

```

import geopandas as gpd

url = "/content/ev_charging_stations.geojson"
stations = gpd.read_file(url)
stations.tail()

```

	address	city	availability	contact	isValidated	issue	lattitude	longitude	name	notes	phones	pincode
154	Noida authority parking, near B-14, sector 2 N...	Noida	1		1	0	28.5842756	77.3137535	EESL Authority Parking Sector 2 Noida			201301
155	Noida Authority parking, near A-12, sector 16,...	Noida	1		1	0	28.578614	77.317483	EESL Sector 16 Near Metro Station Noida			201301
156	In front of marie gold exports ltd, A block, s...	Noida	1		1	0	28.36374688	77.27384312	EESL Marie Gold Exports Block A Sector 64 Noida			201301
157	In front of Hexagon pvt ltd, sector-65, Noida	Noida	1		1	0	28.6132	77.3814	EESL Hexagon Sector 65 Noida			201301
158	The Mamlatdar Office, Goa	Goa	1		1	0	15.588	73.8139	EESL Mamlatdar Office Mapusa Goa			403406

```

stations.info()
stations.columns
stations.sample(5)

```

```

<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 159 entries, 0 to 158
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   address     159 non-null    object  
 1   city        159 non-null    object  
 2   availability 159 non-null    int32  
 3   contact     159 non-null    object  
 4   isValidated 159 non-null    object  
 5   issue       159 non-null    int32  
 6   latitude    159 non-null    object  
 7   longitude   159 non-null    object  
 8   name        159 non-null    object  
 9   notes       159 non-null    object  
 10  phones      159 non-null    object  
 11  pincode    159 non-null    object  
 12  pricing     159 non-null    object  
 13  state       159 non-null    object  
 14  stationid   159 non-null    object  
 15  type        159 non-null    object  
 16  geometry    159 non-null    geometry
dtypes: geometry(1), int32(2), object(14)
memory usage: 20.0+ KB

```

		address	city	availability	contact	isValidated	issue	lattitude	longitude	name	notes	phones	pinc
156		In front of marie gold exports ltd, A block, s...	Noida	1		1	0	28.36374688	77.27384312	EESL Marie Gold Exports Block A Sector 64 Noida			201
12		ECO Park, Gate No1, West Bengal: 700156	Kolkata	1		1	0	22.594	88.4729	EESL ECO Park Gate No 1 New Town			700
128		Khan Market, Rabindra Nagar, New Delhi- 110003	Delhi	1		1	0	28.600324	77.226883	EESL Khan Market			110
69		Susma Swaraj Bhawan, MEA, New Delhi	Delhi	1		1	0	28.5958333	77.1805556	EESL Sushma Swaraj Bhawan			110
89		Noida Authority Parking, near metro station, s...	Noida	1		1	0	28.578614	77.317483	EESL Sector 16 Near Metro Station Noida			201

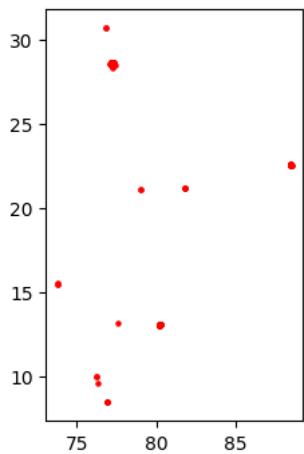
```
stations.isnull().any()
```

```
0
address False
city False
availability False
contact False
isValidated False
issue False
latitude False
longitude False
name False
notes False
phones False
pincode False
pricing False
state False
stationid False
type False
geometry False
```

dtype: bool

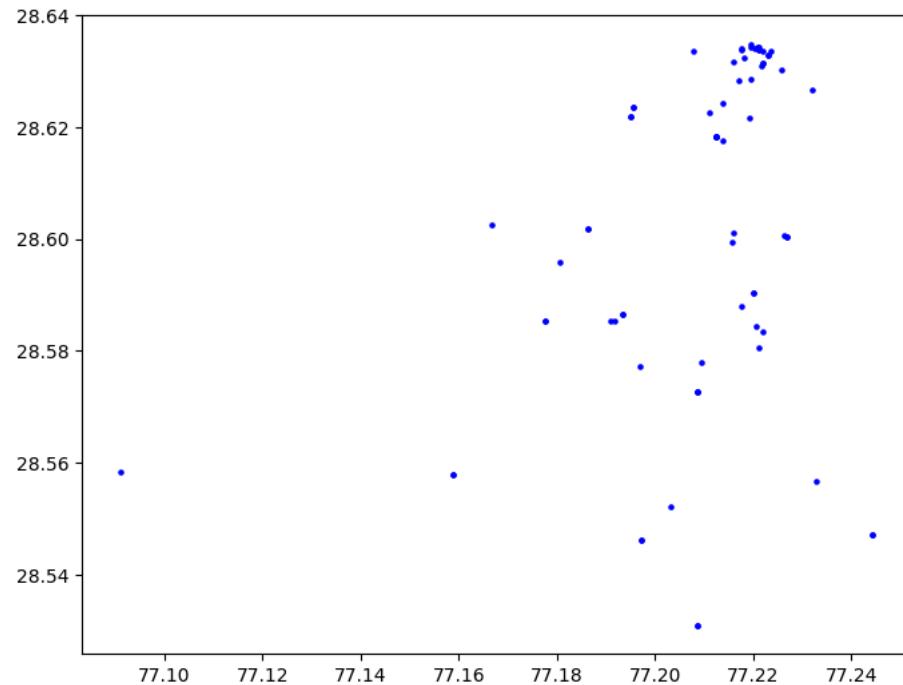
```
stations.plot(figsize=(4,4), color='red', markersize=5)
```

<Axes: >



```
city = "Delhi"
city_stations = stations[stations['address'].str.contains(city, case=False, na=False)]
city_stations.plot(figsize=(8,8), color='blue', markersize=5)
```

```
<Axes: >
```



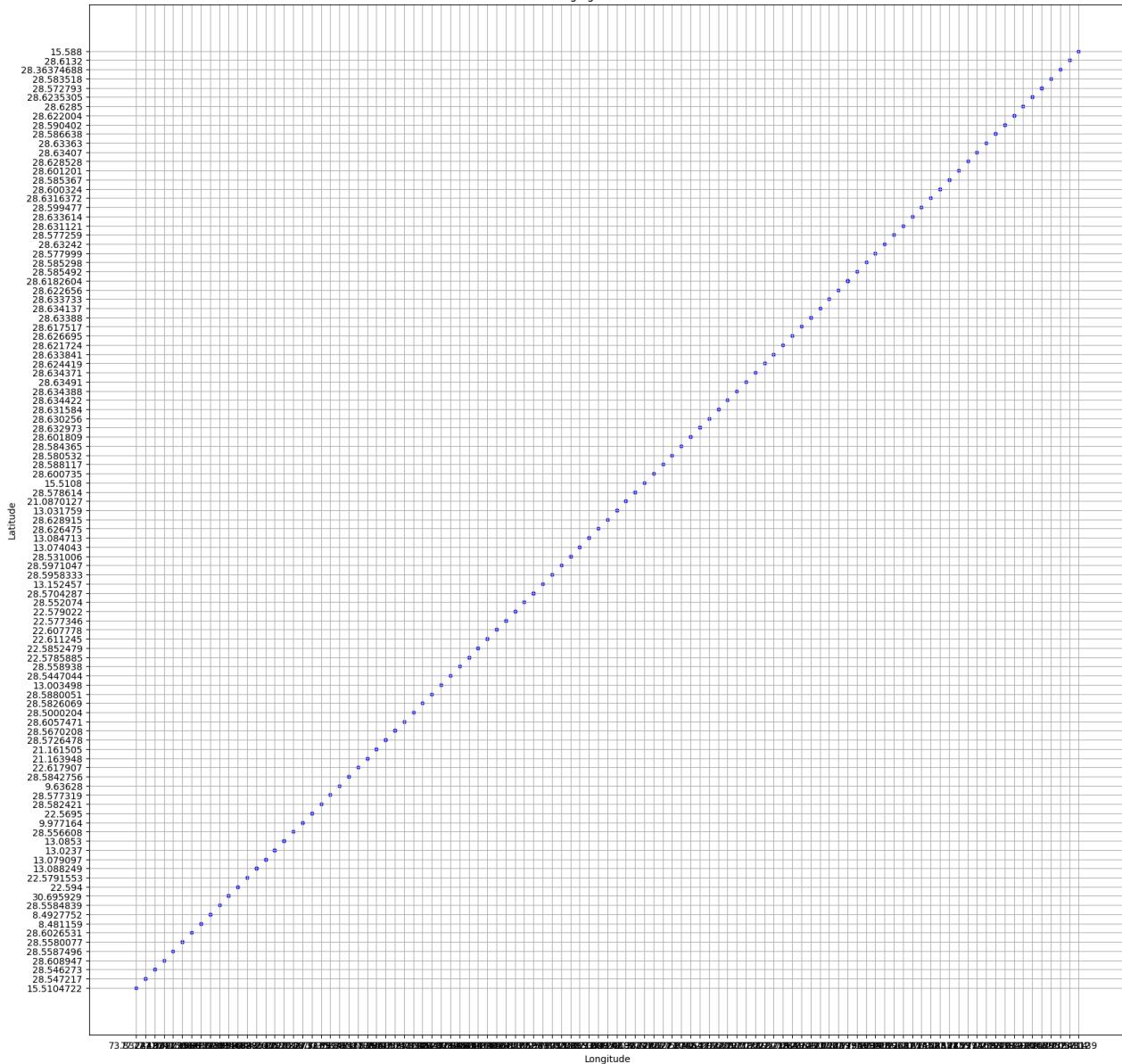
```
print(f"Number of EV stations in {city}: {len(city_stations)}")
```

```
Number of EV stations in Delhi: 74
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(20,20))
plt.scatter(stations['longitude'], stations['latitude'], color='blue', s=10)
plt.title(f'EV Charging Stations in {city}')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)
plt.show()
```

EV Charging Stations in Delhi



```

import folium

m = folium.Map(location=[28.6139, 77.2090], zoom_start=11) # Delhi center

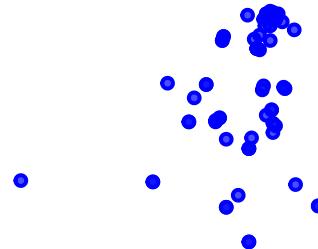
for _, row in city_stations.iterrows():
    folium.CircleMarker(
        location=[row['lattitude'], row['longitude']],
        radius=4,
        color='blue',
        fill=True,
        fill_opacity=0.7,
        popup=row['address']
    ).add_to(m)

m

```

+

-



Leaflet | © OpenStreetMap contributors

```
!pip install geopandas folium matplotlib shapely
```

```
Requirement already satisfied: geopandas in /usr/local/lib/python3.12/dist-packages (1.1.1)
Requirement already satisfied: folium in /usr/local/lib/python3.12/dist-packages (0.20.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: shapely in /usr/local/lib/python3.12/dist-packages (2.0.1)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.12/dist-packages (from geopandas) (2.0.2)
Requirement already satisfied: pygrio>=0.7.2 in /usr/local/lib/python3.12/dist-packages (from geopandas) (0.11.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from geopandas) (25.0)
Requirement already satisfied: pandas>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from geopandas) (2.2.2)
Requirement already satisfied: pyproj>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from geopandas) (3.7.2)
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from folium) (0.8.2)
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.12/dist-packages (from folium) (3.1.6)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from folium) (2.32.4)
Requirement already satisfied: xyzservices in /usr/local/lib/python3.12/dist-packages (from folium) (2025.4.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyParsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.post)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2>=2.9->folium) (3.0.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=2.0.0->geopandas) (2025)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=2.0.0->geopandas) (2022.7)
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from pygrio>=0.7.2->geopandas) (2025.10.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->folium) (2.1.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->folium) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->folium) (2.5.0)
```

```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
import folium
city = "Delhi"
city_stations = stations[stations['address'].str.contains(city, case=False, na=False)]
print(f"EV stations in {city}: {len(city_stations)}")
city_stations.head()
from shapely.geometry import Point

geometry = [Point(xy) for xy in zip(city_stations['longitude'], city_stations['lattitude'])]
geo_stations = gpd.GeoDataFrame(city_stations, geometry=geometry, crs="EPSG:4326")
geo_stations.head()
```

## EV stations in Delhi: 74

	address	city	availability	contact	isValidated	issue	lattitude	longitude	name	notes	phones	pincode	pi
1	SDMC Parking, R Block, GK- 1, DELHI- 110016	Delhi	1		1	0	28.547217	77.244324	EESL R block GK1			110016	
2	SDMC Parking, SDA Market, Hauz Khas, New Delhi...	Delhi	1		1	0	28.546273	77.19711	EESL SDA market			110016	
5	EESL PVR Priya Vasant Vihar, SDMC parking New ...	Delhi	1		1	0	28.5580077	77.1588093	EESL PVR Priya Delhi			110057	
6	EESL Meharchand Double Storey Market, SDMC par...	Delhi	1		1	0	28.6026531	77.1666318	EESL Meharchand Double Storey Market			110003	
9	EESL Veer Savarkar Marg, Lajpat Nagar, SDMC pa...	Delhi	1		1	0	28.5584839	77.0909568	EESL Veer Savarkar Marg, Lajpat Nagar			110024	

```

import geopandas as gpd

# Read the shapefile
delhi_boundary = gpd.read_file("/content/Admin2.shx")

# Check if the CRS is already set, otherwise set a known CRS for the source data if possible
# If you know the original CRS of your shapefile, replace 'EPSG:XXXX' with the correct code
if delhi_boundary.crs is None:
    print("CRS not found in the shapefile. Attempting to set a default CRS.")
    # This is a placeholder. You should replace 'EPSG:XXXX' with the actual CRS of your shapefile.
    # For example, if it's in WGS84, you might use 'EPSG:4326', but check your data source.
    # If you don't know the CRS, you might need to find documentation for the shapefile.
    # For demonstration, let's assume a common Indian CRS if applicable, or handle it based on metadata.
    # As a generic approach without specific knowledge of the source CRS,
    # we might need user input or metadata.
    # Let's try to proceed assuming a need to assign one before transforming.
    # A common approach if the source is local is a projected CRS, but without info, it's hard to guess.
    # If the data is indeed in lat/lon but without CRS info, setting it to 4326 first might be an option
    # IF AND ONLY IF you are sure that is the source CRS. The error suggests it's not even recognized as lat/lon.

    # Let's try setting a placeholder CRS first if none exists, then transform.
    # This might not be correct without knowing the source CRS, and the transformation will be wrong.
    # A better approach is to *know* the source CRS.
    # For now, I will add a placeholder and a print statement.
    # You *must* replace 'EPSG:XXXX' with the correct source CRS.
try:
    # Attempt to set a common CRS if missing, this is a guess and might be incorrect
    # A better fix requires knowing the source CRS of Admin2.shx
    delhi_boundary = delhi_boundary.set_crs(epsg=4326, allow_override=True) # Assuming 4326 might be the intended source
    print("Assigned EPSG:4326 as a placeholder CRS. Verify if this is correct.")
except Exception as e:
    print(f"Could not set CRS automatically: {e}")
    print("Please manually set the correct CRS for the delhi_boundary GeoDataFrame before transforming.")
    # If we can't even set a placeholder, we can't proceed to transform.
    # In a real scenario, you would stop or ask the user for the correct CRS.
    # For the sake of providing *a* modified cell, I'll keep the transform line,
    # but the user needs to fix the CRS setting part.
    pass # Continue to the transform line, which will likely fail if CRS isn't set

# Attempt to transform to EPSG:4326
# This line will work only if the CRS has been successfully set in the step above
delhi_boundary = delhi_boundary.to_crs(epsg=4326)

# Display the first few rows and the CRS to confirm
display(delhi_boundary.head())
print(f"CRS after transformation: {delhi_boundary.crs}")

```

```
CRS not found in the shapefile. Attempting to set a default CRS.  
Assigned EPSG:4326 as a placeholder CRS. Verify if this is correct.
```

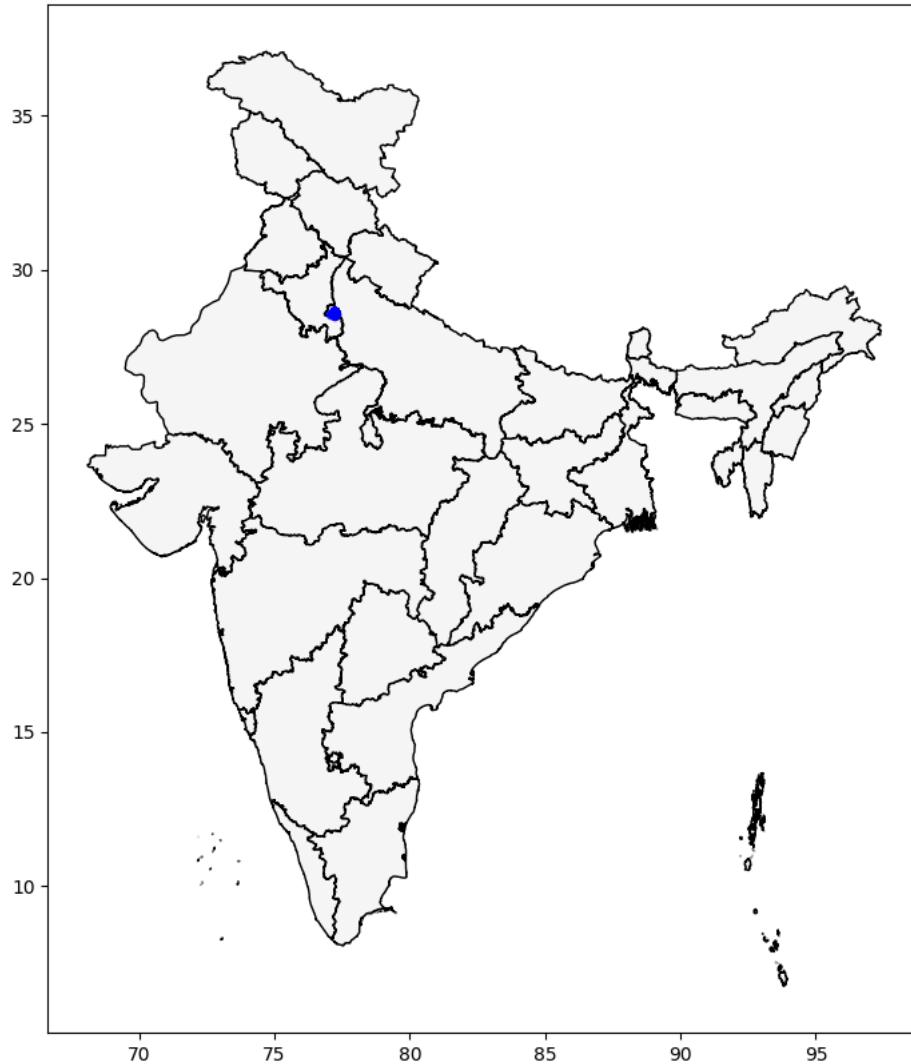
geometry 

- 0 POLYGON ((96.08866 29.45997, 96.09428 29.45477...)
- 1 POLYGON ((95.97166 27.96254, 95.97174 27.96227...)
- 2 POLYGON ((76.77175 30.79498, 76.77231 30.7942,...)
- 3 POLYGON ((77.32647 18.45884, 77.32648 18.45803...)
- 4 POLYGON ((94.57315 25.69156, 94.57522 25.69094...)

```
CRS after transformation: EPSG:4326
```

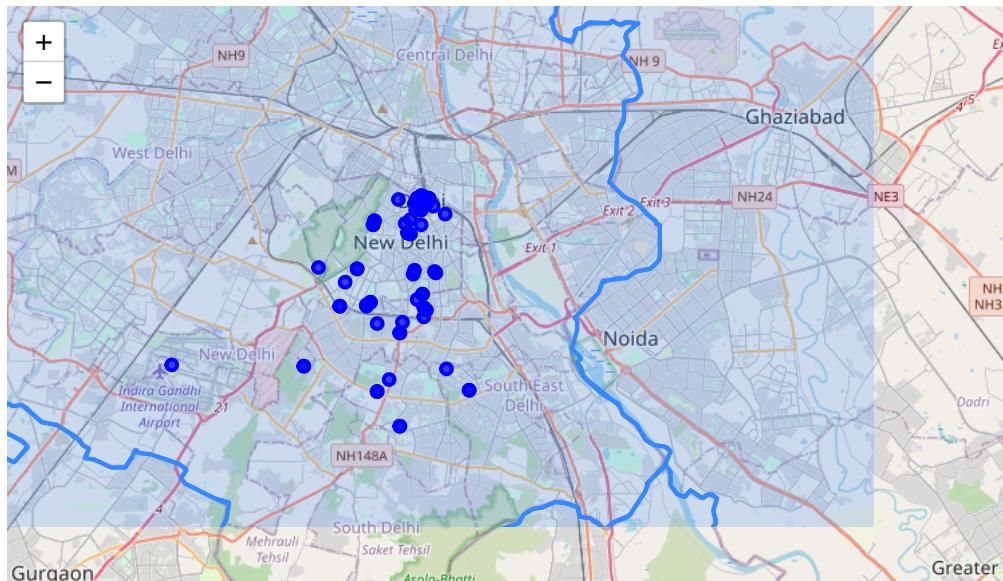
```
fig, ax = plt.subplots(figsize=(10,10))  
delhi_boundary.plot(ax=ax, color='whitesmoke', edgecolor='black')  
geo_stations.plot(ax=ax, color='blue', markersize=20, alpha=0.7)  
plt.title("EV Charging Stations in Delhi", fontsize=15)  
plt.show()
```

EV Charging Stations in Delhi



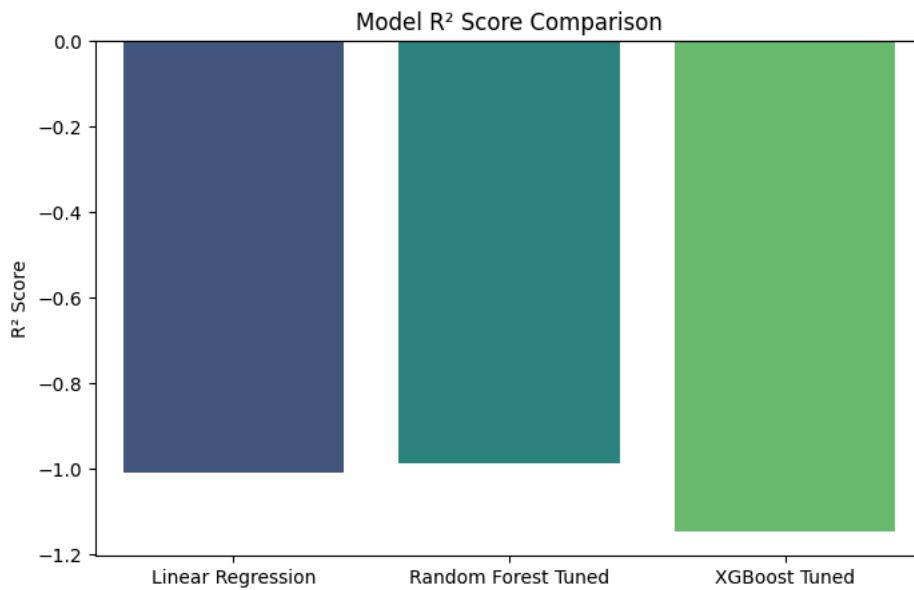
```
m = folium.Map(location=[28.6139, 77.2090], zoom_start=11)  
  
# Plot Delhi boundary  
folium.GeoJson(delhi_boundary.geometry.__geo_interface__, name='Delhi Boundary').add_to(m)  
  
# Plot EV Stations  
for _, row in geo_stations.iterrows():  
    folium.CircleMarker(  
        location=[row['lattitude'], row['longitude']],  
        radius=4,  
        color='blue',  
        fill=True,  
        fill_opacity=0.7,  
        popup=row['address'])  
    .add_to(m)
```

m



Leaflet | © OpenStreetMap contributors

```
plt.figure(figsize=(8,5))
sns.barplot(x=["Linear Regression", "Random Forest Tuned", "XGBoost Tuned"],
            y=[final_results_df.loc["LinearRegression", "R2"],
               final_results_df.loc["RandomForest_Tuned", "R2"],
               final_results_df.loc["XGBoost_Tuned", "R2"]], palette="viridis")
plt.title("Model R2 Score Comparison")
plt.ylabel("R2 Score")
plt.show()
```



```
geometry = [Point(xy) for xy in zip(stations['longitude'], stations['latitude'])]
geo_stations = gpd.GeoDataFrame(stations, geometry=geometry, crs="EPSG:4326")
geo_stations.head()
```

	address	city	availability	contact	isValidated	issue	lattitude	longitude	name	notes	phones	pincode	prici
0	EESL EDC Parking Goa	Goa	1		1	0	15.5104722	73.8372222	EESL Legislative Assembly Goa			403503	fr
1	SDMC Parking, R Block, GK-1, DELHI- 110016	Delhi	1		1	0	28.547217	77.244324	EESL R block GK1			110016	fr
2	SDMC Parking, SDA Market, Hauz Khas, New Delhi...	Delhi	1		1	0	28.546273	77.19711	EESL SDA market			110016	fr
3	Near Tata Advance Systems, sector- 59, Noida	Noida	1		1	0	28.608947	77.362725	EESL Tata Advance Systems Noida			201301	fr
4	EESL Feroze Gandhi Road, Lajpat Nagar, SDMC pa...	Delhi	1		1	0	28.5587496	77.0909566	EESL Feroze Gandhi Road Lajpat Nagar			110024	fr

Next steps: [Generate code with geo\\_stations](#) [New interactive sheet](#)

```
import zipfile
import os

# Path where you uploaded the zip file
zip_path = "/content/gadm41_IND_shp.zip"
extract_dir = "/content/india_shapefile"

# Extract
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

# Check extracted files
os.listdir(extract_dir)
```

```
['gadm41_IND_2.prj',
 'gadm41_IND_1.shx',
 'gadm41_IND_3.dbf',
 'gadm41_IND_3.cpg',
 'gadm41_IND_1.cpg',
 'gadm41_IND_0.prj',
 'gadm41_IND_0.shx',
 'gadm41_IND_2.shp',
 'gadm41_IND_1.shp',
 'gadm41_IND_2.dbf',
 'gadm41_IND_3.shx',
 'gadm41_IND_0.shp',
 'gadm41_IND_2.cpg',
 'gadm41_IND_1.dbf',
 'gadm41_IND_0.dbf',
 'gadm41_IND_1.prj',
 'gadm41_IND_3.prj',
 'gadm41_IND_2.shx',
 'gadm41_IND_0.cpg',
 'gadm41_IND_3.shp']
```

```
import geopandas as gpd
import os

extract_dir = "/content/india_shapefile"

# Load all shapefiles
```

```

levels = ["gadm41_IND_0.shp", "gadm41_IND_1.shp", "gadm41_IND_2.shp", "gadm41_IND_3.shp"]

shapefiles = {}
for level in levels:
    path = os.path.join(extract_dir, level)
    gdf = gpd.read_file(path)
    gdf = gdf.to_crs(epsg=4326)
    shapefiles[level] = gdf
    print(f"{level} loaded and converted.")

```

gadm41\_IND\_0.shp loaded and converted.  
 gadm41\_IND\_1.shp loaded and converted.  
 gadm41\_IND\_2.shp loaded and converted.  
 gadm41\_IND\_3.shp loaded and converted.

```

india_states = gpd.read_file("/content/india_shapefile/gadm41_IND_1.shp")
india_states = india_states.to_crs(epsg=4326)
india_states.head()

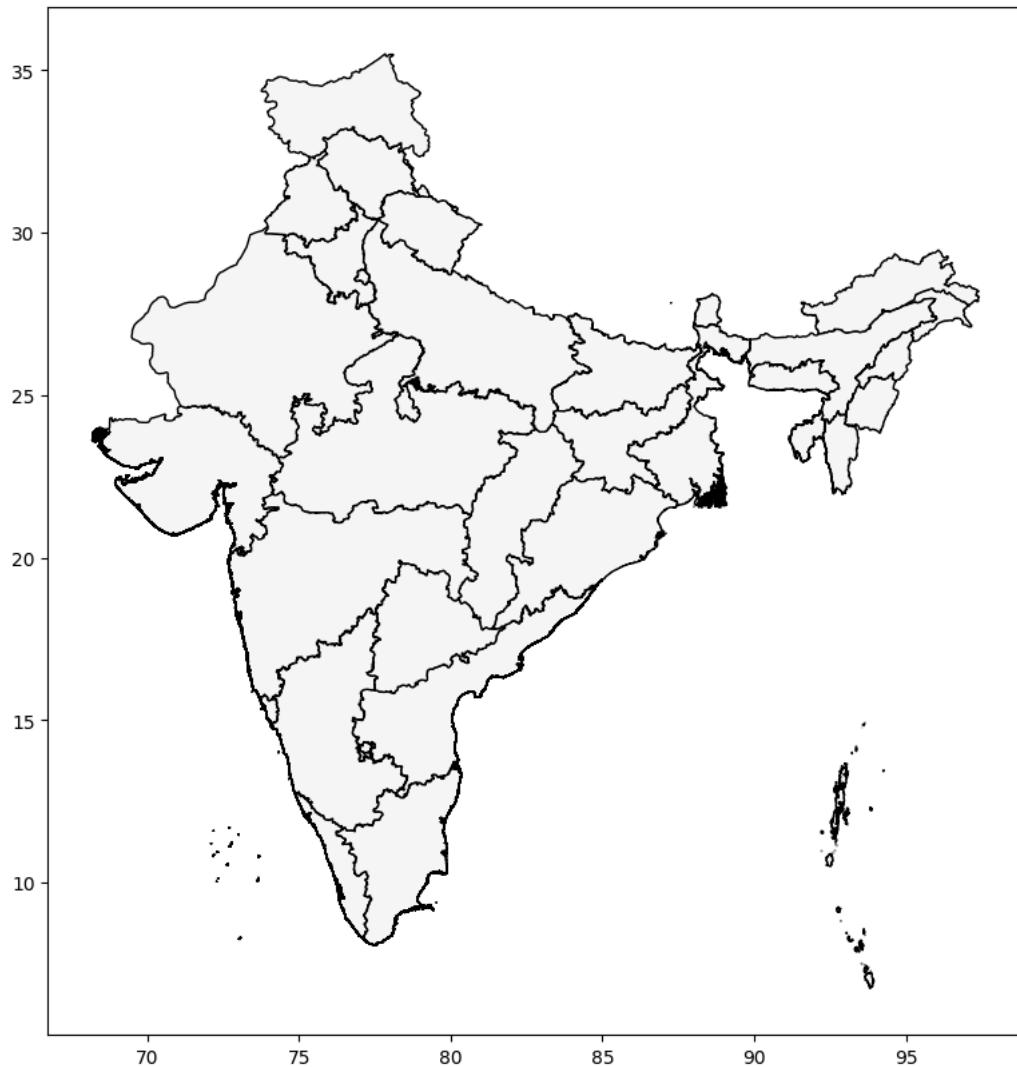
```

	GID_1	GID_0	COUNTRY	NAME_1	VARNAME_1	NL_NAME_1	TYPE_1	ENGTYPE_1	CC_1	HASC_1	ISO_1	geometry
0	IND.1_1	IND	India	Andaman and Nicobar Islands	Andaman & Nicobar Islands Andama	NA	Union Territory	Union Territory	NA	IN.AN	NA	MULTIPOLYGON (((93.79078 6.85139, 93.79092 6.8...))
1	IND.2_1	IND	India	Andhra Pradesh		NA	State	State	NA	IN.AP	IN-AP	MULTIPOLYGON (((78.73952 13.04549, 78.73218 13...))
2	IND.3_1	IND	India	Arunachal Pradesh	Agence de la Frontière du Nord-E	NA	State	State	NA	IN.AR	IN-AR	POLYGON ((95.3683 27.10736, 95.37366 27.108, 95.37366 27.108, 95.3683 27.10736))
3	Z07.3_1	Z07	India	Arunachal Pradesh	Agence de la Frontière du Nord-E	NA	State	State	NA	IN.AR	NA	POLYGON ((94.19125 27.49632, 94.1869 27.49632, 94.19125 27.49632, 94.1869 27.49632))

Next steps: [Generate code with india\\_states](#) [New interactive sheet](#)

```
india_states.plot(figsize=(10,10), color='whitesmoke', edgecolor='black')
```

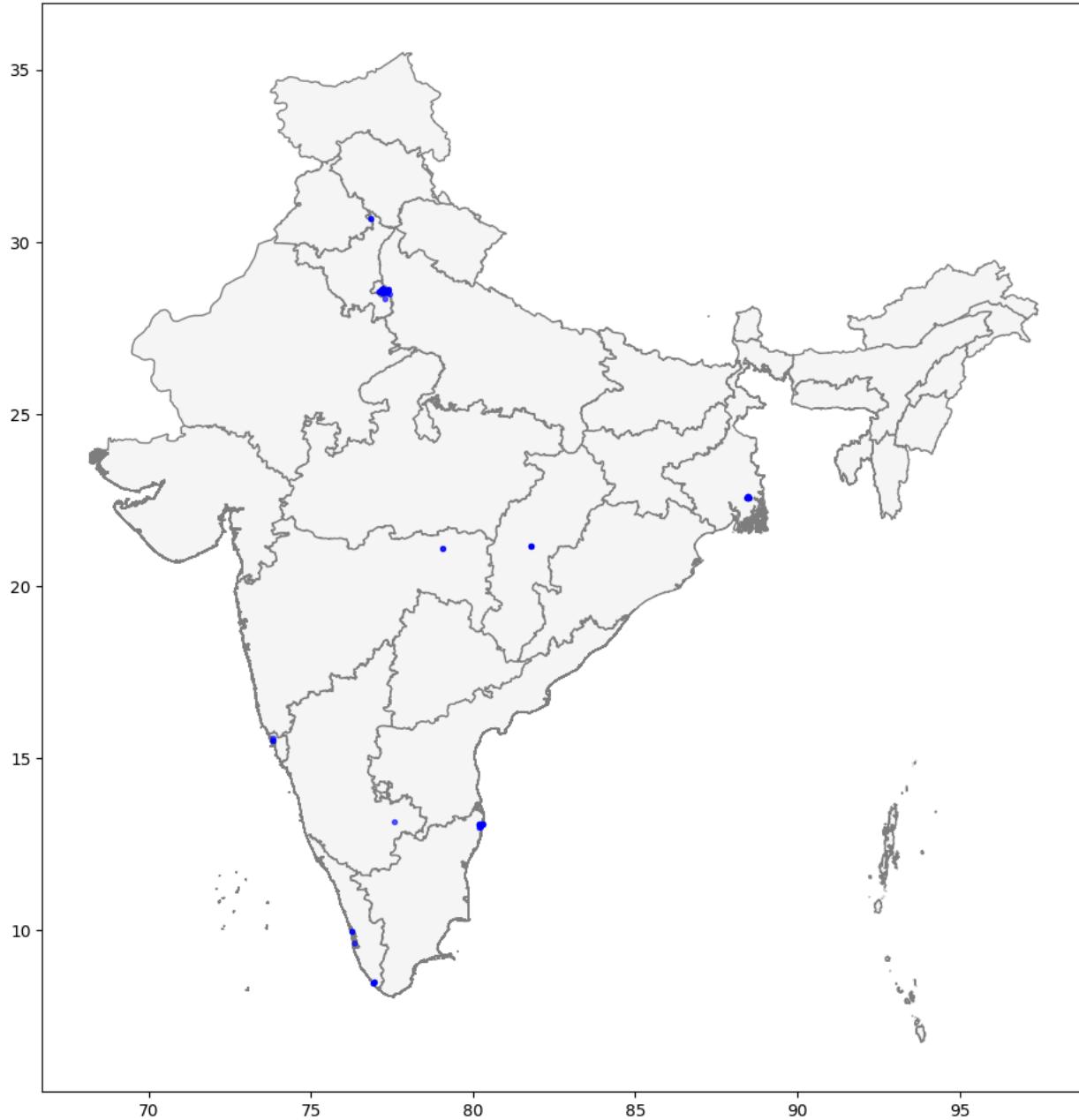
<Axes: >



```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(12,12))
india_states.plot(ax=ax, color='whitesmoke', edgecolor='gray')
geo_stations.plot(ax=ax, color='blue', markersize=8, alpha=0.6)
plt.title("EV Charging Stations Across India", fontsize=16)
plt.show()
```

## EV Charging Stations Across India



```
# Spatial join: assigns each station to a state polygon
joined = gpd.sjoin(geo_stations, india_states, how="inner", predicate='within')

# Count EV stations per state
state_counts = joined.groupby('NAME_1').size().reset_index(name='EV_Stations')
state_counts.head()
```

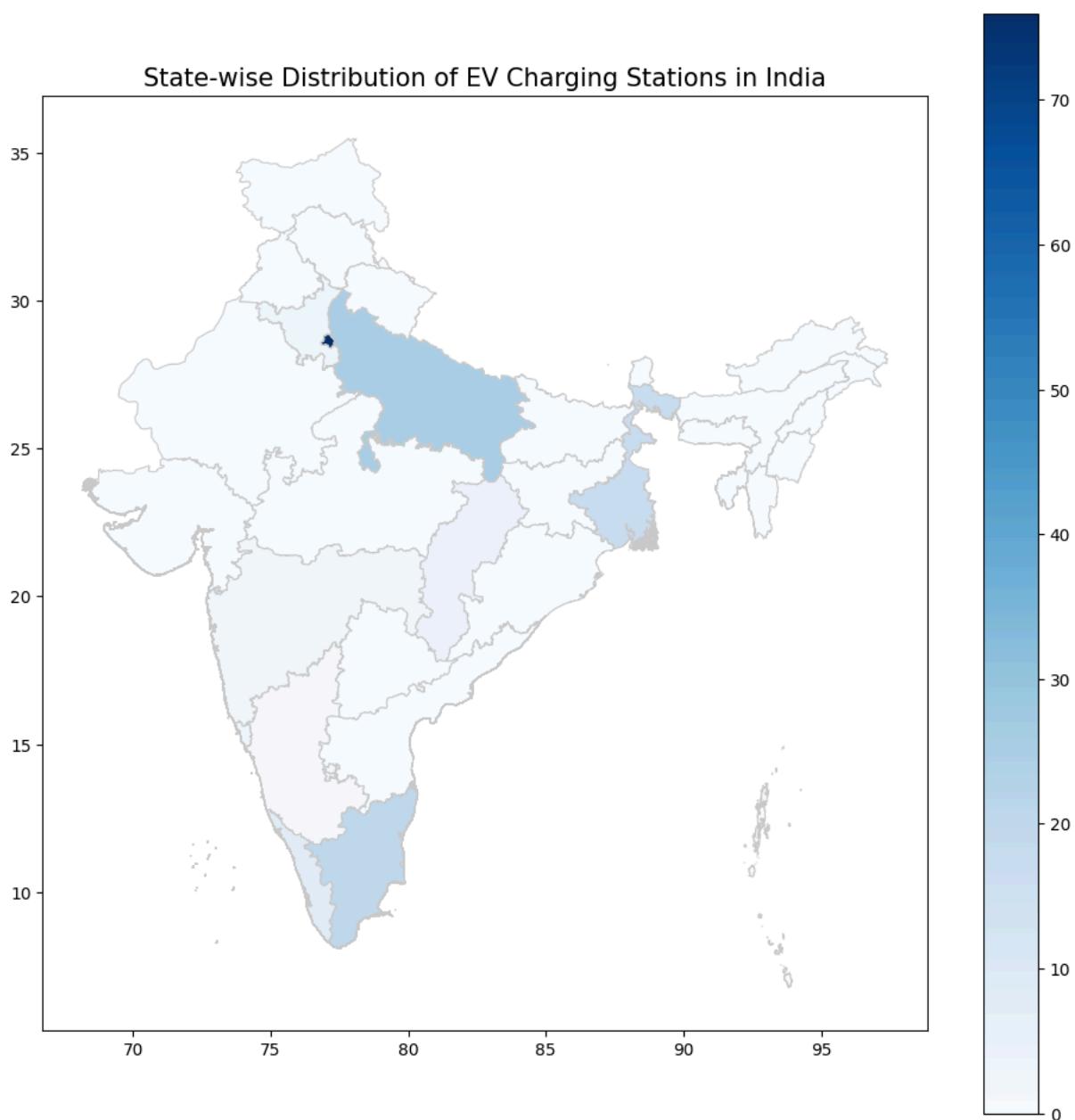
	NAME_1	EV_Stations
0	Chhattisgarh	4
1	Goa	3
2	Haryana	3
3	Karnataka	1
4	Kerala	7

Next steps: [Generate code with state\\_counts](#) [New interactive sheet](#)

```
# Merge counts with the India shapefile
india_states = india_states.merge(state_counts, left_on='NAME_1', right_on='NAME_1', how='left')
india_states['EV_Stations'] = india_states['EV_Stations'].fillna(0)

# Plot as Choropleth (colored by EV station count)
fig, ax = plt.subplots(figsize=(12,12))
```

```
india_states.plot(column='EV_Stations', cmap='Blues', linewidth=0.8, ax=ax, edgecolor='0.8', legend=True)
plt.title("State-wise Distribution of EV Charging Stations in India", fontsize=15)
plt.show()
```



```
import pandas as pd

population_data = pd.DataFrame({
    'State': [
        'Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar', 'Chhattisgarh', 'Goa',
        'Gujarat', 'Haryana', 'Himachal Pradesh', 'Jharkhand', 'Karnataka', 'Kerala',
        'Madhya Pradesh', 'Maharashtra', 'Manipur', 'Meghalaya', 'Mizoram', 'Nagaland',
        'Odisha', 'Punjab', 'Rajasthan', 'Sikkim', 'Tamil Nadu', 'Telangana', 'Tripura',
        'Uttar Pradesh', 'Uttarakhand', 'West Bengal', 'Delhi', 'Jammu and Kashmir'
    ],
    'Population': [
        53903393, 1570458, 35607039, 124799926, 29436231, 1586250, 63872399, 28204692, 7300000,
        38593948, 67562686, 35699443, 85358965, 124904071, 3091545, 3366710, 1239244, 2249695,
        46356334, 30141373, 81032689, 690251, 77841267, 39362732, 4169794, 237882725, 11141000,
        99609303, 16753235, 13606320
    ]
})
```

```
state_counts.rename(columns={'NAME_1':'State'}, inplace=True)
merged = pd.merge(state_counts, population_data, on='State', how='left')

# Compute EV stations per million population
merged['EVs_per_million'] = (merged['EV_Stations'] / merged['Population']) * 1_000_000
```

```
merged.head()
```

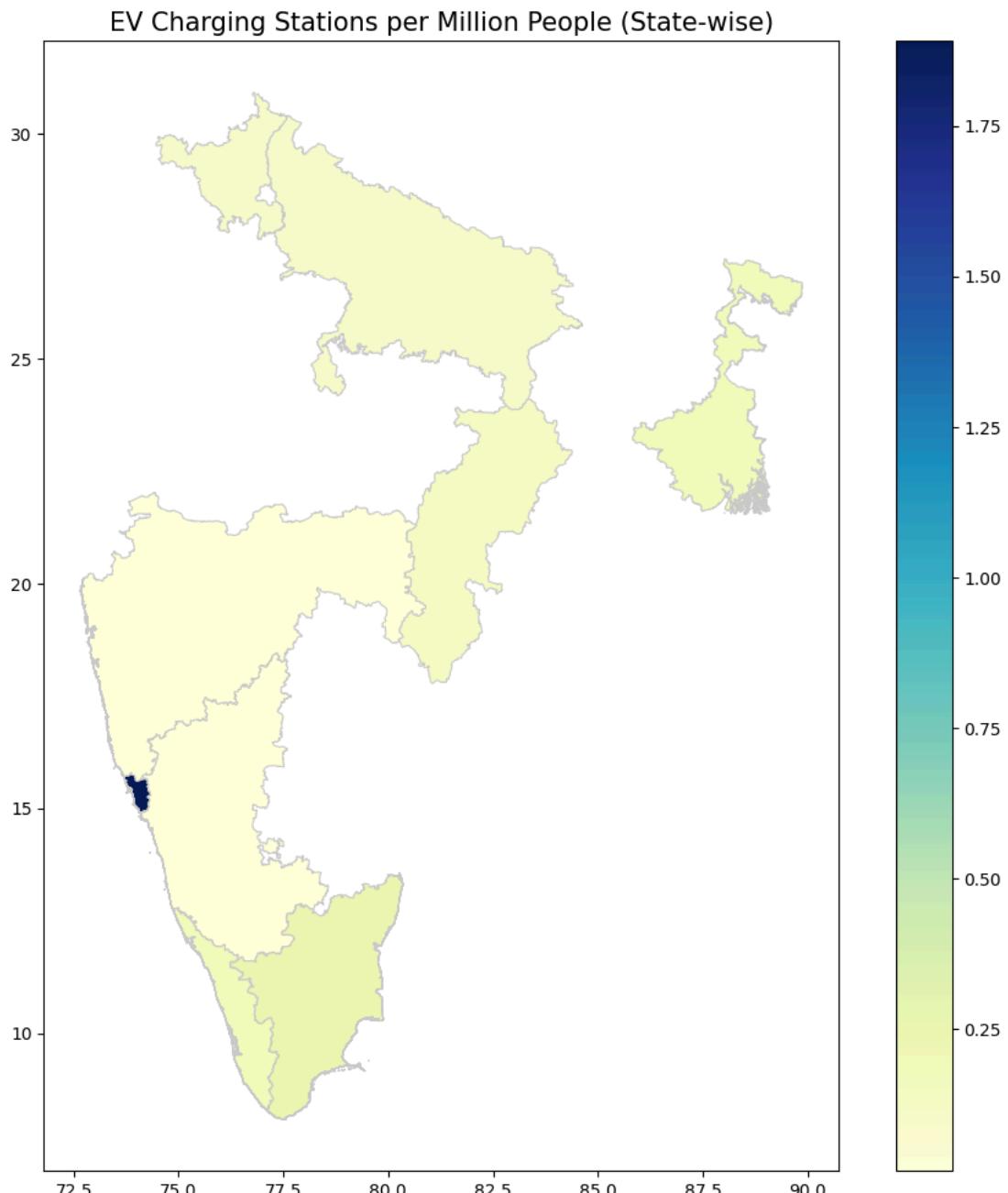
	State	EV_Stations	Population	EVs_per_million	
0	Chhattisgarh	4	29436231.0	0.135887	
1	Goa	3	1586250.0	1.891253	
2	Haryana	3	28204692.0	0.106365	
3	Karnataka	1	67562686.0	0.014801	
4	Kerala	7	35699443.0	0.196081	

Next steps: [Generate code with merged](#) [New interactive sheet](#)

```
india_states = india_states.merge(merged[['State','EVs_per_million']],
left_on='NAME_1', right_on='State', how='left')

import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(12,12))
india_states.plot(column='EVs_per_million', cmap='YlGnBu', linewidth=0.8, ax=ax, edgecolor='0.8', legend=True)
plt.title("EV Charging Stations per Million People (State-wise)", fontsize=15)
plt.show()
```



```
lowest_states = merged.sort_values(by='EVs_per_million').head(10)
lowest_states[['State', 'EV_Stations', 'Population', 'EVs_per_million']]
```

	State	EV_Stations	Population	EVs_per_million	
3	Karnataka	1	67562686.0	0.014801	
5	Maharashtra	2	124904071.0	0.016012	
8	Uttar Pradesh	25	237882725.0	0.105094	
2	Haryana	3	28204692.0	0.106365	
0	Chhattisgarh	4	29436231.0	0.135887	
9	West Bengal	18	99609303.0	0.180706	
4	Kerala	7	35699443.0	0.196081	
7	Tamil Nadu	20	77841267.0	0.256933	
1	Goa	3	1586250.0	1.891253	
6	NCT of Delhi	76	NaN	NaN	

```
# Run once at top of Colab
!pip install geopandas folium xgboost --quiet
!pip install shapely==2.0.1 --quiet # if needed for geopandas compatibility

import warnings
warnings.filterwarnings("ignore")
```

```
import numpy as np
import pandas as pd
import geopandas as gpd

# Ensure merged has required columns
print(merged.columns)

# If india_states has area, compute area_km2 for each state (in EPSG:3857 or similar)
# convert india_states temporarily to metric CRS for area
india_states_m = india_states.to_crs(epsg=3857)
india_states_m['area_km2'] = india_states_m.geometry.area / 1e6
india_states = india_states_m.to_crs(epsg=4326) # back to lat/lon

# Merge area into merged (lowercase/unified names may be required)
# ensure both state names match:
merged['state_key'] = merged['State'].str.strip().str.lower()
india_states['state_key'] = india_states['NAME_1'].str.strip().str.lower()

# merge area
merged2 = merged.merge(india_states[['state_key', 'area_km2']], on='state_key', how='left')

# Feature engineering
merged2['pop_density'] = merged2['Population'] / merged2['area_km2'] # people per km2
merged2['stations_per_million'] = merged2['EV_Stations'] / (merged2['Population']/1_000_000)
merged2 = merged2.fillna(0)

# Choose features and target
features = ['Population', 'area_km2', 'pop_density'] # add more features if available
X = merged2[features]
y = merged2['EV_Stations']

print(X.head())
print(y.head())
```

```
Index(['State', 'EV_Stations', 'Population', 'EVs_per_million'], dtype='object')
   Population      area_km2    pop_density
0  29436231.0  156854.949241  187.665299
1  1586250.0   3997.671488  396.793485
2  28204692.0  57990.838234  486.364620
3  67562686.0  205941.668840  328.067100
4  35699443.0  39243.154181  909.698615
0     4
1     3
2     3
3     1
4     7
Name: EV_Stations, dtype: int64
```

```

from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import xgboost as xgb
from sklearn.metrics import make_scorer, mean_absolute_error, mean_squared_error, r2_score

# Scorers (we compute negative MSE to get RMSE)
def rmse(y_true, y_pred): return np.sqrt(mean_squared_error(y_true, y_pred))
rmse_scoring = make_scorer(rmse, greater_is_better=False) # for cross_val_score (neg values)

models = {
    "LinearRegression": LinearRegression(),
    "Ridge": Ridge(alpha=1.0),
    "RandomForest": RandomForestRegressor(n_estimators=200, random_state=42),
    "XGBoost": xgb.XGBRegressor(n_estimators=200, random_state=42, objective='reg:squarederror'),
    "SVR": SVR(kernel='rbf', C=1.0, epsilon=0.1)
}

cv = KFold(n_splits=5, shuffle=True, random_state=42)

results = []
for name, model in models.items():
    # cross-validate for RMSE, MAE, R2
    neg_mse = cross_val_score(model, X, y, cv=cv, scoring='neg_mean_squared_error')
    rmse_cv = np.sqrt(-neg_mse).mean()
    mae_cv = -cross_val_score(model, X, y, cv=cv, scoring='neg_mean_absolute_error').mean()
    r2_cv = cross_val_score(model, X, y, cv=cv, scoring='r2').mean()
    results.append((name, rmse_cv, mae_cv, r2_cv))

results_df = pd.DataFrame(results, columns=['Model', 'RMSE', 'MAE', 'R2']).sort_values('RMSE')
results_df

```

	Model	RMSE	MAE	R2	
4	SVR	19.050628	15.305545	-20.486052	
2	RandomForest	21.175835	17.421000	-25.783788	
3	XGBoost	26.838312	20.561969	-24.892367	
1	Ridge	34.100477	29.642110	-78.609141	
0	LinearRegression	34.100620	29.642241	-78.609610	

Next steps: [Generate code with results\\_df](#) [New interactive sheet](#)

```

from sklearn.model_selection import GridSearchCV

# Random Forest tuning
rf = RandomForestRegressor(random_state=42)
rf_params = {
    'n_estimators':[100,300,500],
    'max_depth':[5,10,20, None],
    'min_samples_split':[2,5,10]
}
gs_rf = GridSearchCV(rf, rf_params, cv=cv, scoring='neg_mean_squared_error', n_jobs=-1)
gs_rf.fit(X, y)
best_rf = gs_rf.best_estimator_
print("Best RF params:", gs_rf.best_params_)

# XGBoost tuning
xg = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)
xg_params = {
    'n_estimators':[100,300,500],
    'max_depth':[3,6,10],
    'learning_rate':[0.01,0.1,0.2]
}
gs_xg = GridSearchCV(xg, xg_params, cv=cv, scoring='neg_mean_squared_error', n_jobs=-1)
gs_xg.fit(X, y)
best_xg = gs_xg.best_estimator_
print("Best XG params:", gs_xg.best_params_)

```

Best RF params: {'max\_depth': 5, 'min\_samples\_split': 10, 'n\_estimators': 100}  
 Best XG params: {'learning\_rate': 0.01, 'max\_depth': 6, 'n\_estimators': 100}

```

best_model = best_rf # or best_xg depending on results
best_model.fit(X, y)

# Predictions on the training set (we have small N, states ~30 -> use training predictions or CV preds)

```

```

y_pred = best_model.predict(X)

# Performance metrics
print("RMSE:", rmse(y, y_pred))
print("MAE:", mean_absolute_error(y, y_pred))
print("R2:", r2_score(y, y_pred))

# Attach predictions to merged2
merged2['Predicted_Stations'] = y_pred
merged2['Gap'] = merged2['Predicted_Stations'] - merged2['EV_Stations']
merged2[['State', 'EV_Stations','Predicted_Stations','Gap']].sort_values('Gap', ascending=False).head(20)

```

RMSE: 21.697515894682507

MAE: 15.382800000000003

R2: -0.004892731968665132

	State	EV_Stations	Predicted_Stations	Gap	
3	Karnataka	1	17.414	16.414	
5	Maharashtra	2	17.414	15.414	
2	Haryana	3	17.414	14.414	
1	Goa	3	17.414	14.414	
0	Chhattisgarh	4	17.414	13.414	
4	Kerala	7	17.414	10.414	
9	West Bengal	18	17.414	-0.586	
7	Tamil Nadu	20	17.414	-2.586	
8	Uttar Pradesh	25	17.414	-7.586	
6	NCT of Delhi	76	17.414	-58.586	

```

#Choropleth of Gap (needed stations)
# merge merged2 into india_states GeoDataFrame
india_states['state_key'] = india_states['NAME_1'].str.strip().str.lower()
mdf = india_states.merge(merged2[['state_key','Gap','Predicted_Stations','stations_per_million']],
                        on='state_key', how='left')

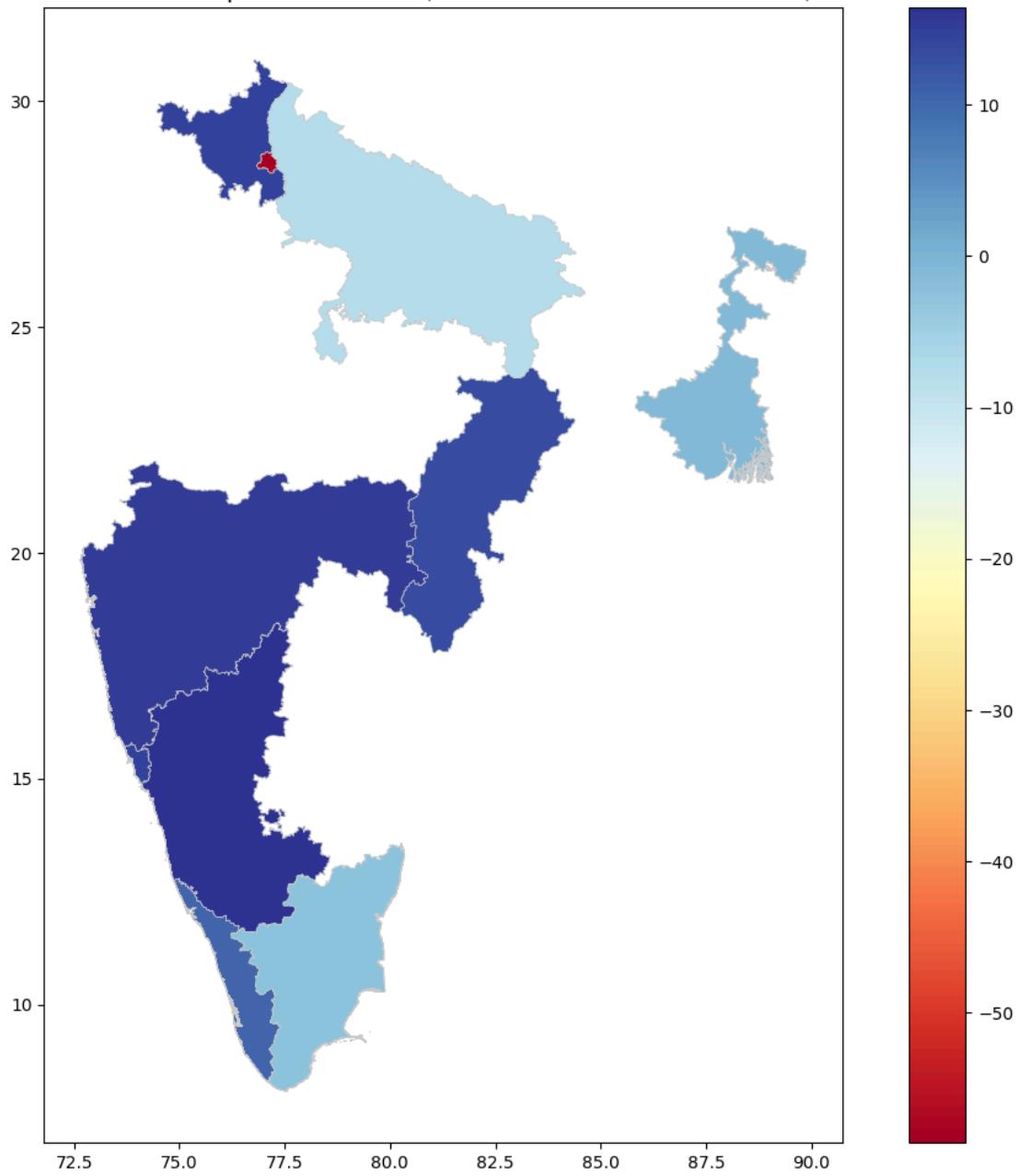
# Print columns of mdf to diagnose
print("Columns in mdf after merge:", mdf.columns)

fig, ax = plt.subplots(1,1, figsize=(14,12))
# Ensure the 'Gap' column exists and is used for plotting
if 'Gap' in mdf.columns: # Check for the correct column name
    mdf.plot(column='Gap', cmap='RdYlBu', legend=True, ax=ax, edgecolor='0.8', linewidth=0.5) # Use the correct column name
    ax.set_title("Predicted Gap in EV Stations (Positive -> Need more stations)", fontsize=14)
    plt.show()
else:
    print("Error: 'Gap' column not found in the merged DataFrame.") # Update error message

```

```
Columns in mdf after merge: Index(['GID_1', 'GID_0', 'COUNTRY', 'NAME_1', 'VARNAME_1', 'NL_NAME_1',
   'TYPE_1', 'ENGTYPPE_1', 'CC_1', 'HASC_1', 'ISO_1', 'geometry',
   'EV_Stations', 'State', 'EVs_per_million', 'area_km2', 'state_key',
   'Gap', 'Predicted_Stations', 'stations_per_million'],
  dtype='object')
```

Predicted Gap in EV Stations (Positive -> Need more stations)



```
from sklearn.cluster import KMeans

# pick a state with large gap
state_name = merged2.sort_values('Gap', ascending=False).iloc[0]['State'] # top need
print("Top state:", state_name)

# filter stations and state polygon
state_key = state_name.strip().lower()
state_poly = india_states[india_states['state_key']==state_key].geometry.unary_union

# Filter stations inside that state
state_stations = geo_stations[geo_stations.within(state_poly)]

# Get coordinates of existing stations and optionally population centroids (if available)
coords = np.array(list(zip(state_stations.geometry.x, state_stations.geometry.y)))

# determine number of new stations to propose (ceiling of gap)
n_new = int(np.ceil(merged2[merged2['state_key']==state_key]['Gap'].values[0]))
n_new = max(1, n_new) # at least 1

# Option 1: cluster existing stations to suggest centroids for coverage expansion
# Option 2 (better): create grid or sample population / OSM nodes and run KMeans on locations to cover underserved areas
if len(coords) >= n_new:
```

```

kmeans = KMeans(n_clusters=n_new, random_state=42).fit(coords)
centroids = kmeans.cluster_centers_
else:
    # fallback: spread points via grid sampling inside polygon
    minx, miny, maxx, maxy = state_poly.bounds
    xs = np.linspace(minx, maxx, int(np.sqrt(n_new)*4))
    ys = np.linspace(miny, maxy, int(np.sqrt(n_new)*4))
    pts = np.array([[x,y] for x in xs for y in ys])
    centroids = pts[:n_new]

centroids

```

```

Top state: Karnataka
array([[74.054306 , 11.57448006],
       [74.054306 , 12.03318939],
       [74.054306 , 12.49189873],
       [74.054306 , 12.95060806],
       [74.054306 , 13.4093174 ],
       [74.054306 , 13.86802673],
       [74.054306 , 14.32673607],
       [74.054306 , 14.7854454 ],
       [74.054306 , 15.24415474],
       [74.054306 , 15.70286407],
       [74.054306 , 16.16157341],
       [74.054306 , 16.62028274],
       [74.054306 , 17.07899208],
       [74.054306 , 17.53770142],
       [74.054306 , 17.99641075],
       [74.054306 , 18.45512009],
       [74.35585019, 11.57448006]])

```

```

# --- STEP 1: Import required libraries ---
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from shapely.geometry import Point, Polygon
from shapely.ops import unary_union

# ML Libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# --- STEP 2: Load dataset ---
data = merged2.dropna(subset=['EVs_per_million'])
data = data.reset_index(drop=True)

# --- STEP 3: Prepare features and target ---
X = data[['Population']]
y = data['EVs_per_million']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# --- STEP 4: Define Models ---
models = {
    "LinearRegression": LinearRegression(),
    "RandomForest": RandomForestRegressor(random_state=42),
    "XGBoost": XGBRegressor(objective='reg:squarederror', random_state=42)
}

results = {}

# --- STEP 5: Train & Evaluate Baseline Models ---
for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    results[name] = {
        "R2": r2_score(y_test, preds),
        "MAE": mean_absolute_error(y_test, preds),
        "MSE": mean_squared_error(y_test, preds)
    }

results_df = pd.DataFrame(results).T
print("📊 Baseline Model Comparison:")
print(results_df)

# --- STEP 6: Hyperparameter Tuning for RF & XGB ---
# Random Forest
rf_params = {

```

```

    'n_estimators': [100, 200, 300, 400],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf_grid = GridSearchCV(
    RandomForestRegressor(random_state=42),
    rf_params,
    cv=3,
    scoring='r2',
    n_jobs=-1,
    verbose=0
)
rf_grid.fit(X_train, y_train)
best_rf = rf_grid.best_estimator_

# XGBoost
xgb_params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0]
}

xgb_grid = GridSearchCV(
    XGBRegressor(objective='reg:squarederror', random_state=42),
    xgb_params,
    cv=3,
    scoring='r2',
    n_jobs=-1,
    verbose=0
)
xgb_grid.fit(X_train, y_train)
best_xgb = xgb_grid.best_estimator_

# --- STEP 7: Evaluate tuned models ---
tuned_models = {
    "LinearRegression": models["LinearRegression"],
    "RandomForest_Tuned": best_rf,
    "XGBoost_Tuned": best_xgb
}

final_results = {}
for name, model in tuned_models.items():
    preds = model.predict(X_test)
    final_results[name] = {
        "R2": r2_score(y_test, preds),
        "MAE": mean_absolute_error(y_test, preds),
        "MSE": mean_squared_error(y_test, preds)
    }

final_results_df = pd.DataFrame(final_results).T
print("\n👉 Tuned Model Comparison:")
print(final_results_df)

# --- STEP 8: Choose Best Model ---
best_model_name = final_results_df['R2'].idxmax()
best_model = tuned_models[best_model_name]
print(f"\n✅ Best model selected: {best_model_name}")

# --- STEP 9: Predictions and Gap Calculation ---
data['Predicted_EVs_per_million'] = best_model.predict(data[['Population']])
data['Gap'] = data['Predicted_EVs_per_million'] - data['EVs_per_million']

# --- STEP 10: Load India Shapefile ---
india_states = gpd.read_file("/content/india_shapefile/gadm41_IND_1.shp")
india_states['NAME_1'] = india_states['NAME_1'].str.strip()

# --- STEP 11: Merge shapefile with predictions ---
merged_map = india_states.merge(data, left_on='NAME_1', right_on='State', how='left')
merged_map['centroid'] = merged_map.geometry.centroid

# --- STEP 12: Identify Top 10 States for New Stations ---
centroids_gdf = gpd.GeoDataFrame(merged_map[['State', 'Gap']], geometry=merged_map['centroid'], crs=merged_map.crs)
high_gap = centroids_gdf.dropna(subset=['Gap']).sort_values('Gap', ascending=False).head(10)

print("\n⚡ Suggested New EV Station Centers (Top 10 States):")
print(high_gap[['State', 'Gap']])

# --- STEP 13: Create Coverage Zones ---
high_gap['coverage_zone'] = high_gap.geometry.buffer(0.5)

```

```

# --- STEP 14: Map Visualization ---
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
ax.set_aspect('equal')
merged_map.plot(column='Gap', cmap='coolwarm', linewidth=0.8, ax=ax, edgecolor='0.8', legend=True,
                 missing_kwds={'color': 'lightgrey'})
high_gap.plot(ax=ax, color='yellow', markersize=50, label='Suggested New EV Stations')
high_gap['coverage_zone'].plot(ax=ax, color='yellow', alpha=0.3, label='Coverage Zone')
plt.title(f'💡 Optimal EV Station Placement ({best_model_name})', fontsize=15)
plt.axis('off')
plt.legend()
plt.show()

# --- STEP 15: Bar Chart of Actual vs Predicted ---
plt.figure(figsize=(12,6))
plt.bar(data['State'], data['EVs_per_million'], label='Existing', alpha=0.6)
plt.bar(data['State'], data['Predicted_EVs_per_million'], label='Predicted Need', alpha=0.6)
plt.xticks(rotation=90)
plt.ylabel("EVs per million")
plt.title("Existing vs Predicted EV Station Density by State")
plt.legend()
plt.tight_layout()
plt.show()

# --- STEP 16: Residual Analysis ---
data['residual'] = data['EVs_per_million'] - data['Predicted_EVs_per_million']
plt.figure(figsize=(8,5))
sns.histplot(data['residual'], kde=True)
plt.title("Distribution of Residuals (Actual - Predicted)")
plt.xlabel("Residual")
plt.show()

# --- STEP 18: Combined Coverage Area ---
combined_coverage = unary_union(list(high_gap['coverage_zone'].dropna()))
print("\n✅ Combined total coverage area (approx, in degrees²):", combined_coverage.area)

```

```
# Choose Best Model
best_model_name = final_results_df['R2'].idxmax()
best_model = tuned_models[best_model_name]
print(f"\n✓ Best model selected: {best_model_name}")

# Predictions and Gap Calculation
data['Predicted_EVs_per_million'] = best_model.predict(data[['Population']])
```