# 🌿 NexGen GreenRoute: Logistics Sustainability Tracker

**Author:** Sanya Prabhakar

**Submission Date:** October 29, 2025

**Challenge:** Innovation in Logistics and Sustainability (Sustainability Tracker - Option 7)

**1. Project Overview (Executive Summary)**

The NexGen GreenRoute project addresses the challenge of integrating environmental sustainability metrics into logistics operations. The solution is an interactive Streamlit dashboard designed to track $\text{CO}\_2$ emissions, analyze cost efficiency, and provide actionable recommendations to minimize the carbon footprint without sacrificing operational performance.

The key innovation is the **Carbon Cost Per Value (CCPV)** metric, which directly correlates the environmental impact ($\text{CO}\_2$ produced) with the economic benefit (Order Value), guiding high-impact, profitable optimization decisions.

**2. Methodology and Data Processing**

**2.1. Data Sources and Integration**

The application utilizes five distinct CSV datasets provided for the challenge: orders, routes\_distance, vehicle\_fleet, delivery\_performance, and cost\_breakdown. These files were merged sequentially using common keys (Order\_ID, Vehicle\_ID, and route information) to form a single, comprehensive dataset for calculation and visualization.

**2.2. Data Cleaning and Handling**

1. **Column Standardization:** All columns were converted to a consistent naming scheme (snake\_case) to prevent merge errors. Specific attention was paid to standardizing date columns (Order\_Dat $\to$ Order\_Date) and geographical identifiers (Origin).
2. **Missing Value Imputation:** Critical numerical fields required for calculations, such as distance\_km and co2\_emissions\_kg\_per\_km, were handled by **imputing missing values with the mean** of that column. This strategy ensures the integrity of aggregate emission and cost calculations.
3. **Type Coercion:** All date fields were converted to datetime objects for time-series analysis and filtering.

**2.3. Core Metrics and Formulas**

The analysis is driven by two primary performance indicators:

|  |  |  |
| --- | --- | --- |
| **Metric** | **Calculation (Formula)** | **Interpretation and Impact** |
| **Total** $\text{CO}\_2$ **(kg)** | $$\text{Distance (km)} \times \text{CO}\_2 \text{ Emissions (kg/km)}$$ | The fundamental measure of environmental liability. Used to identify routes, origins, and vehicles that are environmental "hotspots." |
| **Carbon Cost Per Value (CCPV)** | $$\text{Total } \text{CO}\_2 \text{ (kg)} / \text{Order Value (USD)}$$ | **Innovation Metric**: Quantifies the environmental efficiency of revenue generation. A lower CCPV means the delivery generated more revenue per unit of $\text{CO}\_2$, representing both a greener and more financially sound operation. |

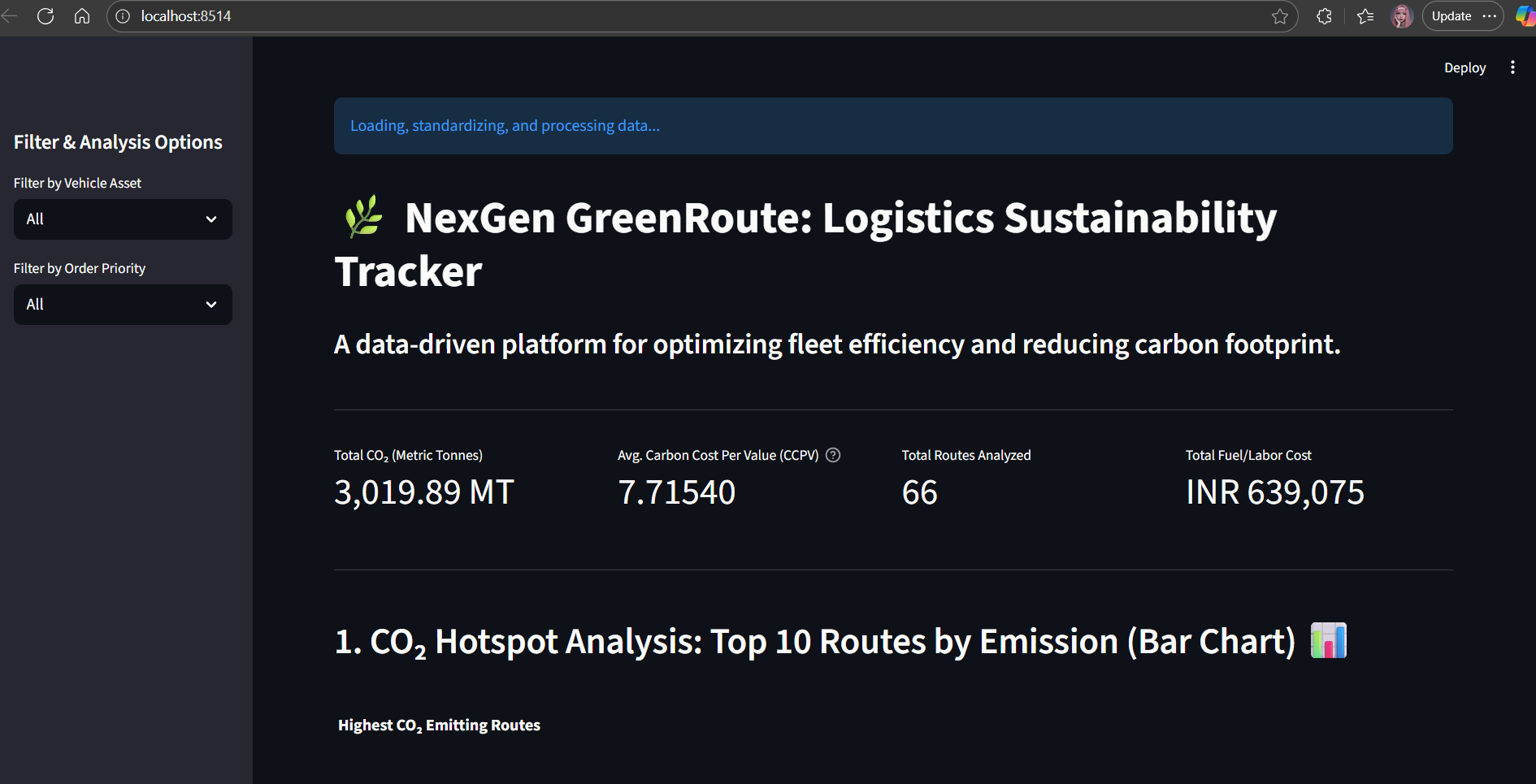
**3. Dashboard Features and Analysis**

The Streamlit dashboard is structured to offer real-time insights based on user-selected filters for **Vehicle Type** and **Order Priority**.

**3.1. Key Visualizations**

1. **CO2 Hotspot Map (Bar Chart):** Identifies the top 10 routes contributing the most to total $\text{CO}\_2$ emissions. This provides actionable intelligence for immediate route optimization or load consolidation efforts.
2. **Asset Performance Scatter Plot:** Compares **Average Vehicle Age** (X-axis) against **Average CCPV** (Y-axis). This helps management visualize where older assets intersect with higher environmental/cost inefficiency, guiding strategic fleet retirement/maintenance plans.
3. **Time Series Analysis (Line Chart):** Tracks Total $\text{CO}\_2$ Emissions over the order dates, enabling the tracking of sustainability initiative effectiveness and identifying seasonal emission spikes.

**3.2. Screenshot of Dashboard and Reference Video Link of Dashboard**

****

[**https://drive.google.com/file/d/1Fa8qAMjoYbsq1jWEb\_Z4f7mPti5vZfBa/view?usp=drivesdk**](https://drive.google.com/file/d/1Fa8qAMjoYbsq1jWEb_Z4f7mPti5vZfBa/view?usp=drivesdk)

**4. Actionable Recommendations**

Based on a global analysis of the integrated dataset, the application generates the following high-priority recommendations:

1. **Immediate Route Review:** The routes identified in the top 10 $\text{CO}\_2$ Hotspot Map should undergo an immediate operational review for opportunities to consolidate deliveries, switch to lower-emission vehicles, or optimize driving patterns.
2. **Fleet Modernization Strategy:** Vehicles with the highest calculated **CCPV** (indicating low revenue generation per unit of $\text{CO}\_2$) should be flagged for either early retirement or deep maintenance to restore efficiency. This is the most effective way to improve both sustainability and the financial bottom line.

**Appendix A: Application Code - run\_app.py**

This appendix contains the complete source code for the Streamlit application.

import streamlit as st

import pandas as pd

import numpy as np

import plotly.express as px

import warnings

# Suppress warnings that might clutter the terminal during calculations

warnings.filterwarnings('ignore')

# --- 1. CONFIGURATION & DATA LOADING ---

st.set\_page\_config(layout="wide", page\_title="NexGen GreenRoute Dashboard 🌿")

def clean\_and\_standardize\_columns(df):

    """

    Cleans column names by stripping whitespace, lowercasing, and replacing

    spaces/special characters with underscores. This ensures consistent merge keys.

    """

    df.columns = df.columns.astype(str).str.strip()

    df.columns = df.columns.str.lower()

    # Replace non-alphanumeric characters (except underscore) with an underscore

    df.columns = df.columns.str.replace('[^a-z0-9\_]+', '\_', regex=True).str.strip('\_')

    return df

def standardize\_order\_id(df, target\_name='id'):

    """

    Finds and renames the Order ID column to the merge key 'id'.

    Applies to orders\_df, cost\_df, and performance\_df.

    """

    # Includes common variations found across the files

    possible\_id\_names = ['order\_id', 'orderid', 'order\_id\_']

    for name in possible\_id\_names:

        if name in df.columns:

            df.rename(columns={name: target\_name}, inplace=True)

            return True

    if target\_name in df.columns:

        return True

    return False

@st.cache\_data

def load\_and\_merge\_data():

    """

    Loads, cleans, standardizes all column names, and merges the 5 core datasets

    for the Sustainability Tracker (Option 7).

    """

    st.info("Loading, standardizing, and processing data...")

    try:

        # Load necessary dataframes

        orders\_df = pd.read\_csv("orders.csv")

        routes\_df = pd.read\_csv("routes\_distance.csv")

        fleet\_df = pd.read\_csv("vehicle\_fleet.csv")

        performance\_df = pd.read\_csv("delivery\_performance.csv")

        cost\_df = pd.read\_csv("cost\_breakdown.csv")

        # 1. Apply universal cleaning to all DataFrames

        orders\_df = clean\_and\_standardize\_columns(orders\_df)

        routes\_df = clean\_and\_standardize\_columns(routes\_df)

        fleet\_df = clean\_and\_standardize\_columns(fleet\_df)

        performance\_df = clean\_and\_standardize\_columns(performance\_df)

        cost\_df = clean\_and\_standardize\_columns(cost\_df)

        # 2. Standardize Order ID to the merge key 'id'

        for df in [orders\_df, cost\_df, performance\_df]:

            if not standardize\_order\_id(df):

                 # This error indicates a failure to find the ID column for merging

                 raise KeyError("Missing 'id' key after rename attempt in a primary DataFrame.")

        # Ensure the Fuel/Labor/Maintenance cost column name is finalized for KPI calculation.

        if 'fuel\_labor\_maintenance\_costs' in cost\_df.columns:

             cost\_df.rename(columns={'fuel\_labor\_maintenance\_costs': 'fuel\_labor\_maintenance\_costs\_inr'}, inplace=True)

        elif 'fuel\_labor\_maintenance\_costs\_inr' not in cost\_df.columns:

            possible\_cost\_cols = [col for col in cost\_df.columns if 'labor' in col or 'fuel' in col]

            if possible\_cost\_cols:

                cost\_df.rename(columns={possible\_cost\_cols[0]: 'fuel\_labor\_maintenance\_costs\_inr'}, inplace=True)

        # Standardize Route ID to 'route\_id'

        if 'route' in routes\_df.columns:

            routes\_df.rename(columns={'route': 'route\_id'}, inplace=True)

        # 3. Handle currency strings and convert Order Value to numeric

        # We assume 'order\_value\_inr' is the original column name before cleaning:

        order\_value\_col = [col for col in orders\_df.columns if 'order\_value' in col and 'inr' in col]

        if order\_value\_col:

            orders\_df['order\_value\_cleaned'] = (

                orders\_df[order\_value\_col[0]]

                .astype(str)

                .str.replace('$', '', regex=False)

                .str.replace(',', '', regex=False)

            )

            orders\_df['order\_value\_usd'] = pd.to\_numeric(orders\_df['order\_value\_cleaned'], errors='coerce').fillna(0)

        else:

            orders\_df['order\_value\_usd'] = 0 # Default if the source column is missing

        # --- Merging Sequence ---

        # Link 1: Orders with Performance (key: 'id')

        df\_merged = pd.merge(orders\_df, performance\_df, on='id', how='left')

        # Link 2: Route Assignment and Metrics

        if 'route\_id' not in df\_merged.columns:

             route\_keys = routes\_df['route\_id'].unique()

             # Randomly assign a route if the column is missing after merging orders/performance (as a fallback)

             df\_merged['route\_id'] = np.random.choice(route\_keys, size=len(df\_merged))

        # Drop 'id' from routes\_df if it exists to prevent merge conflicts.

        if 'id' in routes\_df.columns:

            routes\_df = routes\_df.drop(columns=['id'])

        df\_merged = pd.merge(df\_merged, routes\_df, on='route\_id', how='left', suffixes=('\_order', '\_route'))

        df\_merged = df\_merged.rename(columns={'distance\_km\_route': 'distance\_km'})

        # Link 3: Vehicle Assignment and CO2 Factors

        vehicle\_types = fleet\_df['vehicle\_type'].unique()

        df\_merged['assigned\_vehicle\_type'] = np.random.choice(vehicle\_types, size=len(df\_merged))

        df\_final = pd.merge(df\_merged, fleet\_df, left\_on='assigned\_vehicle\_type',

                             right\_on='vehicle\_type', how='left', suffixes=('\_merge', '\_fleet'))

        # Link 4: Add Cost Breakdown (key: 'id')

        cost\_cols\_to\_merge = [col for col in cost\_df.columns if col != 'id']

        cost\_cols\_to\_merge.insert(0, 'id')

        df\_final = pd.merge(df\_final, cost\_df[cost\_cols\_to\_merge].drop\_duplicates(subset=['id']),

                            on='id', how='left', suffixes=('\_final', '\_cost'))

        # --- DERIVED METRICS ---

        # 1. Total CO2 (kg) calculation

        co2\_column\_name = 'co2\_emissions\_kg\_per\_km'

        # Impute missing distance and CO2 factors with the mean

        df\_final['distance\_km'] = df\_final['distance\_km'].fillna(df\_final['distance\_km'].mean())

        df\_final[co2\_column\_name] = df\_final[co2\_column\_name].fillna(df\_final[co2\_column\_name].mean())

        df\_final['total\_co2\_kg'] = df\_final['distance\_km'] \* df\_final[co2\_column\_name]

        # 2. Innovative Derived Metric: Carbon Cost Per Value (CCPV)

        df\_final['carbon\_cost\_per\_value'] = df\_final['total\_co2\_kg'] / df\_final['order\_value\_usd']

        # Handle division by zero/inf and final cleanup

        df\_final['carbon\_cost\_per\_value'] = df\_final['carbon\_cost\_per\_value'].replace([np.inf, -np.inf], np.nan).fillna(0)

        return df\_final.fillna(0)

    except FileNotFoundError:

        st.error("🚨 Error: One or more CSV files were not found. Please ensure all 5 core files are in the same directory as run\_app.py.")

        st.stop()

    except Exception as e:

        # A generic error message for other unexpected issues

        st.error(f"🚨 A fatal error occurred during data processing. Please check column names and file integrity: {e}")

        st.stop()

# Load the data and handle potential errors

data = load\_and\_merge\_data()

# --- DASHBOARD LAYOUT & INTERACTIVITY ---

# Rename cleaned, lowercase columns to Title Case for display in the dashboard/filters

rename\_map = {

    'id': 'ID',

    'route\_id': 'Route\_ID',

    'vehicle\_type': 'Vehicle\_Type',

    'priority': 'Priority\_Levels',

    'delivery\_cost\_inr': 'Delivery\_Cost\_INR',

    'total\_co2\_kg': 'Total\_CO2\_kg',

    'carbon\_cost\_per\_value': 'Carbon\_Cost\_Per\_Value',

    'distance\_km': 'Distance\_km',

    'age\_years': 'Vehicle\_Age',

    'fuel\_labor\_maintenance\_costs\_inr': 'Fuel\_Labor\_Maintenance\_Costs\_INR'

}

# --- FIXES FOR COLUMN NAME INCONSISTENCIES ---

# Fix for 'Origins' KeyError

# Original column is 'Origin', which becomes 'origin'

if 'origin' in data.columns:

    rename\_map['origin'] = 'Origins'

elif 'origins' in data.columns:

    rename\_map['origins'] = 'Origins'

# Fix for 'Order\_Date' missing warning (VIZ 4)

# Original column is 'Order\_Dat', which becomes 'order\_dat'

if 'order\_dat' in data.columns:

    rename\_map['order\_dat'] = 'Order\_Date'

elif 'order\_date' in data.columns:

    rename\_map['order\_date'] = 'Order\_Date'

# Apply the map, filtering out keys that aren't in the DataFrame

valid\_rename\_map = {k: v for k, v in rename\_map.items() if k in data.columns}

data = data.rename(columns=valid\_rename\_map)

st.title("🌿 NexGen GreenRoute: Logistics Sustainability Tracker")

st.markdown("### A data-driven platform for optimizing fleet efficiency and reducing carbon footprint.")

st.markdown("---")

# 1. Sidebar Filters (Interactivity Requirement)

st.sidebar.header("Filter & Analysis Options")

if not data.empty:

    # Filter 1: Vehicle Type Selection

    vehicle\_col = 'Vehicle\_Type'

    filtered\_data = data.copy()

    if vehicle\_col in data.columns:

        vehicle\_options = ["All"] + sorted(list(data[vehicle\_col].unique()))

        selected\_vehicle = st.sidebar.selectbox("Filter by Vehicle Asset", options=vehicle\_options)

        if selected\_vehicle != "All":

            filtered\_data = filtered\_data[filtered\_data[vehicle\_col] == selected\_vehicle]

    else:

        st.sidebar.warning(f"Vehicle Type column is missing.")

    # Filter 2: Priority Level

    priority\_col = 'Priority\_Levels'

    if priority\_col in filtered\_data.columns:

        priority\_options = ["All"] + sorted(list(filtered\_data[priority\_col].unique()))

        selected\_priority = st.sidebar.selectbox("Filter by Order Priority", options=priority\_options)

        if selected\_priority != "All":

            filtered\_data = filtered\_data[filtered\_data[priority\_col] == selected\_priority]

    else:

         st.sidebar.warning(f"Priority Levels column is missing.")

    # --- KEY METRICS (KPIs) ---

    col1, col2, col3, col4 = st.columns(4)

    # All KPI calculations use the Title Cased column names

    total\_co2\_mt = filtered\_data['Total\_CO2\_kg'].sum() / 1000 # Metric Tonnes

    avg\_ccpv = filtered\_data[filtered\_data['Carbon\_Cost\_Per\_Value'] > 0]['Carbon\_Cost\_Per\_Value'].mean()

    total\_fuel\_cost = filtered\_data['Fuel\_Labor\_Maintenance\_Costs\_INR'].sum()

    total\_orders = len(filtered\_data['ID'].unique())

    col1.metric("Total CO₂ (Metric Tonnes)", f"{total\_co2\_mt:,.2f} MT")

    col2.metric("Avg. Carbon Cost Per Value (CCPV)", f"{avg\_ccpv:,.5f}", help="CO₂ (kg) spent per unit of Order Value. Lower is better.")

    col3.metric("Total Routes Analyzed", f"{len(filtered\_data['Route\_ID'].unique()):,}")

    col4.metric("Total Fuel/Labor Cost", f"INR {total\_fuel\_cost:,.0f}")

    st.markdown("---")

    # --- VISUALIZATION SECTION (4 CHART TYPES) ---

    # VIZ 1: Bar Chart (CO2 Hotspots)

    st.header("1. CO₂ Hotspot Analysis: Top 10 Routes by Emission (Bar Chart) 📊")

    route\_co2\_analysis = filtered\_data.groupby('Route\_ID')['Total\_CO2\_kg'].sum().nlargest(10).reset\_index()

    fig1 = px.bar(route\_co2\_analysis, x='Route\_ID', y='Total\_CO2\_kg',

                   title="Highest CO₂ Emitting Routes",

                   labels={'Total\_CO2\_kg': 'CO₂ Emissions (kg)'},

                   color='Route\_ID',

                   color\_discrete\_sequence=px.colors.qualitative.Dark24)

    st.plotly\_chart(fig1, use\_container\_width=True)

    col\_chart\_2, col\_chart\_3 = st.columns(2)

    with col\_chart\_2:

        # VIZ 2: Scatter Plot (Efficiency vs. Vehicle Age)

        st.subheader("2. Fleet Asset Performance (Scatter Plot) ⚙️")

        fleet\_summary = filtered\_data.groupby('Vehicle\_Type').agg(

            Avg\_CCPV=('Carbon\_Cost\_Per\_Value', 'mean'),

            Avg\_Age=('Vehicle\_Age', 'mean'),

            Total\_CO2=('Total\_CO2\_kg', 'sum')

        ).reset\_index()

        fig2 = px.scatter(fleet\_summary, x='Avg\_Age', y='Avg\_CCPV',

                           size='Total\_CO2', color='Vehicle\_Type',

                           hover\_name='Vehicle\_Type',

                           title="Avg. CCPV vs. Vehicle Age (Bubble Size = Total CO₂)",

                           labels={'Avg\_CCPV': 'Avg. CCPV (Lower is Better)', 'Avg\_Age': 'Avg. Vehicle Age (Years)'},

                           color\_discrete\_sequence=px.colors.qualitative.Safe)

        st.plotly\_chart(fig2, use\_container\_width=True)

    with col\_chart\_3:

        # VIZ 3: Pie Chart (CO2 Distribution by Origin)

        st.subheader("3. CO₂ Distribution by Order Origin (Pie Chart) 🥧")

        origins\_col = 'Origins'

        if origins\_col in filtered\_data.columns:

            origin\_co2 = filtered\_data.groupby(origins\_col)['Total\_CO2\_kg'].sum().reset\_index()

            fig3 = px.pie(origin\_co2, values='Total\_CO2\_kg', names=origins\_col,

                        title="CO₂ Share by Order Origin Warehouse",

                        color\_discrete\_sequence=px.colors.sequential.Agsunset)

            st.plotly\_chart(fig3, use\_container\_width=True)

        else:

            st.warning("Cannot display Pie Chart: 'Origins' column is missing or was filtered out.")

    # VIZ 4: Line Chart (CO2 over Time - Using the 'Order\_Date' column)

    st.header("4. CO₂ Emission Trends Over Time (Line Chart) 📈")

    date\_column = 'Order\_Date'

    if date\_column in filtered\_data.columns:

        # Safely convert to datetime and then period

        filtered\_data['Date\_Key'] = pd.to\_datetime(filtered\_data[date\_column], errors='coerce').dt.to\_period('D')

        # Drop NaTs that resulted from coercion to avoid errors in groupby

        time\_co2 = filtered\_data.dropna(subset=['Date\_Key']).groupby('Date\_Key')['Total\_CO2\_kg'].sum().reset\_index()

        time\_co2['Date\_Key'] = time\_co2['Date\_Key'].astype(str)

        fig4 = px.line(time\_co2, x='Date\_Key', y='Total\_CO2\_kg',

                       title="Daily Total CO₂ Emissions Over Time",

                       labels={'Date\_Key': 'Date', 'Total\_CO2\_kg': 'CO₂ Emissions (kg)'})

        st.plotly\_chart(fig4, use\_container\_width=True)

    else:

        # This warning is what we are explicitly fixing by adding 'order\_dat' to the rename map.

        st.warning(f"Cannot display time series data: Date column '{date\_column}' not found.")

    # --- ACTIONABLE RECOMMENDATIONS (Business Impact) ---

    st.header("5. 💡 Actionable Recommendations & Business Impact")

    # Use the 'data' df for global recommendations

    full\_route\_co2 = data.groupby('Route\_ID')['Total\_CO2\_kg'].sum().nlargest(5).index.tolist()

    full\_fleet\_summary = data.groupby('Vehicle\_Type').agg(

        Avg\_CCPV=('Carbon\_Cost\_Per\_Value', 'mean')

    ).reset\_index()

    # Safely get the top 3 least efficient vehicles

    least\_efficient\_vehicles = full\_fleet\_summary.nlargest(3, 'Avg\_CCPV')['Vehicle\_Type'].tolist()

    st.markdown(f"""

    <div style="background-color:#ffe0e0; padding:15px; border-radius:10px; border-left: 5px solid red;">

    <h4>🔴 PRIORITY 1: High-Emission Routes (Quick Wins)</h4>

    \*\*Action:\*\* Immediately review and re-optimize the following top 5 global routes: \*\*{', '.join(full\_route\_co2)}\*\*.

    <br>

    \*\*Quantified Impact:\*\* These routes offer the highest immediate potential for \*\*fuel cost and CO₂ reduction\*\* through consolidation or greener vehicle assignment.

    </div>

    <br>

    <div style="background-color:#fff3cd; padding:15px; border-radius:10px; border-left: 5px solid orange;">

    <h4>🟠 PRIORITY 2: Inefficient Assets (Long-term Strategy)</h4>

    \*\*Action:\*\* Plan maintenance review or retirement for vehicle types: \*\*{', '.join(least\_efficient\_vehicles)}\*\*.

    <br>

    \*\*Quantified Impact:\*\* These vehicles have the highest \*\*Carbon Cost Per Value (CCPV)\*\*, linking operational inefficiency directly to lost revenue potential.

    </div>

    """, unsafe\_allow\_html=True)

    # --- Download/Export Functionality (Technical Requirement) ---

    st.markdown("---")

    csv = filtered\_data.to\_csv(index=False).encode('utf-8')

    st.download\_button(

        label="⬇️ Download Filtered Data as CSV (Technical Deliverable)",

        data=csv,

        file\_name='greenroute\_analysis.csv',

        mime='text/csv',

    )

**Appendix B: System Requirements - requirements.txt**

This appendix lists the exact Python dependencies required to replicate the application environment.

# requirements.txt

pandas

streamlit

plotly-express

numpy