```
pip install googlemaps
```

```
Requirement already satisfied: googlemaps in /usr/local/lib/python3.11/dist-packages (4.10.0)
Requirement already satisfied: requests<3.0,>=2.20.0 in /usr/local/lib/python3.11/dist-packages (from googlemaps) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0,>=2.20.0->googlem
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0,>=2.20.0->googlemaps) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0,>=2.20.0->googlemaps) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0,>=2.20.0->googlemaps) (
```

```
pip install geopy
```

```
Requirement already satisfied: geopy in /usr/local/lib/python3.11/dist-packages (2.4.1)
Requirement already satisfied: geographiclib<3,>=1.52 in /usr/local/lib/python3.11/dist-packages (from geopy) (2.0)
```

```
# Before this next step make sure Google API is working correctly with Geocoding API in Enable Mode.
```

```
import googlemaps
from geopy.distance import geodesic

# Initialize Google Maps client with API key
gmaps = googlemaps.Client(key="AIzaSyCX1lL08h42xp7vq57_09fW84slw1B-g6o")

# Function to geocode city using Google Maps
def geocode_with_google(city):
    try:
        geocode_result = gmaps.geocode(city)
        if geocode_result:
            # Extract latitude and longitude from the geocode result
            return geocode_result[0]['geometry']['location']
    except Exception as e:
        print(f"Error geocoding {city}: {e}")
    return None

# List of cities in Gujarat
gujarat_cities = ["Ahmedabad", "Surat", "Vadodara", "Vapi", "Rajkot", "Bhavnagar","Gandhinagar", "Junagadh", "Kutch", "Anand", "Navsari", '

# Create the list of city pairs (unique pairs between two lists)
city_distance_dict = {}

# Loop through each city in gujarat_cities
for city1 in gujarat_cities:
    for city2 in gujarat_cities:
        if city1 != city2:  # Ensure the cities are not the same
            # Geocode the cities with Google Maps
            location1 = geocode_with_google(city1)
            location2 = geocode_with_google(city2)

            # If both cities were successfully geocoded, calculate the distance
            if location1 and location2:
                coords_1 = (location1['lat'], location1['lng'])
                coords_2 = (location2['lat'], location2['lng'])

                # Calculate the distance between the cities using geodesic
                distance = geodesic(coords_1, coords_2).kilometers

                # Add the result to the dictionary (key as city pair and value as distance)
                city_distance_dict[(city1, city2)] = distance

# Print out the dictionary
print(city_distance_dict)
```

```
{('Ahmedabad', 'Surat'): 206.8474725190747, ('Ahmedabad', 'Vadodara'): 103.2563681726481, ('Ahmedabad', 'Vapi'): 293.66890667884127, ('
```

```
import requests
import pandas as pd
import time

# Google Maps API Key
API_KEY = "AIzaSyCX1lL08h42xp7vq57_09fW84slw1B-g6o"
```

```
# List of cities in Gujarat
cities = [
    "Ahmedabad", "Surat", "Vadodara", "Vapi", "Rajkot", "Bhavnagar",
    "Gandhinagar", "Junagadh", "Kutch", "Anand", "Navsari", "Nadiad",
    "Patan", "Morbi", "Bharuch"
]

# Function to fetch route summaries between two cities
def get_routes(origin, dest, key):
    url = "https://maps.googleapis.com/maps/api/directions/json"
    params = {
        'origin': origin,
        'destination': dest,
        'alternatives': 'true',  # Get multiple route options
        'key': key
    }

    # Send GET request to the Directions API
    r = requests.get(url, params=params).json()

    # Extract and return route details
    return [{
        "Route ID": f"{origin}-{dest}-Route-{i+1}",        # Unique Route ID
        "Origin": origin,                                  # Start city
        "Destination": dest,                               # End city
        "Summary": route.get("summary", "")                # Route summary (e.g., highway names)
    } for i, route in enumerate(r.get("routes", []))]      # Loop over all route options

# Collect all routes between city pairs
all_routes = []
for c1 in cities:
    for c2 in cities:
        if c1 != c2:  # Skip same city pairs
            print(f"{c1} --> {c2}")  # Status print
            all_routes += get_routes(c1, c2, API_KEY)
            time.sleep(1)  # Delay to avoid hitting API rate limits

# Save collected routes to CSV
pd.DataFrame(all_routes).to_csv("gujarat_unique_routes.csv", index=False)
```

```
Navsari --> Rajkot
Navsari --> Rajkot
Navsari --> Bhavnagar
Navsari --> Gandhinagar
Navsari --> Junagadh
Navsari --> Kutch
Navsari --> Anand
Navsari --> Nadiad
Navsari --> Patan
Navsari --> Morbi
Navsari --> Bharuch
Nadiad --> Ahmedabad
Nadiad --> Surat
Nadiad --> Vadodara
Nadiad --> Vapi
Nadiad --> Rajkot
Nadiad --> Bhavnagar
Nadiad --> Gandhinagar
Nadiad --> Junagadh
Nadiad --> Kutch
Nadiad --> Anand
```

```
import pandas as pd
import random
import itertools
import os
```

```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/EDUNET COLAB IMPLEMENTATION/ICBP Project Data/gujarat_unique_routes.csv")
```

```
df.head(5)
```

	Route ID	Origin	Destination	Summary	Via
0	Ahmedabad-Surat-Route-1-NE 1 and NE 4	Ahmedabad	Surat	NE 1 and NE 4	Turn right onto Swami Vivekananda Rd<div style...
1	Ahmedabad-Vadodara-Route-1-NE 1	Ahmedabad	Vadodara	NE 1	Turn right onto Swami Vivekananda Rd<div style...
2	Ahmedabad-Vapi-Route-1-NE 4 and NH 48	Ahmedabad	Vapi	NE 4 and NH 48	Turn right onto Swami Vivekananda Rd<div style...
3	Ahmedabad-Rajkot-Route-1-NH 47	Ahmedabad	Rajkot	NH 47	Make a U-turn<div style="font-size:0.9em">Cont...
4	Ahmedabad-Rajkot-Route-2-GJ SH 17	Ahmedabad	Rajkot	GJ SH 17	Make a U-turn<div style="font-size:0.9em">Cont...

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Load the base route data from CSV
input_path = '/content/drive/MyDrive/Colab Notebooks/EDUNET COLAB IMPLEMENTATION/ICBP Project Data/gujarat_unique_routes.csv'
df_base = pd.read_csv(input_path)

# Define possible values for each varying condition
traffic_levels = ['Low', 'Medium', 'High']
weather_conditions = ['Clear', 'Rainy', 'Foggy', 'Summer', 'Storm']
route_types = ['Highway', 'Urban', 'Mixed']

# Constants
CO2_per_litre = 2.68  # CO2 emissions per litre of diesel (in kg)
fuel_efficiency_range = (2.5, 4.5)       # km per litre
cargo_weight_range = (2000, 20000)       # in kg
distance_range = (80, 600)               # Simulated route distances (in km)

# Prepare list to collect all synthetic route combinations
rows = []

# Generate all combinations of traffic, weather, and route types
condition_combinations = list(itertools.product(traffic_levels, weather_conditions, route_types))

# Iterate through each base route
for _, row in df_base.iterrows():
    origin = row['Origin']
    destination = row['Destination']
    route_id = row['Route ID']
    summary = row['Summary']

    # Simulate route distance
    base_distance = random.uniform(*distance_range)

    # Generate synthetic data for each combination of conditions
    for traffic, weather, route_type in condition_combinations:
      # Randomize cargo weight and base fuel efficiency
      cargo_weight = random.uniform(*cargo_weight_range)
      fuel_efficiency = random.uniform(*fuel_efficiency_range)

      # Efficiency adjustment factor based on traffic and weather
      efficiency_factor = 1.0
      if traffic == 'Medium':
          efficiency_factor *= 0.9
      elif traffic == 'High':
          efficiency_factor *= 0.75

      if weather in ['Rainy', 'Foggy', 'Storm']:
          efficiency_factor *= 0.85
      elif weather == 'Summer':
          efficiency_factor *= 0.95

      # Calculate adjusted fuel efficiency and CO₂ emission
      adjusted_efficiency = fuel_efficiency * efficiency_factor
      fuel_used = base_distance / adjusted_efficiency
      emission = fuel_used * CO2_per_litre

      # Append the final data row
      rows.append({
          'Route ID': route_id,
          'Origin': origin,
          'Destination': destination,
          'Route_Summary': summary,
          'Route_Distance_km': round(base_distance, 2),
          'Route_Type': route_type,
          'Traffic': traffic,
          'Weather': weather,
          'Cargo_Weight_kg': round(cargo_weight, 2),
          'Fuel_Efficiency_kmpl': round(fuel_efficiency, 2),
          'Adjusted_Efficiency': round(adjusted_efficiency, 2),
          'Fuel_Used_Litres': round(fuel_used, 2),
          'CO2_Emissions_kg': round(emission, 2)
      })


# Convert to DataFrame
df = pd.DataFrame(rows)

# Save the output CSV
output_path = '/content/drive/MyDrive/Colab Notebooks/EDUNET COLAB IMPLEMENTATION/ICBP Project Data/all_route_variants_with_emissions.csv'
os.makedirs(os.path.dirname(output_path), exist_ok=True)
df.to_csv(output_path, index=False)

print(f"CSV saved with {len(df)} rows and {len(df_base)} base routes × {len(condition_combinations)} condition combos.")
```

```
CSV saved with 17190 rows and 382 base routes × 45 condition combos.
```