---

6. Sample CSV File Download

```
<h3>
    <a href="https://cdn.discordapp.com/attachments/1160249560518643804/1348335479296360458/financial_da
        Download Sample CSV File
    </a>
</h3>
```

What This Does

- Provides a **link to download a sample CSV file**.
- The `download` attribute ensures the file is downloaded instead of opening in the browser.

---

7. Closing Tags

```
</body>
</html>
```

- Closes the **body** and **HTML** elements.

Summary

1. **Displays a file upload form** with a button.
2. **Uses CSS** for a mint-green background and centered content.
3. **Links an external stylesheet** for better styling.
4. **Provides a sample CSV file** for users to test.

This is the **first step** in the web app workflow—after uploading, the user is redirected to the visualization page.

Let me know if you need **further simplification or changes!** 🚀

visualization_plotly.html

Detailed Explanation of `visualization_plotly.html`

This HTML file is responsible for **displaying interactive charts** after the user uploads a CSV file. It uses **Plotly.js**, a JavaScript library, to visualize financial data.

---

1. HTML Boilerplate (`<html>`, `<head>`, `<body>`)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Visualization of {{ filename }}</title>
```

What This Does

- `<!DOCTYPE html>`: Declares the file as an **HTML5 document**.
- `<html lang="en">`: Sets the **language to English**.
- `<meta charset="UTF-8">`: Ensures **special characters (₹, €, etc.)** are displayed correctly.
- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Makes the page **responsive** for mobile and desktop users.
- `<title>Visualization of {{ filename }}</title>`:
  - `{{ filename }}` is a **placeholder** filled by Flask.
  - Displays the **name of the uploaded file** in the tab title.

---

2. Importing Plotly.js

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
```

- Loads **Plotly.js**, which is used to generate the **interactive charts**.

---

3. Page Heading

```
<h1>Visualization of {{ filename }}</h1>
```

- Displays the **filename of the uploaded CSV**.
- Example output: **"Visualization of financial_data.csv"**.

---

4. Divs for Different Charts

These `<div>` elements act as **containers** where the charts will be displayed.

```
<h2>Financial Trend Analysis (Line Chart)</h2>
<div id="line_chart"></div>

<h2>Comparative Analysis (Bar Chart)</h2>
```

```
<div id="bar_chart"></div>

<h2>Data Distribution (Histogram)</h2>
<div id="histogram"></div>

<h2>Pie Chart Representation</h2>
<div id="pie_chart"></div>
```

- `<h2>` **elements**: Section headings for different charts.
- `<div>` **elements**: Empty placeholders where **Plotly will insert the charts dynamically**.

---

5. JavaScript to Generate Charts

```
<script>
    var line_graph = {{ line_graphJSON | safe }};
    Plotly.newPlot('line_chart', line_graph.data, line_graph.layout);
```

What This Does

- `{{ line_graphJSON | safe }}`:
  - This is a **JSON object passed from Flask**.
  - It contains the **data and layout** for the **line chart**.
- `Plotly.newPlot('line_chart', line_graph.data, line_graph.layout);`
  - **Generates the line chart** inside the `<div id="line_chart">`.

---

6. Creating Other Charts

The same logic applies to the **bar chart, histogram, and pie chart**.

```
var bar_graph = {{ bar_graphJSON | safe }};
Plotly.newPlot('bar_chart', bar_graph.data, bar_graph.layout);

var hist_graph = {{ hist_graphJSON | safe }};
Plotly.newPlot('histogram', hist_graph.data, hist_graph.layout);

var pie_graph = {{ pie_graphJSON | safe }};
Plotly.newPlot('pie_chart', pie_graph.data, pie_graph.layout);
```

- **Each variable** (e.g., `bar_graphJSON`, `hist_graphJSON`, `pie_graphJSON`) contains data for its respective chart.
- `Plotly.newPlot()` renders the chart inside the corresponding `<div>`.

---

7. Closing Tags

```
</body>
</html>
```

- Closes the **body** and **HTML document**.

How This Page Works (Step-by-Step)

1. The user **uploads a CSV file** on `upload.html`.

2. Flask processes the file and **extracts data**.
3. Flask **creates JSON objects** containing **chart data**.
4. This page **receives those JSON objects** and uses **JavaScript + Plotly** to generate charts.

Summary

- This page **displays four interactive charts** (Line, Bar, Histogram, Pie).
- It uses **Plotly.js** to dynamically generate **data visualizations**.
- Flask passes **chart data** as JSON, which is inserted into the page using **JavaScript**.

This ensures users **instantly see financial trends** after uploading their data. 🚀 Let me know if you need any modifications!

CSV

```
Date,Stock Price (USD),Revenue (Million USD),Expenses (Million USD),Net Profit (Million USD),Volume Traded
2024-01-01,150,50,30,20,1000000
2024-01-02,152,52,31,21,1100000
2024-01-03,148,49,29,20,1050000
2024-01-04,155,55,32,23,1200000
2024-01-05,157,58,34,24,1250000
2024-01-06,160,60,35,25,1300000
2024-01-07,158,59,33,26,1280000
2024-01-08,162,62,36,26,1350000
2024-01-09,165,65,38,27,1400000
2024-01-10,163,63,37,26,1380000
```

Explanation of the CSV Data

This **CSV file contains financial data** related to stock prices and company revenue over **10 days** (from January 1 to January 10, 2024).

Understanding Each Column

Column Name	Meaning
Date	The date of the financial record.
Stock Price (USD)	The stock price of a company in US dollars.
Revenue (Million USD)	The company's revenue (in millions of USD).
Expenses (Million USD)	The company's expenses (in millions of USD).
Net Profit (Million USD)	Profit after subtracting expenses from revenue.
Volume Traded	The number of shares traded on that day.

Breaking Down the Data

Let's analyze the **first few rows**:

Date	Stock Price (USD)	Revenue (Million USD)	Expenses (Million USD)	Net Profit (Million USD)	Volume Traded
2024-01-01	150	50	30	20	1,000,000
2024-01-02	152	52	31	21	1,100,000
2024-01-03	148	49	29	20	1,050,000
2024-01-04	155	55	32	23	1,200,000
2024-01-05	157	58	34	24	1,250,000

Observations from the Data

1. **Stock Price Trends**

- The stock price **fluctuates daily**.
- **Lowest Price:** 148 (on 2024-01-03).
- **Highest Price:** 165 (on 2024-01-09).

2. **Revenue & Profit Growth**
   - Revenue **increases** from **50M USD (Jan 1) to 65M USD (Jan 9)**.
   - Net Profit also **increases**, which suggests **better financial performance**.
3. **Trading Volume**
   - **More shares are traded as stock price rises**.
   - **Highest volume:** 1,400,000 shares (on 2024-01-09).
   - **Lowest volume:** 1,000,000 shares (on 2024-01-01).

How Your Flask App Uses This CSV

1. **User uploads this CSV file.**
2. **Flask reads the data** and extracts:
   - Dates (X-axis).
   - Stock Price, Revenue, Expenses, Profit (Y-axis).
3. **Generates interactive charts**:
   - **Line Chart** for **Stock Price over Time**.
   - **Bar Chart** for **Revenue & Expenses Comparison**.
   - **Histogram** for **Trading Volume Distribution**.
   - **Pie Chart** for **Profit Breakdown**.

Final Summary

- This CSV records **financial performance** of a stock over **10 days**.
- It includes **stock prices, revenue, expenses, profit, and trading volume**.
- The data helps in **analyzing trends and making business decisions**.
- Your **Flask app will visualize** this data using **interactive graphs**.

Would you like to modify this CSV for better insights? 🚀