# Import necessary libraries

import pandas as pd

# Mount Google Drive (if datasets are stored in Drive)

from google.colab import files

# Upload files

uploaded = files.upload()

# Load datasets

ev\_dataset = pd.read\_csv("EV\_Dataset.csv")

charging\_stations = pd.read\_csv("ev-charging-stations-india.csv")

ev\_sales = pd.read\_excel("Ev Sales.xlsx")

# Fix column names

charging\_stations.rename(columns={'lattitude': 'latitude'}, inplace=True)

# Convert Date column in EV\_Dataset to datetime

ev\_dataset['Date'] = pd.to\_datetime(ev\_dataset['Date'], errors='coerce')

# Convert latitude & longitude to numeric, fill missing with mean values

charging\_stations['latitude'] = pd.to\_numeric(charging\_stations['latitude'], errors='coerce')

charging\_stations['longitude'] = pd.to\_numeric(charging\_stations['longitude'], errors='coerce')

charging\_stations['latitude'].fillna(charging\_stations['latitude'].mean(), inplace=True)

charging\_stations['longitude'].fillna(charging\_stations['longitude'].mean(), inplace=True)

# Fill missing addresses with "Unknown"

charging\_stations['address'].fillna("Unknown", inplace=True)

# Fill missing charging station type with "Unknown"

charging\_stations['type'].fillna("Unknown", inplace=True)

# Check cleaned data

print("EV Dataset Sample:")

print(ev\_dataset.head())

print("\nCharging Stations Sample:")

print(charging\_stations.head())

print("\nEV Sales Sample:")

print(ev\_sales.head())

# Save cleaned datasets

ev\_dataset.to\_csv("Cleaned\_EV\_Dataset.csv", index=False)

charging\_stations.to\_csv("Cleaned\_EV\_Charging\_Stations.csv", index=False)

ev\_sales.to\_excel("Cleaned\_Ev\_Sales.xlsx", index=False)

print("\n✅ Data Cleaning Complete! Cleaned files saved.")

 Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving EV\_Dataset.csv to EV\_Dataset (1).csv

Saving ev-charging-stations-india.csv to ev-charging-stations-india (1).csv

Saving Ev Sales.xlsx to Ev Sales (1).xlsx

<ipython-input-1-c67043a8cdf1>:25: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

charging\_stations['latitude'].fillna(charging\_stations['latitude'].mean(), inplace=True)

<ipython-input-1-c67043a8cdf1>:26: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

charging\_stations['longitude'].fillna(charging\_stations['longitude'].mean(), inplace=True)

<ipython-input-1-c67043a8cdf1>:29: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

charging\_stations['address'].fillna("Unknown", inplace=True)

<ipython-input-1-c67043a8cdf1>:32: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

charging\_stations['type'].fillna("Unknown", inplace=True)

<ipython-input-1-c67043a8cdf1>:32: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'Unknown' has dtype incompatible with float64, please explicitly cast to a compatible dtype first.

charging\_stations['type'].fillna("Unknown", inplace=True)

EV Dataset Sample:

Year Month\_Name Date State Vehicle\_Class \

0 2014.0 jan 2014-01-01 Andhra Pradesh ADAPTED VEHICLE

1 2014.0 jan 2014-01-01 Andhra Pradesh AGRICULTURAL TRACTOR

2 2014.0 jan 2014-01-01 Andhra Pradesh AMBULANCE

3 2014.0 jan 2014-01-01 Andhra Pradesh ARTICULATED VEHICLE

4 2014.0 jan 2014-01-01 Andhra Pradesh BUS

Vehicle\_Category Vehicle\_Type EV\_Sales\_Quantity

0 Others Others 0.0

1 Others Others 0.0

2 Others Others 0.0

3 Others Others 0.0

4 Bus Bus 0.0

Charging Stations Sample:

name state city \

0 Neelkanth Star DC Charging Station Haryana Gurugram

1 Galleria DC Charging Station Haryana Gurugram

2 Highway Xpress (Jaipur-Delhi) DC charging station Rajasthan Behror

3 Food Carnival DC Charging Station Uttar Pradesh Khatauli

4 Food Carnival AC Charging Station Uttar Pradesh Khatauli

address latitude longitude \

0 Neelkanth Star Karnal, NH 44, Gharunda, Kutail... 29.6019 76.9803

1 DLF Phase IV, Sector 28, Gurugram, Haryana 122022 28.4673 77.0818

2 Jaipur to Delhi Road, Behror Midway, Behror, R... 27.8751 76.2760

3 Fun and Food Carnival, NH 58, Khatauli Bypass,... 29.3105 77.7218

4 NH 58, Khatauli Bypass, Bhainsi, Uttar Pradesh... 29.3105 77.7218

type

0 12.0

1 12.0

2 12.0

3 12.0

4 12.0

EV Sales Sample:

YEAR 2 W 3 W 4 W BUS TOTAL

0 2017-04-01 96 4748 198 0 5042

1 2017-05-01 91 6720 215 2 7028

2 2017-06-01 137 7178 149 1 7465

3 2017-07-01 116 8775 120 0 9011

4 2017-08-01 99 8905 137 0 9141

✅ Data Cleaning Complete! Cleaned files saved.

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load cleaned datasets

ev\_sales = pd.read\_excel("Cleaned\_Ev\_Sales.xlsx")

ev\_dataset = pd.read\_csv("Cleaned\_EV\_Dataset.csv")

charging\_stations = pd.read\_csv("Cleaned\_EV\_Charging\_Stations.csv")

# Strip spaces and lowercase all column names

ev\_sales.columns = ev\_sales.columns.str.strip().str.lower()

# Rename the column correctly

ev\_sales.rename(columns={'total': 'Total'}, inplace=True)

ev\_sales.rename(columns={'year': 'Year'}, inplace=True)

ev\_dataset.rename(columns={'Vehicle\_Type': 'Vehicle Type'}, inplace=True)

# Convert Date column to datetime if not already

ev\_dataset['Date'] = pd.to\_datetime(ev\_dataset['Date'])

### 🚗 EV Sales Trends Over Time ###

plt.figure(figsize=(10, 5))

sns.lineplot(x=ev\_sales["Year"], y=ev\_sales["Total"], marker='o', color="blue")

plt.xlabel("Year")

plt.ylabel("Total EV Sales")

plt.title("EV Sales Trend Over the Years in India")

plt.grid()

plt.show()

### 📍 Top 10 EV-Adopting States ###

top\_states = ev\_dataset.groupby("State")["Vehicle Type"].count().sort\_values(ascending=False).head(10)

plt.figure(figsize=(12, 6))

sns.barplot(x=top\_states.index, y=top\_states.values, palette="coolwarm")

plt.xlabel("State")

plt.ylabel("Number of EVs Registered")

plt.title("Top 10 States with Highest EV Adoption")

plt.xticks(rotation=45)

plt.show()

### 🔌 EV Charging Station Distribution ###

plt.figure(figsize=(12, 6))

sns.histplot(charging\_stations["state"], kde=False, color="green")

plt.xlabel("State")

plt.ylabel("Number of Charging Stations")

plt.title("EV Charging Station Distribution by State")

plt.xticks(rotation=45)

plt.show()

### 🗺️ Heatmap of Charging Stations (Geospatial) ###

import folium

from folium.plugins import HeatMap

# Create a map centered in India

india\_map = folium.Map(location=[20.5937, 78.9629], zoom\_start=5)

# Add heatmap

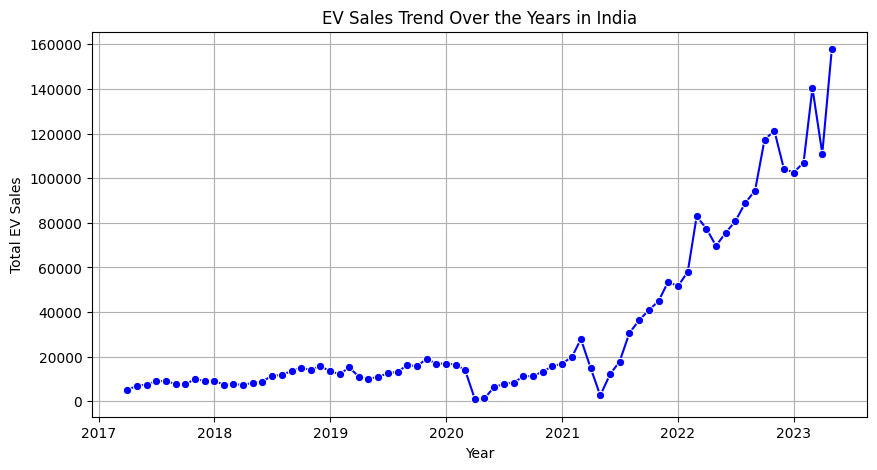
heat\_data = list(zip(charging\_stations["latitude"], charging\_stations["longitude"]))

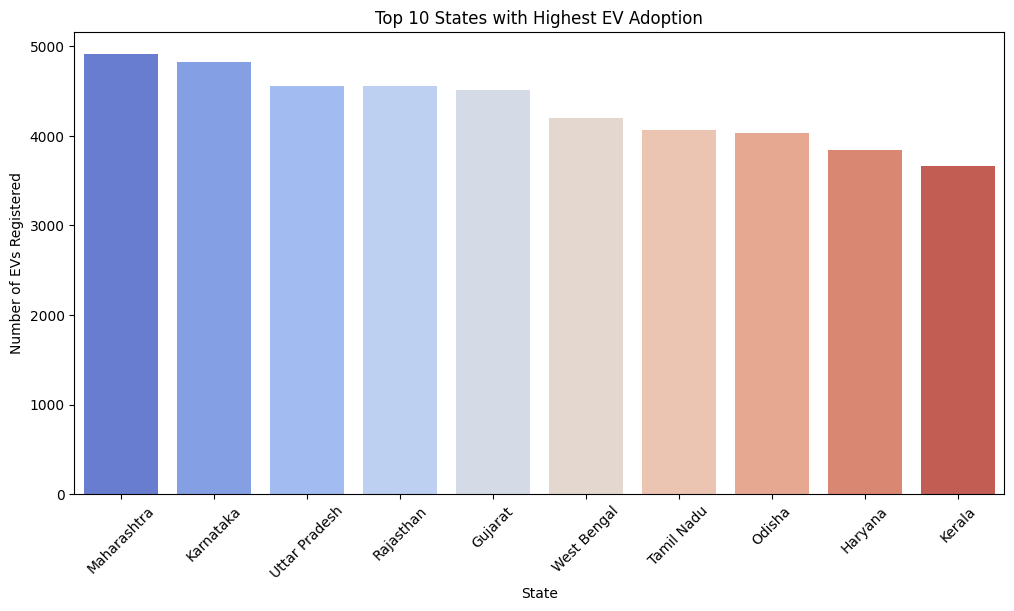
HeatMap(heat\_data).add\_to(india\_map)

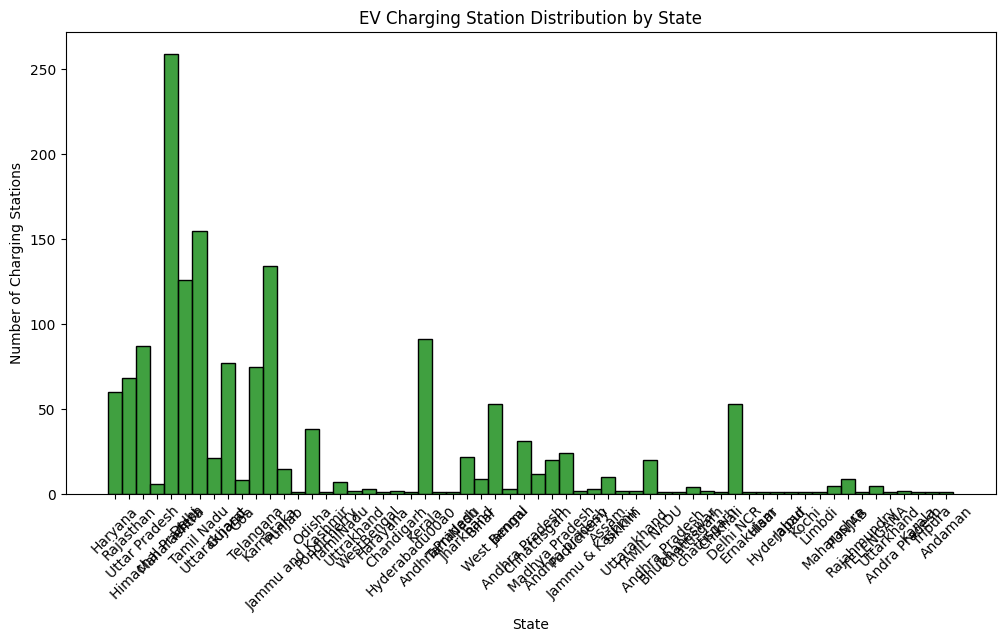
# Save map as HTML file

india\_map.save("EV\_Charging\_Heatmap.html")

print("✅ Heatmap saved as EV\_Charging\_Heatmap.html (Open in browser)")







# Install Prophet if not already installed

!pip install prophet

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

from prophet import Prophet

# Load cleaned EV Sales dataset

ev\_sales = pd.read\_excel("Cleaned\_Ev\_Sales.xlsx")

# Check the first few rows

print(ev\_sales.head())

# Ensure correct column names (adjust if needed)

ev\_sales.columns = ev\_sales.columns.str.strip().str.lower()

ev\_sales.rename(columns={'year': 'ds', 'total': 'y'}, inplace=True)

# Convert 'ds' column to datetime format

ev\_sales["ds"] = pd.to\_datetime(ev\_sales["ds"], format="%Y")

# Initialize and fit the Prophet model

model = Prophet()

model.fit(ev\_sales)

# Create future dates for the next 5 years

future = model.make\_future\_dataframe(periods=5, freq='Y')

# Predict future EV sales

forecast = model.predict(future)

# Plot forecast

plt.figure(figsize=(10, 5))

model.plot(forecast)

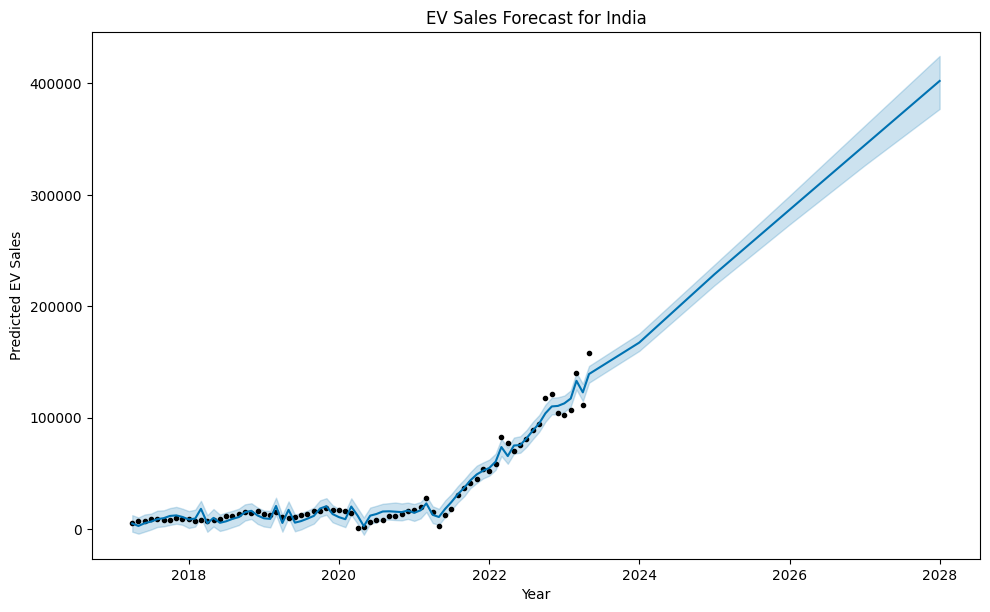
plt.title("EV Sales Forecast for India")

plt.xlabel("Year")

plt.ylabel("Predicted EV Sales")

plt.grid()

plt.show()



# Install required libraries

!pip install geopandas folium scikit-learn

# Import necessary libraries

import pandas as pd

import geopandas as gpd

import folium

from sklearn.cluster import KMeans

# Load the EV charging station dataset

ev\_charging = pd.read\_csv("/content/Cleaned\_EV\_Charging\_Stations.csv")

# Ensure correct column names

ev\_charging.columns = ev\_charging.columns.str.strip().str.lower()

# Extract latitude and longitude columns

ev\_charging = ev\_charging[['latitude', 'longitude']].dropna()

# Convert to numeric to avoid formatting issues

ev\_charging["latitude"] = pd.to\_numeric(ev\_charging["latitude"], errors="coerce")

ev\_charging["longitude"] = pd.to\_numeric(ev\_charging["longitude"], errors="coerce")

# Drop any remaining NaN values

ev\_charging = ev\_charging.dropna(subset=["latitude", "longitude"])

# Filter data to include only locations in India (approximate ranges)

ev\_charging = ev\_charging[

    (ev\_charging.latitude >= 6.5) & (ev\_charging.latitude <= 35.5) &

    (ev\_charging.longitude >= 68.0) & (ev\_charging.longitude <= 97.0)

]

# Convert to GeoDataFrame

gdf = gpd.GeoDataFrame(ev\_charging, geometry=gpd.points\_from\_xy(ev\_charging.longitude, ev\_charging.latitude))

# Create a folium map centered on India's mean coordinates

m = folium.Map(location=[gdf.latitude.mean(), gdf.longitude.mean()], zoom\_start=5)

# Add markers for each charging station

for \_, row in gdf.iterrows():

    folium.CircleMarker(

        [row.latitude, row.longitude],

        radius=3,

        color="blue",

        fill=True,

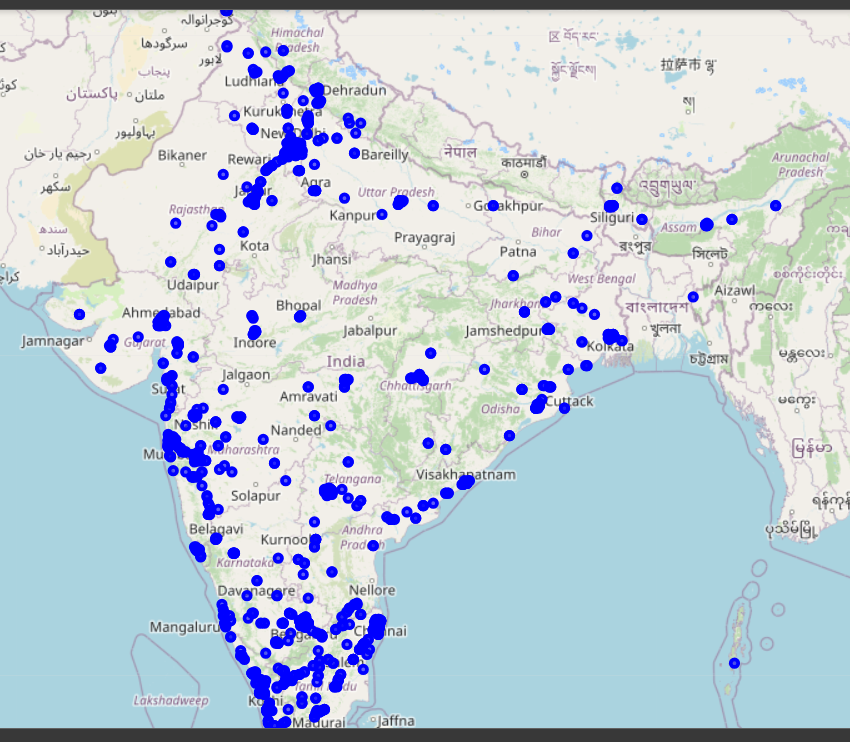
        fill\_color="blue",

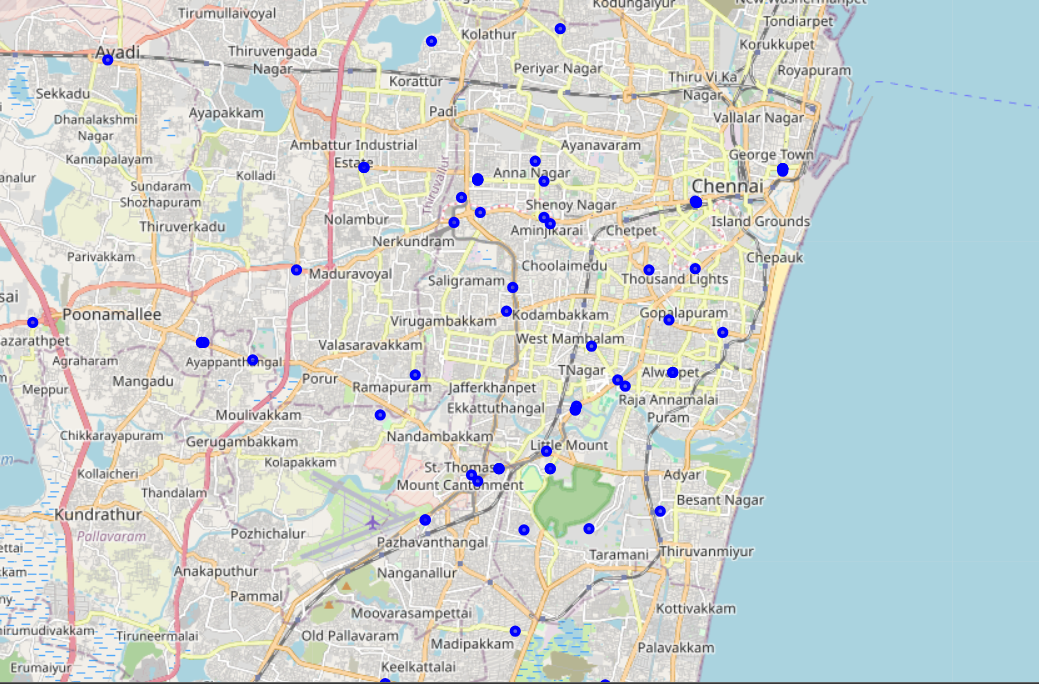
        fill\_opacity=0.6

    ).add\_to(m)

# Show the map

m





# Install required libraries

!pip install scikit-learn folium

# Import necessary libraries

import pandas as pd

import numpy as np

import folium

from sklearn.cluster import KMeans

# Load the cleaned EV charging station data

ev\_charging = pd.read\_csv("/content/Cleaned\_EV\_Charging\_Stations.csv")

# Ensure correct column names

ev\_charging.columns = ev\_charging.columns.str.strip().str.lower()

# Convert to numeric (handle errors properly)

ev\_charging["latitude"] = pd.to\_numeric(ev\_charging["latitude"], errors="coerce")

ev\_charging["longitude"] = pd.to\_numeric(ev\_charging["longitude"], errors="coerce")

# Drop invalid values

ev\_charging = ev\_charging.dropna(subset=["latitude", "longitude"])

# Check if coordinates are within India's range

ev\_charging = ev\_charging[(ev\_charging.latitude >= 6.5) & (ev\_charging.latitude <= 35.5)]

ev\_charging = ev\_charging[(ev\_charging.longitude >= 68.0) & (ev\_charging.longitude <= 97.0)]

# Apply K-Means clustering

k = 8  # Number of clusters (can be adjusted)

kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)

ev\_charging["cluster"] = kmeans.fit\_predict(ev\_charging[['latitude', 'longitude']])

# Get cluster centers (suggested new charging station locations)

cluster\_centers = kmeans.cluster\_centers\_

# Create a map centered on India

m = folium.Map(location=[ev\_charging.latitude.mean(), ev\_charging.longitude.mean()], zoom\_start=5)

# Add existing charging stations

for \_, row in ev\_charging.iterrows():

    folium.CircleMarker(

        [row.latitude, row.longitude],

        radius=3,

        color="blue",

        fill=True,

        fill\_color="blue",

        fill\_opacity=0.6

    ).add\_to(m)

# Add new suggested charging station locations (cluster centers)

for lat, lon in cluster\_centers:

    folium.Marker([lat, lon], popup="Suggested Charging Station", icon=folium.Icon(color="red")).add\_to(m)

# Show the map

m

