Steps to Run `traffic_analysis_dashboard.py` on a New PC

Libraries Required

Before running the code, ensure that the following Python libraries are installed:

1. **Dash**
    - For building the dashboard.
    - Install using: `pip install dash`
2. **Dash Bootstrap Components (dbc)**
    - For styling and layout using Bootstrap.
    - Install using: `pip install dash-bootstrap-components`
3. **Pandas**
    - For handling and manipulating data.
    - Install using: `pip install pandas`
4. **Plotly**
    - For creating interactive visualizations.
    - Install using: `pip install plotly`
5. **Base64** (Built-in)
    - For handling encoded content strings (built into Python).
6. **IO** (Built-in)
    - For input/output operations (built into Python).

Full Installation Command:

```
pip install dash dash-bootstrap-components pandas plotly
```

Steps to Run the Code

1. **Ensure Python is Installed**:
    - Install Python (version 3.7 or higher).
    - Check the installation by running: `python --version`.
2. **Download the Script**:
    - Save the `traffic_analysis_dashboard.py` file in a folder on your PC.
3. **Open a Terminal/Command Prompt**:
    - Navigate to the folder where `traffic_analysis_dashboard.py` is saved.
4. **Install Dependencies**:
    - Run the following command to install the required libraries:

      ```
      pip install dash dash-bootstrap-components pandas plotly
      ```
5. **Run the Script**:
    - Execute the script by running:

      ```
      python traffic_analysis_dashboard.py
      ```
6. **Access the Dashboard**:
    - Open a browser and navigate to: `http://127.0.0.1:8050`
    - This is the local address where the Dash app will be hosted.

Code Functionality Overview

1. Header Section

- Displays a logo, title (“Urban Traffic Congestion Analysis”), and two buttons (“Dashboard” and “Contact Us”).
- Buttons link to external URLs.

2. Horizontal Line

- Adds a gradient-colored line below the header for design purposes.

3. Team Members and Upload Section

- **Team Members**:

    - Displays a list of project team members.

- **File Upload**:
    - Allows users to upload a CSV file.
    - Accepts traffic-related data with specific columns (explained below).

4. CSV Requirements

- The uploaded CSV file must have the following columns:
    - **Timestamp**: Datetime format.
    - **Location**: String.
    - **Vehicle_count**: Integer.
    - **Congestion_level**: Categorical (Low, Medium, High).

5. Benefits of the Analysis

- Lists the advantages of using the traffic dashboard, such as better urban planning, reducing fuel consumption, and more.

6. Graph Details

- Explains the types of visualizations in the dashboard:
    - **Scatter Map**: Displays traffic locations and congestion levels.
    - **Line Chart**: Shows traffic trends over time.
    - **Bar Chart**: Highlights average vehicle counts by location.

7. File Upload Handling

- The uploaded CSV file is read and processed.
- Missing values are filled using forward-fill (`ffill` method).
- Timestamps are converted to datetime format, and invalid rows are dropped.

8. Visualizations

- Three visualizations are created:
    1. **Scatter Map**:

- Plots traffic locations and congestion levels on a map.

2. **Line Chart**:
   - Shows trends in vehicle counts over time for each location.
3. **Bar Chart**:
   - Displays average vehicle counts for each location.

9. Callbacks

- A Dash callback updates the dashboard with uploaded data.
- Displays a data summary (first 10 rows) and graphs.

10. Running the App

- The app is hosted locally at `http://127.0.0.1:8050`.

Key Notes

1. **Ensure Data Compatibility**:
   - The uploaded CSV file must match the expected format (e.g., proper column names and data types).
2. **Customize Visualization**:
   - Latitude and longitude in the scatter map are placeholders; replace them with actual data from your dataset.
3. **Debugging**:
   - If errors occur, check the terminal for error messages.

By following the above steps and requirements, you can successfully run the `traffic_analysis_dashboard.py` script and analyze traffic congestion effectively.

```
import pandas as pd
import dash
from dash import dcc, html, Input, Output, dash_table
import dash_bootstrap_components as dbc
```

```
import plotly.express as px
import base64
import io
from sklearn.preprocessing import LabelEncoder

# Dash App
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])

# Layout
app.layout = dbc.Container([
    # Header Section
    dbc.Row([
        # Logo
        dbc.Col(html.Img(src="https://iac.edu.pk/wp-content/uploads/2024/02/logo.png",
height="80px"), width=2),

        # Title
        dbc.Col(html.H1("AI Traffic Congestion Analysis", className="text-center"), width=8),

        # Buttons
        dbc.Col([
            html.A(
                dbc.Button(
                    "Dashboard",
                    className="mx-1 btn-hover",
                    style={
                        "width": "100px",
                        "backgroundColor": "#ff0000",
                        "color": "white",
                        "fontWeight": "600",
                        "border": "none"
                    }
                ),
                href="https://iac.edu.pk/",
                target="_blank"
            ),
            html.A(
                dbc.Button(
                    "Contact Us",
                    color="success",
                    className="mx-1 btn-hover",
                    style={"width": "120px", "fontWeight": "600"}
                ),
                href="https://iac.edu.pk/contact-us/",
```

```
            target="_blank"
        ),
    ], width=2, className="d-flex align-items-center justify-content-end"),
], align="center"),

# Horizontal Line (Three Colors)
html.Div(style={
    "width": "100%",
    "height": "5px",
    "background": "linear-gradient(to right, red, yellow, green)",
    "marginBottom": "20px"
}),

# Team Members and Upload Section in a Single Row
dbc.Row([
    # Team Members Section
    dbc.Col([
        html.H4("Project Team Members"),
        html.Ul([
            html.Li("1. Falak"),
            html.Li("2. Amber"),
            html.Li("3. Tazeem"),
            html.Li("4. Suleman")
        ])
    ], width=6, style={"paddingLeft": "60px"}),

    # Upload Section
    dbc.Col([
        html.H4("Upload Your CSV File", className="text-center", style={"paddingBottom": "25px"}),

        dcc.Upload(
            id="upload-data",
            children=html.Div(["Drag and Drop or ", html.A("Select a CSV File")]),
            style={
                "width": "100%",
                "height": "60px",
                "lineHeight": "60px",
                "borderWidth": "1px",
                "borderStyle": "dashed",
                "borderRadius": "5px",
                "textAlign": "center",
                "marginBottom": "20px",
            },
        )
```

```
        ], width=6, style={"paddingRight": "60px"}),
    ], className="mb-4"),

    # CSV Requirements, Benefits, and Graph Details in One Row
    dbc.Row([
        dbc.Col([
            html.H4("CSV File Requirements", style={"color": "red", "paddingLeft": "30px"}),
            html.Ul([
                html.Li("Timestamp - Datetime"),
                html.Li("Location - String"),
                html.Li("Vehicle_count - Integer"),
                html.Li("Country_city - String")
            ])
        ], width=4),
        dbc.Col([
            html.H4("Benefits of Using This Analysis", style={"color": "#ffeb00", "paddingLeft": "30px"}),
            html.Ul([
                html.Li("Identifies high-congestion areas for better urban planning."),
                html.Li("Predicts traffic trends to optimize commute times."),
                html.Li("Helps reduce fuel consumption and emissions."),
                html.Li("Supports efficient emergency and logistics routing.")
            ])
        ], width=4),
        dbc.Col([
            html.H4("Traffic Graph Details", style={"color": "green", "paddingLeft": "30px"}),
            html.Ul([
                html.Li("Scatter Map: Shows traffic locations and congestion levels across the city."),
                html.Li("Line Chart: Displays trends in vehicle counts over time at different locations."),
                html.Li("Bar Chart: Highlights average vehicle counts by location for identifying hotspots.")
            ])
        ], width=4),
    ], className="mb-4"),

    # Output Section
    html.Div(id="output-data-upload"),
], fluid=True)

# Helper Functions
def parse_data(contents, filename):
    content_type, content_string = contents.split(",")
    decoded = base64.b64decode(content_string)
    try:
        if "csv" in filename:
            # Assume CSV format
```

```
            df = pd.read_csv(io.StringIO(decoded.decode("utf-8")))
            return df
    except Exception as e:
        print(f"Error parsing file: {e}")
        return None
    return None

# Approximate location data for locations in London (lat, lon)
location_coordinates = {
    'Westminster': (51.4974, -0.1278),
    'Camden': (51.5292, -0.1426),
    'Islington': (51.5364, -0.1037),
    'Kensington': (51.4974, -0.1925),
    'Hackney': (51.5471, -0.0464),
    'Bromley': (51.4052, 0.0167),
    'Greenwich': (51.4769, 0.0005),
    'Croydon': (51.3760, -0.0980),
    'Brent': (51.5583, -0.2817),
    'Tower Hamlets': (51.5074, -0.0290)
}

# Callback to handle uploaded data
@app.callback(
    Output("output-data-upload", "children"),
    Input("upload-data", "contents"),
    Input("upload-data", "filename")
)
def update_output(contents, filename):
    if contents is None:
        return html.Div()

    # Parse the file
    df = parse_data(contents, filename)

    if df is None:
        return html.Div("Invalid file format. Please upload a valid CSV.")

    # 2. Print Detail of Fields & 3. Use Describe Command
    field_details = html.Div([
        html.H4("Uploaded Data Summary", className="my-3"),
        dash_table.DataTable(
            data=df.describe().reset_index().to_dict("records"),  # Using describe() to summarize the fields
            columns=[{"name": i, "id": i} for i in df.describe().reset_index().columns],
            style_table={"overflowX": "auto"},
            style_header={"backgroundColor": "rgb(30, 30, 30)", "color": "white"},
```

```
            style_cell={"textAlign": "center", "padding": "10px"}
        )
    ])

    # 4. Handle Missing Values
    df.fillna(method="ffill", inplace=True)

    # 5. Label Encoding/One Hot Encoding
    if "location" in df.columns:
        df["location_encoded"] = LabelEncoder().fit_transform(df["location"])  # Label Encoding for
'location' column

    # Ensure timestamp is in datetime format
    df["timestamp"] = pd.to_datetime(df["timestamp"], errors="coerce")
    df.dropna(subset=["timestamp"], inplace=True)

    # Get latitude and longitude from location names
    latitudes = []
    longitudes = []
    for location in df['location']:
        if location in location_coordinates:
            latitudes.append(location_coordinates[location][0])
            longitudes.append(location_coordinates[location][1])
        else:
            latitudes.append(51.5074)  # Default to central London
            longitudes.append(-0.1278)  # Default to central London

    # Add latitudes and longitudes to the DataFrame
    df['latitude'] = latitudes
    df['longitude'] = longitudes

    # Calculate the statistics for the first 50 rows
    df_top_50 = df.head(50)
    location_stats = df_top_50.groupby('location')['vehicle_count'].agg(['max', 'median']).reset_index()

    # Categorize the locations based on max vehicle count
    max_value = location_stats['max'].max()
    min_value = location_stats['max'].min()
    median_value = location_stats['max'].median()

    def categorize(row):
        if row['max'] == max_value:
            return 'High'
        elif row['max'] == median_value:
            return 'Medium'
```

```
    else:
        return 'Low'

location_stats['category'] = location_stats.apply(categorize, axis=1)

# Horizontal Bar Chart for Locations Categorized by Vehicle Count
fig_location_category = px.bar(
    location_stats,
    x='max',
    y='location',
    color='category',
    orientation='h',
    title="Location Congestion Areas  ",
    labels={"max": "Max Vehicle Count", "location": "Location"},
    color_discrete_map={"High": "red", "Medium": "yellow", "Low": "green"}
)

# Visualizations
try:
    fig_map = px.scatter_mapbox(
        df,
        lat="latitude",
        lon="longitude",
        size="vehicle_count",
        color="location",
        title="Traffic Locations",
        mapbox_style="open-street-map",
        zoom=10,
    )
    fig_map.update_layout(
        margin={"r": 0, "t": 40, "l": 0, "b": 0},
        height=500
    )

    fig_trend = px.line(
        df, x="timestamp", y="vehicle_count", color="location",
        title="Traffic Trends Over Time",
        markers=True
    )
    fig_trend.update_traces(line=dict(width=2.5))
    fig_trend.update_layout(hovermode="x unified", height=500)

    fig_bar = px.bar(
        df.groupby("location")["vehicle_count"].mean().reset_index(),
        x="location",
```

```
            y="vehicle_count",
            color="location",
            title="Average Vehicle Count per Location",
            text_auto=True
        )
        fig_bar.update_traces(marker=dict(line=dict(width=2, color="black")))

    except Exception as e:
        return html.Div(f"Error generating graphs: {e}")

    return html.Div([
        field_details,
        html.H4("Visualizations", className="my-4 text-center"),
        dcc.Graph(figure=fig_map),
        dcc.Graph(figure=fig_trend),
        dcc.Graph(figure=fig_bar),
        html.H4("Prediction of Traffic Trends", className="my-4 text-center"),
        dcc.Graph(figure=fig_location_category),
    ])

if __name__ == "__main__":
    app.run_server(debug=True)
```

o run the provided Python code on a new PC, follow these steps:

1. Install Python

If Python isn't installed yet, you'll need to install it:

- Go to python.org.
- Download and install the latest version (make sure to check "Add Python to PATH" during installation).

2. Install Required Libraries

The code uses `pandas` and `numpy`, which aren't built into Python by default. You need to install these libraries using `pip`. Here's how:

- Open a command prompt or terminal.
- Run the following commands to install `pandas` and `numpy`:
  bash
  Copy Edit
  ```
  pip install pandas numpy
  ```

3. Create a New Python Script

- Open a text editor (like Notepad or Visual Studio Code) or an IDE (like PyCharm or VS Code).
- Create a new file, e.g., `generate_traffic_data.py`, and paste the provided Python code into it.

4. Run the Python Script

- Save the file (`generate_traffic_data.py`).
- Open the command prompt or terminal.
- Navigate to the folder where the Python script is saved using the `cd` command (e.g., if the script is in `C:\Projects`, run):
  bash
  Copy Edit
  ```
  cd C:\Projects
  ```
- Run the script by typing:
  bash
  Copy Edit
  ```
  python generate_traffic_data.py
  ```

5. Check for the Generated CSV File

- After running the script, the file `traffic_data_london.csv` will be created in the same directory.
- Open the directory where you ran the script, and you'll see the CSV file containing the traffic data.

Let me know if you need further details or run into any issues!