```
"""
Smart Route Planner - NexGen Logistics Innovation Challenge
An intelligent routing system that optimizes for cost, time, and environmental impact
"""

import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')

# Page configuration
st.set_page_config(
    page_title="Smart Route Planner - NexGen Logistics",
    page_icon="■",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Custom CSS for better styling
st.markdown("""
    <style>
    .main-header {
        font-size: 3rem;
        font-weight: bold;
        color: #1f77b4;
        text-align: center;
        margin-bottom: 1rem;
    }
    .sub-header {
        font-size: 1.5rem;
        color: #ff7f0e;
        margin-top: 2rem;
        margin-bottom: 1rem;
    }
    .metric-card {
        background-color: #f0f2f6;
        padding: 1rem;
        border-radius: 0.5rem;
        border-left: 4px solid #1f77b4;
    }
    .insight-box {
        background-color: #e8f4f8;
        padding: 1rem;
        border-radius: 0.5rem;
        border-left: 4px solid #2ca02c;
        margin: 1rem 0;
    }
    </style>
""", unsafe_allow_html=True)

# Load data
@st.cache_data
def load_data():
    """Load and prepare route data"""
    try:
        df = pd.read_csv('routes_data.csv')
    except:
        st.error("Error loading data file. Please ensure routes_data.csv is in the same
directory.")
        return None

    # Handle missing values
    df['Weather_Impact'] = df['Weather_Impact'].fillna('None')

    # Calculate derived metrics
    FUEL_PRICE_PER_LITER = 102.0  # INR
    CO2_PER_LITER = 2.68  # kg CO2 per liter of fuel

    df['Fuel_Cost_INR'] = df['Fuel_Consumption_L'] * FUEL_PRICE_PER_LITER
    df['Total_Cost_INR'] = df['Fuel_Cost_INR'] + df['Toll_Charges_INR']
    df['CO2_Emissions_KG'] = df['Fuel_Consumption_L'] * CO2_PER_LITER
    df['Total_Time_Hours'] = (df['Distance_KM'] / 60) + (df['Traffic_Delay_Minutes'] / 60)
    df['Efficiency_Score'] = 100 - ((df['Total_Cost_INR'] / df['Total_Cost_INR'].max() * 30)
+
                                     (df['Total_Time_Hours'] / df['Total_Time_Hours'].max() *
30) +
                                     (df['CO2_Emissions_KG'] / df['CO2_Emissions_KG'].max() *
40))

    # Extract origin and destination
    df[['Origin', 'Destination']] = df['Route'].str.split('-', expand=True)

    # Categorize routes
    df['Route_Type'] = df['Destination'].apply(
        lambda x: 'International' if x in ['Dubai', 'Singapore', 'Hong Kong', 'Bangkok'] else
'Domestic'
    )

    return df

# Calculate optimization scores
def calculate_optimization_scores(df_filtered):
    """Calculate optimization scores for different priorities"""
    scores = {}

    # Cost Optimization (minimize total cost)
    df_filtered['Cost_Score'] = 100 - (df_filtered['Total_Cost_INR'] /
df_filtered['Total_Cost_INR'].max() * 100)

    # Time Optimization (minimize time)
    df_filtered['Time_Score'] = 100 - (df_filtered['Total_Time_Hours'] /
df_filtered['Total_Time_Hours'].max() * 100)

    # Environmental Optimization (minimize emissions)
    df_filtered['Eco_Score'] = 100 - (df_filtered['CO2_Emissions_KG'] /
df_filtered['CO2_Emissions_KG'].max() * 100)

    # Balanced Score (equal weights)
    df_filtered['Balanced_Score'] = (df_filtered['Cost_Score'] + df_filtered['Time_Score'] +
df_filtered['Eco_Score']) / 3

    return df_filtered

# Main app
def main():
    # Header
    st.markdown('<h1 class="main-header">■ Smart Route Planner</h1>', unsafe_allow_html=True)
    st.markdown('<p style="text-align: center; font-size: 1.2rem; color: #666;">NexGen
Logistics Innovation Challenge - Optimizing Routes for Cost, Time & Environment</p>',
unsafe_allow_html=True)
    st.markdown("---")

    # Load data
    df = load_data()
    if df is None:
        return

    # Sidebar filters
    st.sidebar.header("■ Filter Options")

    # Route type filter
    route_types = ['All'] + list(df['Route_Type'].unique())
    selected_route_type = st.sidebar.selectbox("Route Type", route_types)

    # Origin filter
    origins = ['All'] + sorted(df['Origin'].unique().tolist())
    selected_origin = st.sidebar.selectbox("Origin City", origins)

    # Destination filter
    destinations = ['All'] + sorted(df['Destination'].unique().tolist())
    selected_destination = st.sidebar.selectbox("Destination City", destinations)

    # Weather filter
    weather_conditions = ['All'] + sorted([x for x in df['Weather_Impact'].unique() if
pd.notna(x)])
    selected_weather = st.sidebar.selectbox("Weather Condition", weather_conditions)

    # Distance range
    st.sidebar.subheader("Distance Range (KM)")
    distance_range = st.sidebar.slider(
        "Select Range",
        float(df['Distance_KM'].min()),
        float(df['Distance_KM'].max()),
        (float(df['Distance_KM'].min()), float(df['Distance_KM'].max()))
    )

    # Optimization priority
    st.sidebar.markdown("---")
    st.sidebar.subheader("■■ Optimization Priority")
    optimization_priority = st.sidebar.radio(
        "Choose your priority:",
        ["Balanced", "Cost", "Time", "Environmental"]
    )

    # Apply filters
    df_filtered = df.copy()

    if selected_route_type != 'All':
        df_filtered = df_filtered[df_filtered['Route_Type'] == selected_route_type]

    if selected_origin != 'All':
        df_filtered = df_filtered[df_filtered['Origin'] == selected_origin]

    if selected_destination != 'All':
        df_filtered = df_filtered[df_filtered['Destination'] == selected_destination]

    if selected_weather != 'All':
        df_filtered = df_filtered[df_filtered['Weather_Impact'] == selected_weather]

    df_filtered = df_filtered[
        (df_filtered['Distance_KM'] >= distance_range[0]) &
        (df_filtered['Distance_KM'] <= distance_range[1])
    ]

    # Calculate optimization scores
    df_filtered = calculate_optimization_scores(df_filtered)

    # Display results count
    st.sidebar.markdown("---")
    st.sidebar.metric("■ Routes Found", len(df_filtered))

    # Export functionality
    if len(df_filtered) > 0:
        csv = df_filtered.to_csv(index=False)
        st.sidebar.download_button(
            label="■ Download Filtered Data",
            data=csv,
            file_name="filtered_routes.csv",
            mime="text/csv"
        )

    # Main content
    if len(df_filtered) == 0:
        st.warning("■■ No routes match your filter criteria. Please adjust the filters.")
        return

    # Key metrics
    st.markdown('<h2 class="sub-header">■ Key Performance Metrics</h2>',
unsafe_allow_html=True)

    col1, col2, col3, col4, col5 = st.columns(5)

    with col1:
        total_distance = df_filtered['Distance_KM'].sum()
        st.metric("Total Distance", f"{total_distance:,.0f} KM")

    with col2:
        total_cost = df_filtered['Total_Cost_INR'].sum()
        st.metric("Total Cost", f"■{total_cost:,.0f}")

    with col3:
        total_emissions = df_filtered['CO2_Emissions_KG'].sum()
        st.metric("CO■ Emissions", f"{total_emissions:,.0f} KG")

    with col4:
        total_time = df_filtered['Total_Time_Hours'].sum()
        st.metric("Total Time", f"{total_time:,.1f} hrs")

    with col5:
        avg_efficiency = df_filtered['Efficiency_Score'].mean()
        st.metric("Avg Efficiency", f"{avg_efficiency:.1f}%")

    st.markdown("---")

    # Optimization recommendations
    st.markdown('<h2 class="sub-header">■ Optimization Recommendations</h2>',
unsafe_allow_html=True)

    # Get best routes based on priority
    if optimization_priority == "Cost":
        best_routes = df_filtered.nsmallest(5, 'Total_Cost_INR')
        metric_col = 'Total_Cost_INR'
        metric_name = 'Cost'
        metric_format = '■{:,.2f}'
    elif optimization_priority == "Time":
        best_routes = df_filtered.nsmallest(5, 'Total_Time_Hours')
        metric_col = 'Total_Time_Hours'
        metric_name = 'Time'
        metric_format = '{:.2f} hrs'
    elif optimization_priority == "Environmental":
        best_routes = df_filtered.nsmallest(5, 'CO2_Emissions_KG')
        metric_col = 'CO2_Emissions_KG'
        metric_name = 'CO■'
        metric_format = '{:.2f} kg'
    else:  # Balanced
        best_routes = df_filtered.nlargest(5, 'Balanced_Score')
        metric_col = 'Balanced_Score'
        metric_name = 'Score'
        metric_format = '{:.1f}%'
    col1, col2 = st.columns([1, 1])

    with col1:
        st.markdown(f"### ■ Top 5 Routes ({optimization_priority} Optimized)")
        for idx, row in best_routes.iterrows():
            with st.expander(f"**{row['Route']}** - {metric_format.format(row[metric_col])}"):
                st.write(f"**Order ID:** {row['Order_ID']}")
                st.write(f"**Distance:** {row['Distance_KM']:.2f} KM")
                st.write(f"**Total Cost:** ■{row['Total_Cost_INR']:.2f}")
                st.write(f"**Time:** {row['Total_Time_Hours']:.2f} hours")
                st.write(f"**CO■ Emissions:** {row['CO2_Emissions_KG']:.2f} kg")
                st.write(f"**Weather:** {row['Weather_Impact']}")
                st.write(f"**Efficiency Score:** {row['Efficiency_Score']:.1f}%")

    with col2:
        st.markdown("### ■ Insights & Recommendations")

        # Calculate insights
        avg_cost = df_filtered['Total_Cost_INR'].mean()
        avg_time = df_filtered['Total_Time_Hours'].mean()
        avg_emissions = df_filtered['CO2_Emissions_KG'].mean()

        # Weather impact analysis
        weather_issues = len(df_filtered[df_filtered['Weather_Impact'] != 'None'])
        weather_pct = (weather_issues / len(df_filtered)) * 100 if len(df_filtered) > 0 else 0

        st.markdown(f"""
        <div class="insight-box">
        <b>■ Cost Analysis:</b><br>
        Average cost per route: ■{avg_cost:,.2f}<br>
        Best route saves: ■{(avg_cost - best_routes.iloc[0]['Total_Cost_INR']):,.2f}
        </div>
        """, unsafe_allow_html=True)

        st.markdown(f"""
        <div class="insight-box">
        <b>■■ Time Analysis:</b><br>
        Average time per route: {avg_time:.2f} hours<br>
        Traffic delays affecting: {len(df_filtered[df_filtered['Traffic_Delay_Minutes'] >
30])} routes
        </div>
        """, unsafe_allow_html=True)

        st.markdown(f"""
        <div class="insight-box">
        <b>■ Environmental Impact:</b><br>
        Average CO■ per route: {avg_emissions:.2f} kg<br>
        Weather impacting: {weather_pct:.1f}% of routes
        </div>
        """, unsafe_allow_html=True)

    st.markdown("---")

    # Visualizations
    st.markdown('<h2 class="sub-header">■ Data Visualizations</h2>', unsafe_allow_html=True)

    tab1, tab2, tab3, tab4 = st.tabs(["■ Overview", "■■ Route Analysis", "■ Performance", "■
Comparison"])

    with tab1:
        col1, col2 = st.columns(2)

        with col1:
            # Distance distribution
            fig1 = px.histogram(
                df_filtered,
                x='Distance_KM',
                nbins=30,
                title='Distribution of Route Distances',
                labels={'Distance_KM': 'Distance (KM)', 'count': 'Number of Routes'},
                color_discrete_sequence=['#1f77b4']
            )
            fig1.update_layout(showlegend=False)
            st.plotly_chart(fig1, use_container_width=True)

        with col2:
            # Route type breakdown
            route_type_counts = df_filtered['Route_Type'].value_counts()
            fig2 = px.pie(
                values=route_type_counts.values,
                names=route_type_counts.index,
                title='Domestic vs International Routes',
                color_discrete_sequence=px.colors.qualitative.Set2
            )
            st.plotly_chart(fig2, use_container_width=True)

        col1, col2 = st.columns(2)

        with col1:
            # Weather impact
            weather_counts = df_filtered['Weather_Impact'].value_counts()
            fig3 = px.bar(
                x=weather_counts.index,
                y=weather_counts.values,
                title='Routes by Weather Condition',
                labels={'x': 'Weather Condition', 'y': 'Number of Routes'},
                color=weather_counts.values,
                color_continuous_scale='Blues'
            )
            st.plotly_chart(fig3, use_container_width=True)

        with col2:
            # Cost breakdown
            cost_data = pd.DataFrame({
                'Category': ['Fuel Cost', 'Toll Charges'],
                'Amount': [df_filtered['Fuel_Cost_INR'].sum(),
df_filtered['Toll_Charges_INR'].sum()]
            })
            fig4 = px.pie(
                cost_data,
                values='Amount',
                names='Category',
                title='Total Cost Breakdown',
                color_discrete_sequence=['#ff7f0e', '#2ca02c']
            )
            st.plotly_chart(fig4, use_container_width=True)

    with tab2:
        col1, col2 = st.columns(2)

        with col1:
            # Top origin cities
            origin_counts = df_filtered['Origin'].value_counts().head(10)
            fig5 = px.bar(
                x=origin_counts.values,
                y=origin_counts.index,
                orientation='h',
                title='Top 10 Origin Cities',
                labels={'x': 'Number of Routes', 'y': 'City'},
                color=origin_counts.values,
                color_continuous_scale='Viridis'
            )
            st.plotly_chart(fig5, use_container_width=True)

        with col2:
            # Top destination cities
            dest_counts = df_filtered['Destination'].value_counts().head(10)
            fig6 = px.bar(
                x=dest_counts.values,
                y=dest_counts.index,
                orientation='h',
                title='Top 10 Destination Cities',
                labels={'x': 'Number of Routes', 'y': 'City'},
                color=dest_counts.values,
                color_continuous_scale='Plasma'
            )
            st.plotly_chart(fig6, use_container_width=True)

        # Traffic delay analysis
        fig7 = px.scatter(
            df_filtered,
            x='Distance_KM',
            y='Traffic_Delay_Minutes',
            color='Weather_Impact',
            size='Total_Cost_INR',
            hover_data=['Route', 'Total_Cost_INR'],
            title='Distance vs Traffic Delay (sized by Total Cost)',
            labels={'Distance_KM': 'Distance (KM)', 'Traffic_Delay_Minutes': 'Traffic Delay
(Minutes)'}
        )
        st.plotly_chart(fig7, use_container_width=True)

    with tab3:
        # Efficiency score distribution
        fig8 = px.box(
            df_filtered,
            x='Route_Type',
            y='Efficiency_Score',
            color='Route_Type',
            title='Efficiency Score by Route Type',
            labels={'Efficiency_Score': 'Efficiency Score (%)', 'Route_Type': 'Route Type'}
        )
        st.plotly_chart(fig8, use_container_width=True)

        col1, col2 = st.columns(2)

        with col1:
            # Fuel consumption vs distance
            fig9 = px.scatter(
                df_filtered,
                x='Distance_KM',
                y='Fuel_Consumption_L',
                color='Route_Type',
                title='Fuel Consumption vs Distance',
                labels={'Distance_KM': 'Distance (KM)', 'Fuel_Consumption_L': 'Fuel
Consumption (L)'}
            )
            st.plotly_chart(fig9, use_container_width=True)

        with col2:
            # CO2 emissions by route
            top_emitters = df_filtered.nlargest(10, 'CO2_Emissions_KG')
            fig10 = px.bar(
                top_emitters,
                x='CO2_Emissions_KG',
                y='Route',
                orientation='h',
                title='Top 10 Routes by CO■ Emissions',
                labels={'CO2_Emissions_KG': 'CO■ Emissions (KG)', 'Route': 'Route'},
                color='CO2_Emissions_KG',
                color_continuous_scale='Reds'
            )
            st.plotly_chart(fig10, use_container_width=True)

    with tab4:
        # Multi-metric comparison
        st.markdown("### ■ Multi-Metric Route Comparison")

        # Select routes to compare
        available_routes = df_filtered['Route'].unique().tolist()
        if len(available_routes) > 0:
            selected_routes = st.multiselect(
                "Select routes to compare (max 5):",
                available_routes,
                default=available_routes[:min(3, len(available_routes))],
                max_selections=5
            )

            if selected_routes:
                comparison_df = df_filtered[df_filtered['Route'].isin(selected_routes)]

                # Radar chart
                categories = ['Cost_Score', 'Time_Score', 'Eco_Score']

                fig11 = go.Figure()

                for route in selected_routes:
                    route_data = comparison_df[comparison_df['Route'] == route].iloc[0]
                    fig11.add_trace(go.Scatterpolar(
                        r=[route_data['Cost_Score'], route_data['Time_Score'], 
route_data['Eco_Score']],
                        theta=['Cost Efficiency', 'Time Efficiency', 'Environmental'],
                        fill='toself',
                        name=route
                    ))

                fig11.update_layout(
                    polar=dict(radialaxis=dict(visible=True, range=[0, 100])),
                    showlegend=True,
                    title='Route Performance Comparison'
                )
                st.plotly_chart(fig11, use_container_width=True)

                # Comparison table
                st.markdown("### ■ Detailed Comparison")
                comparison_table = comparison_df[[
                    'Route', 'Distance_KM', 'Total_Cost_INR', 'Total_Time_Hours',
                    'CO2_Emissions_KG', 'Traffic_Delay_Minutes', 'Weather_Impact', 
'Efficiency_Score'
                ]].copy()

                comparison_table.columns = ['Route', 'Distance (KM)', 'Total Cost (■)', 'Time 
(hrs)',
                                           'CO■ (kg)', 'Traffic Delay (min)', 'Weather', 
'Efficiency (%)']

                st.dataframe(
                    comparison_table.style.format({
                        'Distance (KM)': '{:.2f}',
                        'Total Cost (■)': '■{:,.2f}',
                        'Time (hrs)': '{:.2f}',
                        'CO■ (kg)': '{:.2f}',
                        'Efficiency (%)': '{:.1f}%'
                    }),
                    use_container_width=True
                )

    st.markdown("---")

    # Detailed data table
    st.markdown('<h2 class="sub-header">■ Detailed Route Data</h2>', unsafe_allow_html=True)

    # Sort options
    sort_by = st.selectbox(
        "Sort by:",
        ['Efficiency_Score', 'Total_Cost_INR', 'Total_Time_Hours', 'CO2_Emissions_KG',
'Distance_KM'],
        format_func=lambda x: {
            'Efficiency_Score': 'Efficiency Score',
            'Total_Cost_INR': 'Total Cost',
            'Total_Time_Hours': 'Total Time',
            'CO2_Emissions_KG': 'CO■ Emissions',
            'Distance_KM': 'Distance'
        }[x]
    )

    sort_order = st.radio("Order:", ['Descending', 'Ascending'], horizontal=True)

    df_display = df_filtered.sort_values(by=sort_by, ascending=(sort_order == 'Ascending'))

    display_cols = [
        'Order_ID', 'Route', 'Distance_KM', 'Fuel_Cost_INR', 'Toll_Charges_INR',
        'Total_Cost_INR', 'Total_Time_Hours', 'CO2_Emissions_KG', 'Traffic_Delay_Minutes',
        'Weather_Impact', 'Efficiency_Score'
    ]

    st.dataframe(
        df_display[display_cols].style.format({
            'Distance_KM': '{:.2f}',
            'Fuel_Cost_INR': '■{:,.2f}',
            'Toll_Charges_INR': '■{:,.2f}',
            'Total_Cost_INR': '■{:,.2f}',
            'Total_Time_Hours': '{:.2f}',
            'CO2_Emissions_KG': '{:.2f}',
            'Efficiency_Score': '{:.1f}%'
        }),
        use_container_width=True,
        height=400
    )

    # Footer
    st.markdown("---")
    st.markdown("""
        <div style="text-align: center; color: #666; padding: 2rem;">
            <p><b>Smart Route Planner</b> | NexGen Logistics Innovation Challenge</p>
            <p>Optimizing logistics operations through data-driven insights</p>
        </div>
    """, unsafe_allow_html=True)

if __name__ == "__main__":
    main()
```