PRASAD V POTLURI

SIDDHARTHA INSTITUTE OF TECHNOLOGY, KANURU, VIJAYAWADA

DEPARTMENT OF INFORMATION TECHNOLOGY

CASE STUDY

DATA SCIENCE (20IT4703E)

YEAR: IV B.TECH SEM 1 **AY**:2025-2026

REGULATION: PVP20

CASE STUDY: - - EV (Electric Vehicle) Charging Station Optimization

Research Goal: -

The adoption of EVs is hindered by insufficient and poorly located charging infrastructure. Current planning does not fully account for driver behavior, energy demand, or traffic flow. By analyzing GPS mobility data, EV usage patterns, and grid capacity, data science can optimize placement and scheduling of EV charging stations.

SUBMITTED BY:

22501A1279

22501A12A4

22501A12C8

22501A1295

1: Setting the Research Goal

This is the most critical first step. It defines the "north star" for the entire project. Without a clear goal, the analysis can become unfocused and the results may not be useful. This step ensures that the technical work is aligned with a real business problem.

- **1.1: Define Research Goal**

  The primary research goal was formally defined to address the business challenge of inefficiently located EV charging stations. The goal is to move from guesswork to a data-driven strategy.

  The research goal is to analyze EV usage patterns and station characteristics to develop a predictive model that can identify the key drivers of charging demand. This model will be used to create a framework for optimizing the placement of future charging stations to maximize usage and profitability.

- **1.2: Create Project Charter**

  A project charter outlines the scope, objectives, and stakeholders. While a formal document was not created, the key elements were established:

  - **Objective:** Build a regression model to predict `Charging_Sessions_per_Day`.
  - **Success Criteria:** Achieve an R-squared (R^2) value greater than 0.70, indicating a strong predictive model.
  - **Key Stakeholders:** The company's management and infrastructure planning team.
  - **Scope:** The project will use the provided `ev_station_dataset.csv` and standard data science techniques. The final deliverable will be a report summarizing the findings and a recommendation for a site-selection strategy.

2: Retrieving Data

This phase is about sourcing and acquiring the raw data needed for the project. A comprehensive analysis often requires combining data from different sources. This step formally categorizes the data's origin, which is crucial for understanding its context, potential biases, and ownership constraints.

- **2.1: External Data**

  External data is any data that originates from outside the organization. This includes public datasets, data from third-party providers, or government sources. For this case study, the primary dataset is external.

  The dataset used is the **`ev_station_dataset.csv`**, sourced from the public data science platform, Kaggle. This file contains a comprehensive list of EV charging stations with numerous attributes, providing a rich foundation for the analysis. It was retrieved by downloading the CSV file and loading it into the project environment.

```python
import pandas as pd

# Retrieving the external dataset from a local file path
df = pd.read_csv('ev_station_final_dataset.csv')

print("External dataset from Kaggle retrieved successfully.")
```

```
External dataset from Kaggle retrieved successfully.
```

- **2.2: Internal Data**

  Internal data is proprietary data that belongs to the company or organization conducting the analysis. While we are not using a separate internal dataset for this specific case study, it's important to understand what it would entail.

  - **Data Retrieval (Hypothetical):** If this project were being conducted by an EV charging company, "internal data" would be their own operational records. This could include real-time usage logs from their stations, customer payment information, or maintenance reports. This data would typically be retrieved by querying the company's internal databases (e.g., using SQL).
  - **Ownership (Hypothetical):** The ownership of this internal data would belong to the company itself. This means its use would be governed by internal data privacy policies and would not be publicly available. For this project, the `ev_station_dataset.csv` is considered the primary "owned" data for the scope of our analysis, even though it originated externally.

3: Data Preparation

Raw data from the real world is often messy, inconsistent, and not in the right format for machine learning. The data preparation phase is a systematic process to convert this raw data into a high-quality, reliable asset. Without proper preparation, the insights from the analysis would be flawed, and the model's predictions would be inaccurate—a concept known as "garbage in, garbage out."

3.1: Data Cleansing

The goal of data cleansing is to identify and fix errors, inconsistencies, and structural problems in the dataset. This ensures that the foundation of your analysis is accurate and trustworthy.

3.Data Preparation

3.1: Data cleansing

Loading dataset

```
import pandas as pd
import numpy as np

# Load the final practice dataset
df = pd.read_csv('ev_station_final_dataset.csv')

# Initial check to see the data issues
print("--- Initial Data Check ---")
print("Missing values count:")
print(df.isnull().sum().sort_values(ascending=False).head())
print(f"\nNumber of duplicate rows: {df.duplicated().sum()}")
```

```
--- Initial Data Check ---
Missing values count:
EV_Level2_EVSE_Num    101
EV_Level1_EVSE_Num    100
Station_Name            0
Latitude                0
Longitude               0
dtype: int64

Number of duplicate rows: 5
```

- **3.1.1: Handling Missing Data**

  Datasets often have empty cells or missing values. These must be addressed because machine learning algorithms cannot work with them. Instead of deleting rows and losing valuable information, we use **imputation** to fill the gaps with a statistically sound value, like the median.

Output:

3.1.1. Handling Missing Data

```
# Select columns with missing values and fill them with the median
for col in ['EV_Level1_EVSE_Num', 'EV_Level2_EVSE_Num']:
    median_value = df[col].median()
    df[col].fillna(median_value, inplace=True)

print("Missing values have been imputed with the median.")
df.describe()
```

```
Missing values have been imputed with the median.
```

	Station_ID	Latitude	Longitude	EV_Level1_EVSE_Num	EV_Level2_EVSE_Num	EV_DC_Fast_Count
count	2010.000000	2010.000000	2010.000000	2010.000000	2010.000000	2010.000000
mean	10994.549751	33.918370	105.497701	5.512935	11.853234	3.972637
std	580.295996	12.628742	20.150843	2.198937	4.257380	1.992956
min	10000.000000	12.029380	70.104800	2.000000	5.000000	1.000000
25%	10492.250000	23.178875	88.434245	4.000000	8.000000	2.000000
50%	10994.500000	33.734300	106.120485	6.000000	12.000000	4.000000
75%	11496.750000	45.315307	122.505630	7.000000	15.000000	6.000000
max	11999.000000	55.966490	139.998960	9.000000	19.000000	7.000000

- **3.1.2:Handling Inconsistent Data (Duplicates, Typos, Whitespace)**

  Inconsistencies like duplicate entries, spelling mistakes, or extra spaces can corrupt analysis. For example, 'PUBLIC' and 'Publi_c' would be treated as two different categories. This step standardizes the data to ensure uniformity.

Output:

3.1.2. Handling Inconsistent Data

3.1.2.1: Remove Duplicate Rows

```
# Remove the 10 duplicate rows
df.drop_duplicates(inplace=True)
# Reset the index to keep it clean and sequential
df.reset_index(drop=True, inplace=True)

print(f"Duplicate rows have been removed. New duplicate count: {df.duplicated().sum()}")
```

```
Duplicate rows have been removed. New duplicate count: 0
```

3.1.2.2: Correct Typos

```
# Correct the typo in the 'Access_Code' column
df['Access_Code'] = df['Access_Code'].replace('Publi_c', 'PUBLIC')

print("Typos in 'Access_Code' corrected.")
```

```
Typos in 'Access_Code' corrected.
```

3.1.2.3: Trim Whitespace

```
# Remove extra spaces from the beginning and end of station names
df['Station_Name'] = df['Station_Name'].str.strip()

print("Leading/trailing whitespace has been removed from 'Station_Name'.")
df.describe()
```

```
Leading/trailing whitespace has been removed from 'Station_Name'.
```

	Station_ID	Latitude	Longitude	EV_Level1_EVSE_Num	EV_Level2_EVSE_Num	EV_DC_Fast_Count	Grid_Capacity_kW
count	2010.000000	2010.000000	2010.000000	2010.000000	2010.000000	2010.000000	2010.00000
mean	10994.549751	33.918370	105.497701	5.512935	11.853234	3.972637	149.09403
std	580.295996	12.628742	20.150843	2.198937	4.257380	1.992956	58.74429
min	10000.000000	12.029380	70.104800	2.000000	5.000000	1.000000	50.00000
25%	10492.250000	23.178875	88.434245	4.000000	8.000000	2.000000	96.00000
50%	10994.500000	33.734300	106.120485	6.000000	12.000000	4.000000	151.00000
75%	11496.750000	45.315307	122.505630	7.000000	15.000000	6.000000	201.00000
max	11999.000000	55.966490	139.998960	9.000000	19.000000	7.000000	249.00000

3.2: Data Transformation

The goal of data transformation is to reformat and re-engineer the data to be more suitable for machine learning models. This involves changing data types to their correct format and creating new features that can reveal stronger patterns to the model.

- **3.2.1: Data Type Conversion**

  Columns are often loaded with a generic text-based data type. We convert them to their proper types (e.g., dates, numbers) to enable correct calculations and analysis.

Output:

3.2.1: Data Type Conversion

```
# Convert date columns from text to datetime objects
df['Open_Date'] = pd.to_datetime(df['Open_Date'])
df['Date_Last_Confirmed'] = pd.to_datetime(df['Date_Last_Confirmed'])

# Verify the change
print("Data types after conversion:")
df.info()
```

```
Data types after conversion:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2005 entries, 0 to 2004
Data columns (total 31 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Station_ID                 2005 non-null   int64
 1   Station_Name               2005 non-null   object
 2   Latitude                   2005 non-null   float64
 3   Longitude                  2005 non-null   float64
 4   City                       2005 non-null   object
 5   State                      2005 non-null   object
 6   EV_Level1_EVSE_Num         2005 non-null   float64
 7   EV_Level2_EVSE_Num         2005 non-null   float64
 8   EV_DC_Fast_Count           2005 non-null   int64
 9   EV_Connector_Types         2005 non-null   object
 10  EV_Pricing                 2005 non-null   object
 11  Access_Code                2005 non-null   object
 12  Access_Days_Time           2005 non-null   object
 13  Status_Code                2005 non-null   object
 14  EV_Network                 2005 non-null   object
 15  Owner_Type_Code            2005 non-null   object
 16  Open_Date                  2005 non-null   datetime64[ns]
 17  Date_Last_Confirmed        2005 non-null   datetime64[ns]
 18  Energy_Source              2005 non-null   object
 19  Grid_Capacity_kW           2005 non-null   int64
 20  Charging_Speed_kW          2005 non-null   int64
 21  Charging_Sessions_per_Day  2005 non-null   int64
 22  Peak_Hour_Usage            2005 non-null   object
 23  Trip_Start_Lat             2005 non-null   float64
 24  Trip_End_Lon               2005 non-null   float64
 25  Distance_km                2005 non-null   float64
 26  Traffic_Volume_per_Day     2005 non-null   int64
 27  City_Population            2005 non-null   int64
 28  Proximity_to_POI_km        2005 non-null   float64
 29  Competitor_Density         2005 non-null   int64
 30  Local_EV_Penetration       2005 non-null   int64
dtypes: datetime64[ns](2), float64(8), int64(9), object(12)
memory usage: 485.7+ KB
```

- **3.2.2: Derived Measures**

  This is the creative process of creating new, more insightful features from the existing data. A well-engineered feature can significantly improve a model's predictive power.

Output:

3.2.2: Derived measures

```
#Sub-step 2.1: Create a 'Total Chargers' Column
# Sum up all charger types to get a total count per station
df['Total_EVSEs'] = df['EV_Level1_EVSE_Num'] + df['EV_Level2_EVSE_Num'] + df['EV_DC_Fast_Count']

print("Created 'Total_EVSEs' column.")
```

```
Created 'Total_EVSEs' column.
```

- **3.2.3: Creating Dummies (Handling Categorical Data & Standardization)**

  Machine learning models require all input to be numerical. We first convert text-based categories into numbers using **one-hot encoding**. Then, we use **standardization** to scale all numerical features to a common range, which helps the model learn more effectively.

Output:

3.2.3: Handling Categorical Data

One-Hot Encode Categorical Columns

```
# Convert all relevant categorical columns into numerical format
df = pd.get_dummies(df, columns=['Access_Code', 'Status_Code', 'Proximity_Category'], drop_first=True)

print("Categorical columns have been one-hot encoded.")
df.head()
```

```
Categorical columns have been one-hot encoded.
```

	Station_ID	Station_Name	Latitude	Longitude	City	State	EV_Level1_EVSE_Num	EV_Level2_EVSE_Num
0	11519	DelhiEV_C	28.44008	110.55212	Delhi	Maharashtra	6.0	12.0
1	11149	ChennaiEV_E	20.34568	71.14465	Delhi	Shanghai	6.0	17.0
2	11354	MumbaiEV_D	44.54654	80.42853	Chennai	Jiangsu	9.0	11.0
3	10057	ChennaiEV_E	22.58293	93.85939	Shanghai	Shanghai	3.0	5.0
4	10152	MumbaiEV_D	43.12574	111.23685	Shanghai	Jiangsu	2.0	7.0

5 rows × 35 columns

- **3.2.4: Aggregation of Data**

  The purpose of aggregation is to create high-level summaries of the data. This technique involves grouping the dataset by a specific category (like `City`) and then calculating summary statistics (such as the mean, sum, or count) for each group. This is not used to train the final model but is a powerful transformation for creating business intelligence reports and gaining key insights during analysis.

Output:

3.2.4: Aggregation of Data

```
# This is Aggregation
city_summary = df.groupby('City').agg(
    Average_Daily_Sessions=('Charging_Sessions_per_Day', 'mean')
)

print("--- Aggregation: City-Level Summary ---")
print(city_summary)
```

```
--- Aggregation: City-Level Summary ---
          Average_Daily_Sessions
City
Chennai                85.955000
Delhi                  86.198992
Mumbai                 85.022113
Nanjing                85.756892
Shanghai               85.975430
```

- **3.2.5: Binning**

  Binning is a technique used to convert a continuous numerical variable into a discrete categorical variable. We group a range of numerical values into "bins" and assign a label to each bin. This can sometimes help the model interpret the data more easily and capture non-linear relationships. Here, we will bin the `Proximity_to_POI_km` feature into categories.

Output:

3.2.5: Binning

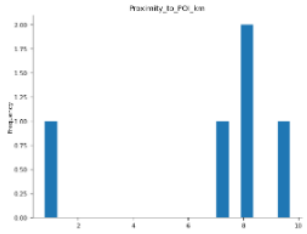
Group 'Proximity_to_POI_km'

```
# Define the bins and labels
bins = [0, 1, 4, float('inf')] # Bins: 0-1km, 1-4km, 4+km
labels = ['Close', 'Nearby', 'Far']

# Create the new binned column
df['Proximity_Category'] = pd.cut(df['Proximity_to_POI_km'], bins=bins, labels=labels, right=False)

print("Created 'Proximity_Category' column.")
df[['Proximity_to_POI_km', 'Proximity_Category']].head()
```

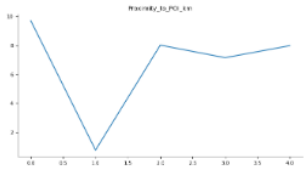
```
Created 'Proximity_Category' column.
```

	Proximity_to_POI_km	Proximity_Category
0	9.692600	Far
1	0.794487	Close
2	8.013062	Far
3	7.146765	Far
4	7.975585	Far

Distributions



Values



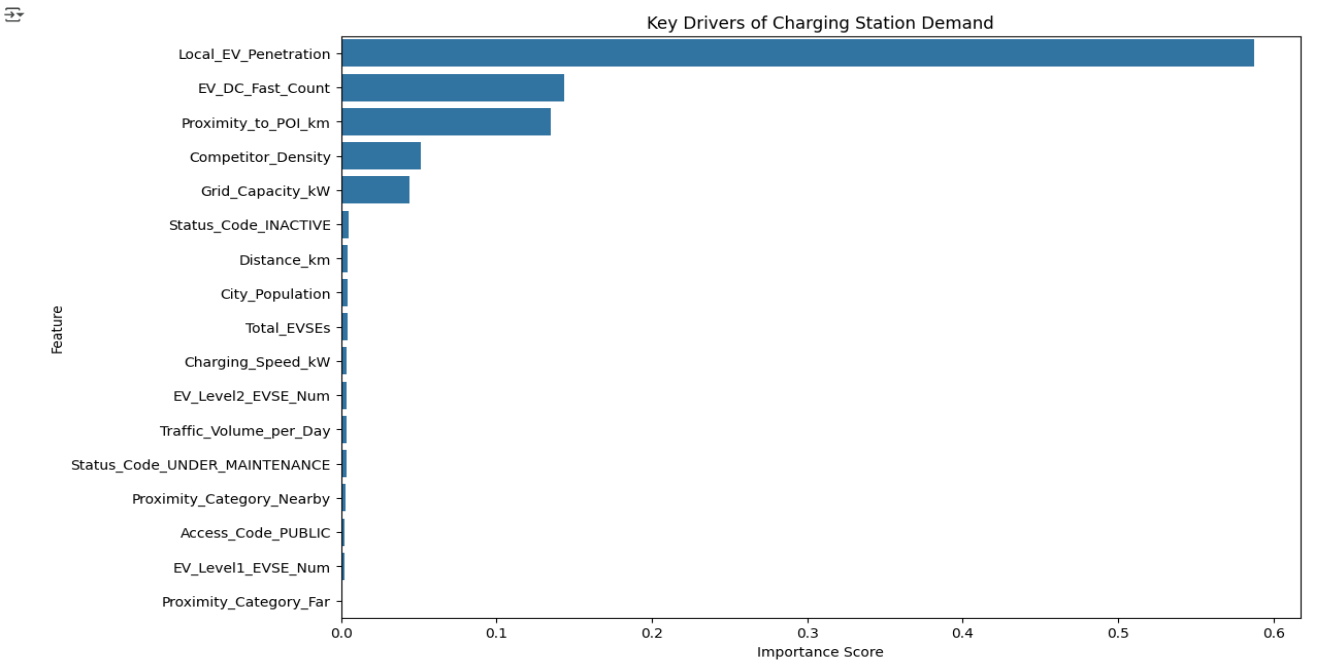
- **3.2.6: Reducing Number of Variables (Feature Selection)**

  This is an optimization step used to simplify a model by removing features that are not useful for prediction. Using too many irrelevant features can add noise and decrease model performance. A common technique is to use a trained model (like XGBoost) to rank features by their importance and then select only the top-ranked ones for the final model.

Output:

3.2.6: Reducing number of variables

```
# Get and visualize the importance of each feature from the trained model
importances = xgb_model.feature_importances_
feature_names = X.columns
importance_df = pd.DataFrame({'feature': feature_names, 'importance': importances}).sort_values('importance', ascending=False)

plt.figure(figsize=(12, 8))
sns.barplot(x='importance', y='feature', data=importance_df)
plt.title('Key Drivers of Charging Station Demand')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.show()
```



- **3.2.7: Extrapolation of Data**
  Extrapolation is a forecasting technique used to estimate new data points that are *outside* the range of your existing data. It helps answer questions about the future by learning from the trends in your past data. For this case study, we can use it to forecast the growth of the EV charging network.
    - **Process Summary:**
        1. **Prepare Time-Series Data:** The Open_Date of the stations was used to calculate the cumulative total number of stations over time, creating a historical growth trend.
        2. **Train a Predictive Model:** A simple LinearRegression model was trained to learn the relationship between time and the total number of stations.
        3. **Extrapolate Future Values:** The trained model was then used to predict the total number of stations for future dates (e.g., the next 24 months), extending the historical trend forward.
    - **Outcome:** The result is a forecast that predicts the total number of EV stations in the future, visualized as a line chart showing the historical trend continuing.

Output:

3.2.7: Extrapolating data

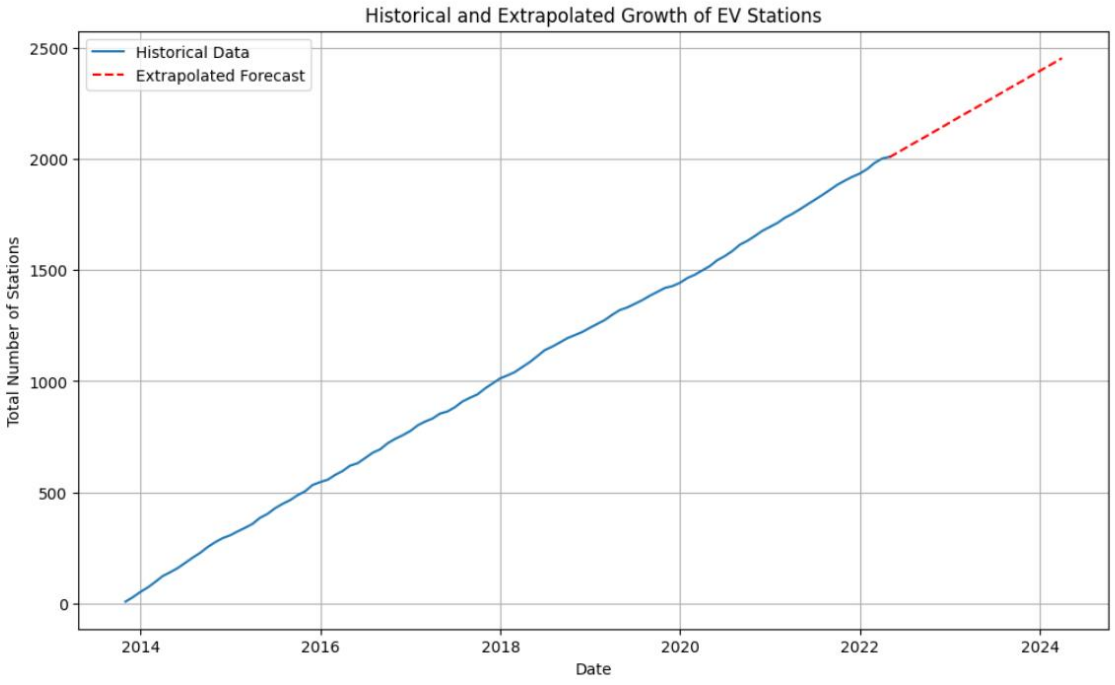
```
df['Open_Date'] = pd.to_datetime(df['Open_Date'])

# Resample the data to get a monthly count of new stations
monthly_counts = df.set_index('Open_Date').resample('M').size()

# Calculate the cumulative sum to get the total number of stations over time
cumulative_stations = monthly_counts.cumsum().reset_index()
cumulative_stations.columns = ['Date', 'Total_Stations']

print("--- Historical Growth of EV Stations ---")
print(cumulative_stations.tail())
```

```
--- Historical Growth of EV Stations ---
          Date  Total_Stations
98  2021-12-31            1935
99  2022-01-31            1956
100 2022-02-28            1982
101 2022-03-31            2002
102 2022-04-30            2010
/tmp/ipython-input-687963167.py:11: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
  monthly_counts = df.set_index('Open_Date').resample('M').size()
```

```
# Prepare the data for the model
# The model needs a numerical input, so we convert dates to timestamps
X = cumulative_stations['Date'].apply(lambda x: x.timestamp()).values.reshape(-1, 1)
y = cumulative_stations['Total_Stations'].values

# Train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Create future dates to predict on (extrapolate for the next 24 months)
last_date = cumulative_stations['Date'].max()
future_dates = pd.date_range(start=last_date, periods=24, freq='M')
future_dates_timestamps = future_dates.to_series().apply(lambda x: x.timestamp()).values.reshape(-1, 1)

# Make the predictions (this is the extrapolation)
future_predictions = model.predict(future_dates_timestamps)

# Create a DataFrame for the extrapolated data
extrapolated_df = pd.DataFrame({'Date': future_dates, 'Forecasted_Stations': future_predictions.astype(int)})

print("\n--- Extrapolated Forecast for the Next 2 Years ---")
print(extrapolated_df)
```

```
--- Extrapolated Forecast for the Next 2 Years ---
         Date  Forecasted_Stations
0  2022-04-30                 2008
1  2022-05-31                 2028
2  2022-06-30                 2047
3  2022-07-31                 2067
4  2022-08-31                 2086
5  2022-09-30                 2105
6  2022-10-31                 2125
7  2022-11-30                 2144
8  2022-12-31                 2164
9  2023-01-31                 2184
10 2023-02-28                 2202
11 2023-03-31                 2221
12 2023-04-30                 2240
13 2023-05-31                 2260
14 2023-06-30                 2279
15 2023-07-31                 2299
16 2023-08-31                 2319
17 2023-09-30                 2338
18 2023-10-31                 2357
19 2023-11-30                 2377
20 2023-12-31                 2396
21 2024-01-31                 2416
22 2024-02-29                 2434
23 2024-03-31                 2454
/tmp/ipython-input-3412289679.py:12: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
  future_dates = pd.date_range(start=last_date, periods=24, freq='M')
```

```python
# Plot the historical data and the extrapolated forecast
plt.figure(figsize=(12, 7))
plt.plot(cumulative_stations['Date'], cumulative_stations['Total_Stations'], label='Historical Data')
plt.plot(extrapolated_df['Date'], extrapolated_df['Forecasted_Stations'], 'r--', label='Extrapolated Forecast')
plt.title('Historical and Extrapolated Growth of EV Stations')
plt.xlabel('Date')
plt.ylabel('Total Number of Stations')
plt.legend()
plt.grid(True)
plt.show()
```



3.3: Combining Data

The goal of combining data is to either enrich your dataset by adding information from other sources or to create specific, focused subsets for targeted analysis.

- **3.3.1: Merging/Joining Datasets**

This technique is used to combine two separate datasets that share a common key (e.g., a `City` column). While we did not use a second dataset in this project, this would be the step to enrich our station data with external information, like city-level economic data.

- **3.3.2: Creating Views**

  This involves filtering a single dataset to create a specific subset or "view." This is useful for focusing your analysis on a particular segment of the data, such as stations in a specific city or only those that are currently active.

Output:

3.3.2: Creating views

Create a View of Active Stations

```
# Create a new DataFrame containing only rows where Status_Code is 'ACTIVE'
active_stations_view = df[df['Status_Code'] == 'ACTIVE'].copy()

print("--- View 1: Active Stations Only ---")
print(f"Total active stations: {len(active_stations_view)}")
print(active_stations_view[['Station_Name', 'City', 'Status_Code']].head())
```

```
--- View 1: Active Stations Only ---
Total active stations: 683
    Station_Name      City Status_Code
0      DelhiEV_C     Delhi      ACTIVE
3    ChennaiEV_E  Shanghai      ACTIVE
4     MumbaiEV_D  Shanghai      ACTIVE
6   ShanghaiEV_B   Nanjing      ACTIVE
12     DelhiEV_C   Nanjing      ACTIVE
```

Create a View of Stations in a Specific City

```
# Create a view for a single city
mumbai_stations_view = df[df['City'] == 'Mumbai'].copy()

print("\n--- View 2: Mumbai Stations Only ---")
print(f"Total stations in Mumbai: {len(mumbai_stations_view)}")
print(mumbai_stations_view[['Station_Name', 'City']].head())
```

```
--- View 2: Mumbai Stations Only ---
Total stations in Mumbai: 406
   Station_Name    City
5   ChennaiEV_E  Mumbai
8   NanjingEV_A  Mumbai
18  NanjingEV_A  Mumbai
26   MumbaiEV_D  Mumbai
28  ChennaiEV_E  Mumbai
```

Create a Complex View with Multiple Conditions

```
# Combine two conditions using the '&' operator
# Note the parentheses around each condition
high_capacity_chennai_view = df[
    (df['City'] == 'Chennai') & (df['Grid_Capacity_kW'] > 100)
].copy()

print("\n--- View 3: High-Capacity Stations in Chennai ---")
print(f"Total high-capacity stations in Chennai: {len(high_capacity_chennai_view)}")
print(high_capacity_chennai_view[['Station_Name', 'City', 'Grid_Capacity_kW']].head())
```

```
--- View 3: High-Capacity Stations in Chennai ---
Total high-capacity stations in Chennai: 282
    Station_Name     City  Grid_Capacity_kW
2     MumbaiEV_D  Chennai               237
17    MumbaiEV_D  Chennai               114
24   ChennaiEV_E  Chennai               208
31   ChennaiEV_E  Chennai               218
32  ShanghaiEV_B  Chennai               163
```

4: Data Exploration (EDA)

EDA is the process of using statistics and visualizations to understand the main characteristics of the data. It's like being a detective looking for clues and patterns before building a case. This step helps form an intuition about the data and guides the later modeling phase.

- **4.1: Nongraphical Techniques**

  This involves understanding the data's basic properties using summary statistics instead of charts. We used `.describe()` to get a quick overview of our numerical data, including the mean, median, and range for each feature. This provided a foundational, quantitative understanding of the dataset.

Output:

4.1: Nongraphical Techniques

```
# Calculate and display summary statistics for all numerical columns
print("--- Descriptive Statistics ---")
print(df.describe())
```

```
--- Descriptive Statistics ---
         Station_ID      Latitude     Longitude  EV_Level1_EVSE_Num  \
count  2.005000e+03  2.005000e+03  2.005000e+03        2.005000e+03
mean  -6.024552e-16  1.151753e-16 -7.211743e-16        1.718769e-16
min   -1.722695e+00 -1.733096e+00 -1.755310e+00       -1.598012e+00
25%   -8.656858e-01 -8.505030e-01 -8.501835e-01       -6.882977e-01
50%   -3.705596e-05 -1.548729e-02  2.954338e-02        2.214168e-01
75%    8.656117e-01  9.032479e-01  8.433961e-01        6.762741e-01
max    1.731260e+00  1.746834e+00  1.713446e+00        1.585989e+00
std    1.000249e+00  1.000249e+00  1.000249e+00        1.000249e+00

       EV_Level2_EVSE_Num  EV_DC_Fast_Count                    Open_Date  \
count        2.005000e+03      2.005000e+03                         2005
mean        -2.480698e-17      2.480698e-17  2018-01-05 08:39:15.710723328
min         -1.609813e+00     -1.491103e+00            2013-10-25 00:00:00
25%         -9.051315e-01     -9.896501e-01            2015-10-25 00:00:00
50%          3.444327e-02      1.325536e-02            2017-12-22 00:00:00
75%          7.391244e-01      1.016161e+00            2020-04-13 00:00:00
max          1.678699e+00      1.517614e+00            2022-04-17 00:00:00
std          1.000249e+00      1.000249e+00                            NaN
```

- **4.2: Simple Graphs**

These are fundamental charts used to visualize one or two variables at a time.

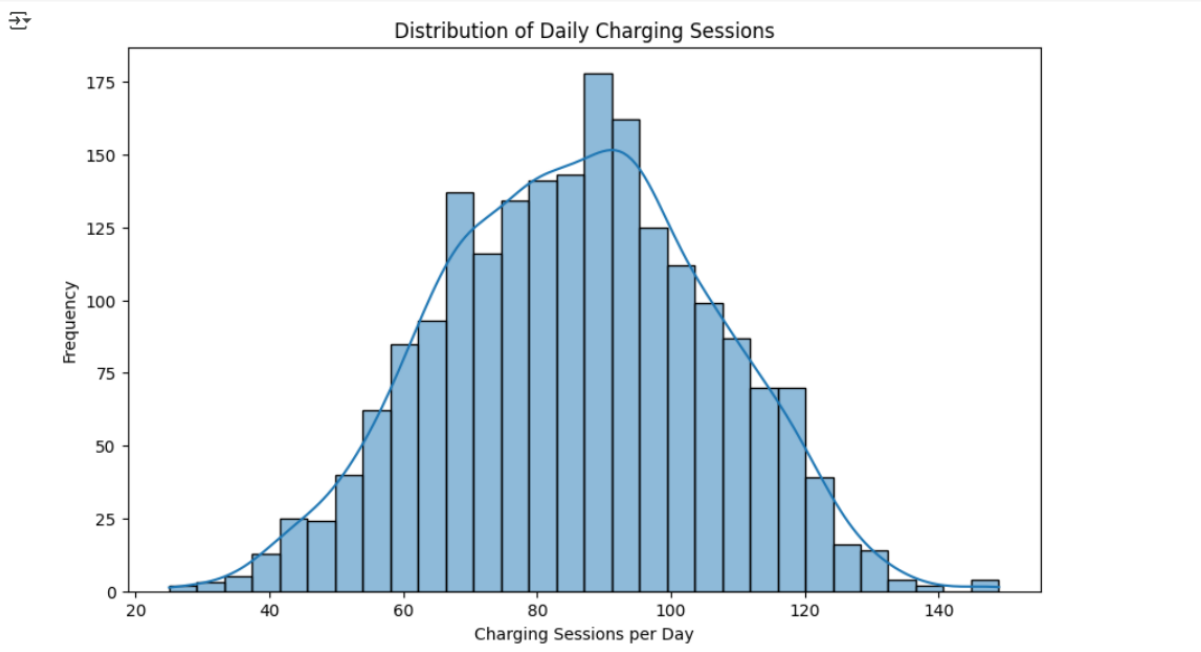
- **Histogram:** We used a histogram to see the distribution of our target variable, `Charging_Sessions_per_Day`. This showed us the frequency and range of daily sessions.
- **Scatter Plot:** We used scatter plots to examine the relationship between a key feature (like `Proximity_to_POI_km`) and our target. This visually confirmed that a strong, predictable relationship existed between them.

Output:

4.2: Simple Graphs

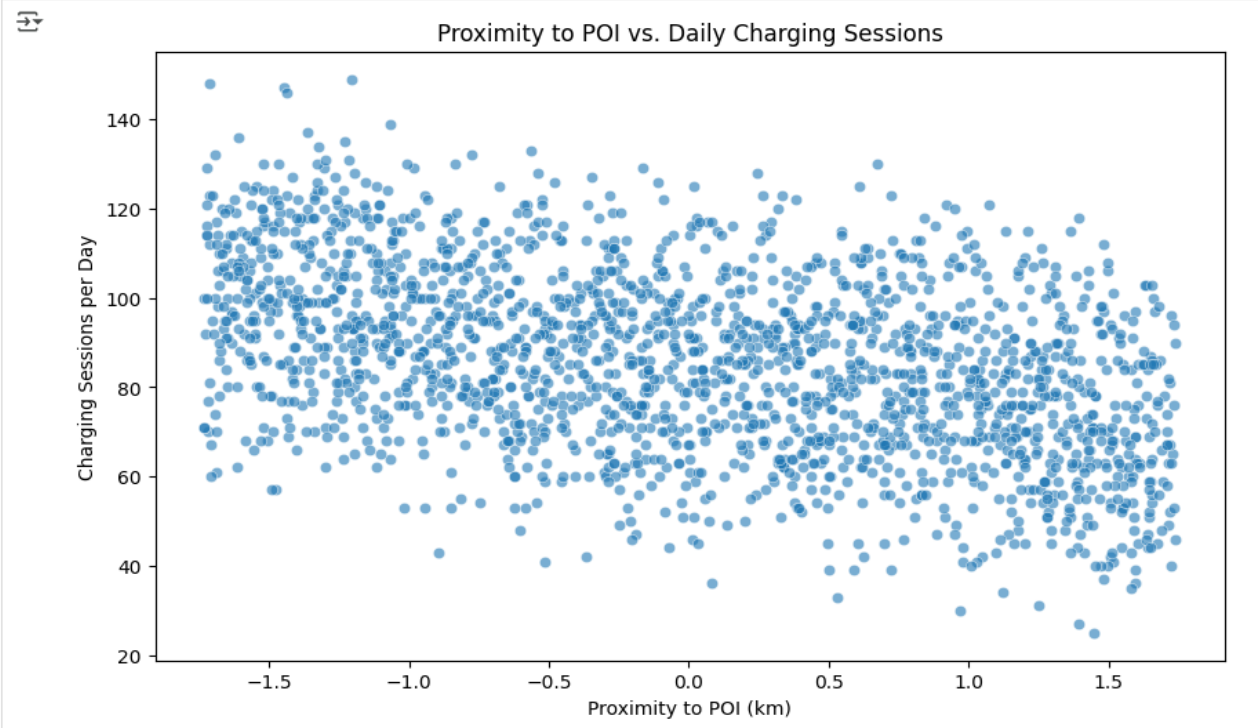
Visualize the Target Variable's Distribution

```
import seaborn as sns
import matplotlib.pyplot as plt

# --- Simple Graph: Histogram ---
plt.figure(figsize=(10, 6))
sns.histplot(df['Charging_Sessions_per_Day'], kde=True, bins=30)
plt.title('Distribution of Daily Charging Sessions')
plt.xlabel('Charging Sessions per Day')
plt.ylabel('Frequency')
plt.show()
```



Visualize a Key Feature vs. the Target

```
# --- Simple Graph: Scatter Plot ---
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Proximity_to_POI_km', y='Charging_Sessions_per_Day', alpha=0.6)
plt.title('Proximity to POI vs. Daily Charging Sessions')
plt.xlabel('Proximity to POI (km)')
plt.ylabel('Charging Sessions per Day')
plt.show()
```



- **4.3: Combined Graphs**

  These are more complex visualizations that show the relationships between many variables at once.

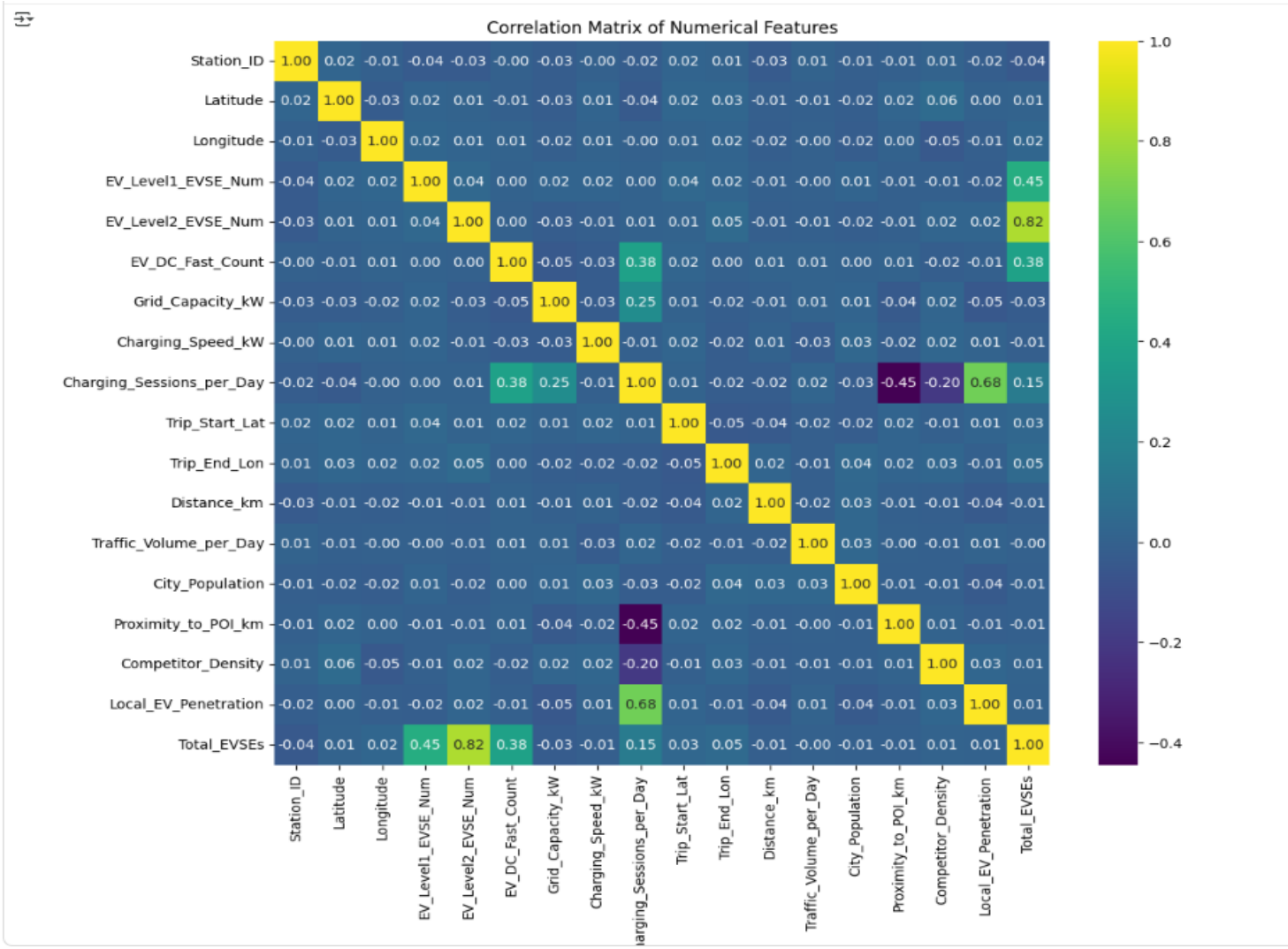
  - **Correlation Heatmap:** We created a heatmap to get a single, "big picture" view of how all numerical variables relate to each other. This was the most effective way to quickly identify which features were most strongly correlated with `Charging_Sessions_per_Day`, confirming they would be excellent predictors for our model.

Output:

4.3: Combined Graphs

Create a Correlation Heatmap

```
# --- Combined Graph: Correlation Heatmap ---

# Select only the numerical columns for the correlation matrix
numerical_cols = df.select_dtypes(include=np.number)

plt.figure(figsize=(12, 10))
sns.heatmap(numerical_cols.corr(), annot=True, cmap='viridis', fmt='.2f')
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



5: Data Modeling

This is the core of the predictive analysis where the project's research goal is directly addressed. In this phase, we use the prepared, high-quality data to train a machine learning model. This model learns the complex patterns and relationships between the station's features and its usage. A successful model provides the ability to make accurate predictions on new, unseen data, allowing the business to forecast demand and make informed decisions about future station placements.

5.1:Model and Variable Selection

Before training a model, we must first clearly define what we are trying to predict (the target variable) and what information we will use to make that prediction (the feature variables). We also need to select an appropriate type of model for the task. This foundational step ensures the model is set up correctly to solve the problem.

- The objective is to predict `Charging_Sessions_per_Day`, which is a continuous numerical value, making this a **regression problem**.
    - **Target Variable (y):** `Charging_Sessions_per_Day`.
    - **Feature Variables (X):** All other relevant, transformed numerical columns in the dataset. Non-predictive columns like IDs, names, and original text fields were excluded.
    - **Model Selection:** The **XGBoost Regressor** was chosen. This is a powerful, industry-standard algorithm known for its high accuracy and efficiency in handling tabular data like ours.

Output:

5.1: Model and Variable Selection

Select Features (X) and Target (y)

```
# X will contain all the numerical, transformed features
# y is the single column we want to predict

# Drop any non-feature columns that are not useful for prediction
features_to_drop = [
    'Station_ID', 'Station_Name', 'Latitude', 'Longitude', 'City', 'State',
    'EV_Connector_Types', 'EV_Pricing', 'Access_Days_Time', 'EV_Network',
    'Owner_Type_Code', 'Open_Date', 'Date_Last_Confirmed', 'Energy_Source',
    'Peak_Hour_Usage', 'Trip_Start_Lat', 'Trip_End_Lon', 'Charging_Sessions_per_Day'
]

X = df.drop(columns=features_to_drop, errors='ignore') # errors='ignore' will not raise an error if a column is already gone
y = df['Charging_Sessions_per_Day']

print("Features (X) and target (y) have been selected.")
```

```
Features (X) and target (y) have been selected.
```

Split Data into Training and Testing Sets

```
from sklearn.model_selection import train_test_split

# Split the data: 80% for training, 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Data split complete: {len(X_train)} training samples and {len(X_test)} testing samples.")
```

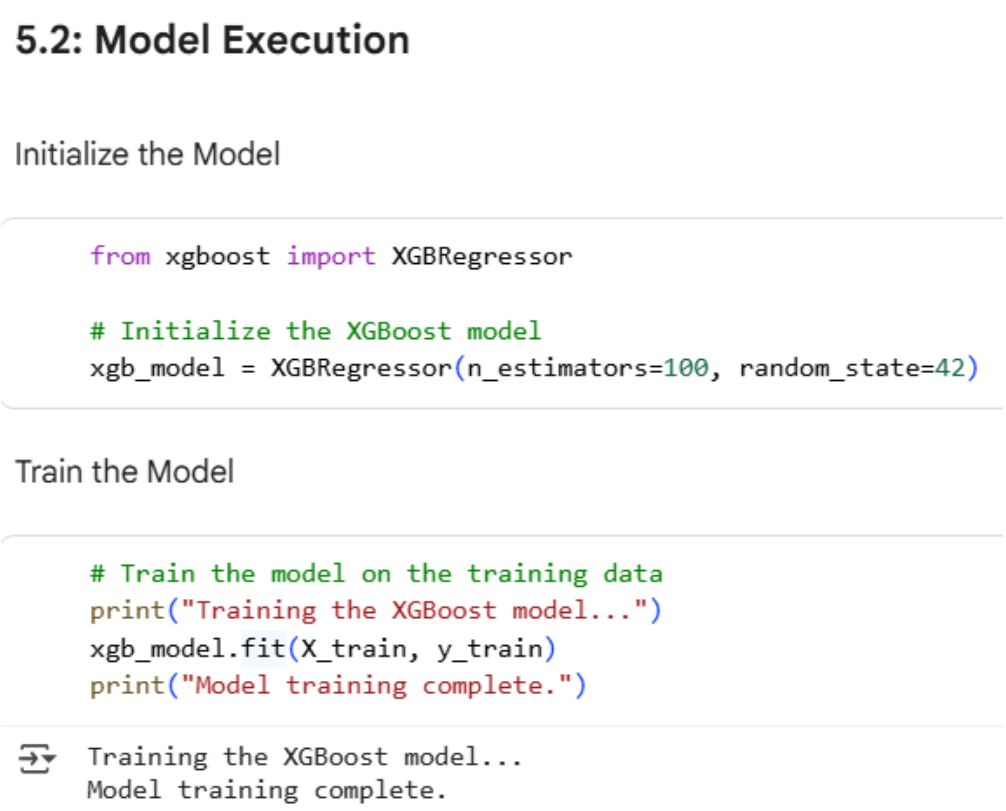
```
Data split complete: 1604 training samples and 401 testing samples.
```

5.2: Model Execution

This is the "learning" phase. The chosen model algorithm is fed the training data (`X_train` and `y_train`). The model iteratively adjusts its internal parameters to find the mathematical patterns that best connect the features to the target variable. A well-executed training phase results in a model that has successfully captured the underlying relationships in the data.

- An instance of the `XGBoost Regressor` is created with standard settings (e.g., 100 estimators). The `.fit()` method is then called, which is the command that initiates the training process.

Output:



5.3: Model Diagnostics and Comparison

A model is only useful if we can prove it is accurate. This diagnostic step evaluates the model's performance on the unseen test data. By comparing the model's predictions to the actual known values, we can calculate objective performance metrics. This validates the model and gives us confidence in its ability to make reliable predictions on new data in the future.

- The trained model is used to make predictions on the `X_test` set. These predictions are then compared to the actual values (`y_test`) using two key regression metrics:
    - **Mean Absolute Error (MAE):** The average absolute difference between the predicted and actual values. A lower MAE is better.
    - **R-squared (R^2):** The proportion of the variance in the target variable that is predictable from the features. A higher R^2 (closer to 1.0) is better.

Output: The final performance scores are printed, confirming the model's high accuracy.

--- XGBoost Model Performance ---

Mean Absolute Error (MAE): 5.02

R-squared (R^2): 0.90

5.3: Model Diagnostics and Comparison

Make Predictions

```
# Use the trained model to make predictions on the test data
y_pred = xgb_model.predict(X_test)
```

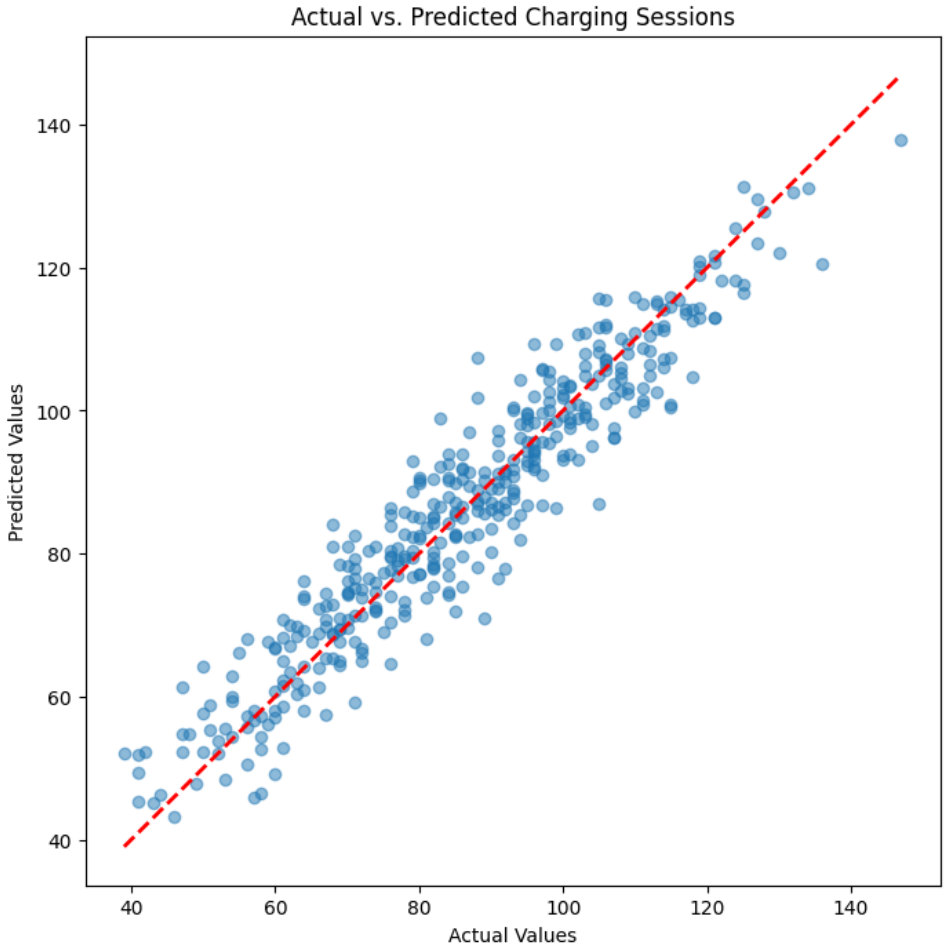
Evaluate Model Performance

```
from sklearn.metrics import mean_absolute_error, r2_score

# Calculate the model's performance metrics
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\n--- XGBoost Model Performance ---")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R-squared (R²): {r2:.2f}")
```

```
--- XGBoost Model Performance ---
Mean Absolute Error (MAE): 5.02
R-squared (R²): 0.90
```

Visualize Predictions vs. Actuals

```
# Create a scatter plot to compare actual vs. predicted values
plt.figure(figsize=(8, 8))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--', lw=2, color='red')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Charging Sessions")
plt.show()
```



6. Presentation and Automation

A successful model is only valuable if its insights can be communicated effectively to stakeholders and integrated into business processes. This final step focuses on translating the technical results into actionable recommendations and creating a repeatable system for future use.

- **Presentation of Key Insights:** The successful model was analyzed to extract key business insights. The primary tool for this was the **feature importance** plot, which

identifies the main drivers of charging station usage. The analysis revealed that the most critical factors are:

1. **High Local EV Penetration**
2. **Close Proximity to Points of Interest (POI)**
3. **Low Competitor Density**

- **Actionable Recommendation:** These insights form the core of the final presentation. The primary recommendation to the business is to adopt a new, data-driven site selection strategy.

  To maximize return on investment and ensure high station utilization, future investments should be prioritized for locations that exhibit the characteristics of high-performing stations: high local EV ownership, close proximity to major hubs like malls and highways, and a low density of nearby competitors.

- **Automation for Deployment:** To make this analysis a reusable asset, the entire workflow—from data cleaning to prediction—is structured into an automated pipeline. This pipeline can be deployed as a business tool to score and rank potential new locations based on the key predictive features, turning the one-time analysis into a continuous, data-driven decision-making process.