Project Name : EV Data Analysis

Name : Manish Tomar

Project Summary : Dataset Summary
The file features columns such as VIN, County, City, State (all WA), Postal Code, Model Year
(ranging from 2016 to 2026 in the sample), Make (e.g., TESLA, AUDI, POLESTAR), Model, Electric Vehicle Type (BEV or PHEV),
CAFV Eligibility, Electric Range (0-322 miles), and geospatial data like Vehicle Location and Census Tract.
Entries show a focus on Tesla models in counties like King and Snohomish, with utilities like Puget Sound Energy.

Objective
This dataset tracks the population and characteristics of electric vehicles to support analysis of EV adoption,
infrastructure needs, policy eligibility (e.g., CAFV), and geographic distribution in Washington state.
It aids research on clean transportation trends, utility planning, and legislative districts.

Key Insights
Tesla dominates registrations, especially in King and Snohomish counties, with models like Model Y and Model 3 showing ranges up to 322 miles and CAFV eligibility. The dataset highlights geographic clustering around Puget Sound Energy areas, aiding infrastructure planning.

Adoption Trends
EV sales reached 20-22% of new vehicles in 2023-2025, exceeding national averages but facing challenges to hit the 35% mandate by 2026. Growth slowed slightly in early 2025 amid protests, yet Washington leads nationally toward 100% zero-emission sales by 2035.

Implications
This data underscores accelerating EV uptake for emissions reduction, though charging reliability and policy compliance remain critical hurdles. Continued tracking will inform utilities, legislators, and planners on equitable expansion.

Github link : https://github.com/Moni0802/EV_Data_Analysis

IMPORT LIBRARIES

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib as plt
```

Data Import

In [2]:

```
df = pd.read_csv('D:\manish\Python\GROW AI\python\Electric_Vehicle_Population_Data.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	E
0	5YJYGDEE8L	Thurston	Tumwater	WA	98501.0	2020	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	
1	5YJXCAE2XJ	Snohomish	Bothell	WA	98021.0	2018	TESLA	MODEL X	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	
2	5YJ3E1EBXK	King	Kent	WA	98031.0	2019	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	
3	7SAYGDEE4T	King	Issaquah	WA	98027.0	2026	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	
4	WAUUPBFF9G	King	Seattle	WA	98103.0	2016	AUDI	A3	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	

In [4]:

```
df.shape
```

Out[4]:
```
(270262, 16)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['VIN (1-10)', 'County', 'City', 'State', 'Postal Code', 'Model Year',
       'Make', 'Model', 'Electric Vehicle Type',
       'Clean Alternative Fuel Vehicle (CAFV) Eligibility', 'Electric Range',
       'Legislative District', 'DOL Vehicle ID', 'Vehicle Location',
       'Electric Utility', '2020 Census Tract'],
      dtype='object')
```

In [6]:
```
df.dtypes
```

Out[6]:
```
VIN (1-10)                                            object
County                                                object
City                                                  object
State                                                 object
Postal Code                                          float64
Model Year                                             int64
Make                                                  object
Model                                                 object
Electric Vehicle Type                                 object
Clean Alternative Fuel Vehicle (CAFV) Eligibility     object
Electric Range                                       float64
Legislative District                                 float64
DOL Vehicle ID                                         int64
Vehicle Location                                      object
Electric Utility                                      object
2020 Census Tract                                    float64
dtype: object
```

In [7]:
```
df
```

Out[7]:

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type
0	5YJYGDEE8L	Thurston	Tumwater	WA	98501.0	2020	TESLA	MODEL Y	Battery Electric Vehicle (BEV)
1	5YJXCAE2XJ	Snohomish	Bothell	WA	98021.0	2018	TESLA	MODEL X	Battery Electric Vehicle (BEV)
2	5YJ3E1EBXK	King	Kent	WA	98031.0	2019	TESLA	MODEL 3	Battery Electric

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type
									Vehicle (BEV)
3	7SAYGDEE4T	King	Issaquah	WA	98027.0	2026	TESLA	MODEL Y	Battery Electric Vehicle (BEV)
4	WAUUPBFF9G	King	Seattle	WA	98103.0	2016	AUDI	A3	Plug-in Hybrid Electric Vehicle (PHEV)
...	...	...	...	...	...	...	...	...	...
270257	1C4RJXN60R	Pierce	Joint Base Lewis Mcchord	WA	98433.0	2024	JEEP	WRANGLER	Plug-in Hybrid Electric Vehicle (PHEV)
270258	1C4JJXR66N	Mason	Hoodsport	WA	98548.0	2022	JEEP	WRANGLER	Plug-in Hybrid Electric Vehicle (PHEV)
270259	7SAYGDEEXP	Pierce	Tacoma	WA	98406.0	2023	TESLA	MODEL Y	Battery Electric Vehicle (BEV)
270260	5YJYGDEE2M	Snohomish	Bothell	WA	98021.0	2021	TESLA	MODEL Y	Battery Electric Vehicle (BEV)
270261	JN1BF0BA5P	Chelan	Wenatchee	WA	98801.0	2023	NISSAN	ARIYA HATCHBACK	Battery Electric Vehicle (BEV)

270262 rows × 16 columns

Data Cleaning

How many missing values exist in the dataset, and in which columns?

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270262 entries, 0 to 270261
Data columns (total 16 columns):
 #   Column                                             Non-Null Count   Dtype  
---  ------                                             --------------   -----  
 0   VIN (1-10)                                         270262 non-null  object 
 1   County                                             270252 non-null  object 
 2   City                                               270252 non-null  object 
 3   State                                              270262 non-null  object 
 4   Postal Code                                        270252 non-null  float64
 5   Model Year                                         270262 non-null  int64  
 6   Make                                               270262 non-null  object 
 7   Model                                              270262 non-null  object 
 8   Electric Vehicle Type                              270262 non-null  object 
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  270262 non-null  object 
 10  Electric Range                                     270257 non-null  float64
 11  Legislative District                               269613 non-null  float64
 12  DOL Vehicle ID                                     270262 non-null  int64  
 13  Vehicle Location                                   270174 non-null  object 
 14  Electric Utility                                   270252 non-null  object 
 15  2020 Census Tract                                  270252 non-null  float64
dtypes: float64(4), int64(2), object(10)
memory usage: 33.0+ MB
```

In [9]:

```
df.isnull().sum()
```

Out[9]:

```
VIN (1-10)                                             0
County                                                10
City                                                  10
State                                                  0
Postal Code                                           10
Model Year                                             0
Make                                                   0
Model                                                  0
Electric Vehicle Type                                  0
Clean Alternative Fuel Vehicle (CAFV) Eligibility      0
Electric Range                                         5
Legislative District                                 649
DOL Vehicle ID                                         0
Vehicle Location                                      88
Electric Utility                                      10
2020 Census Tract                                     10
dtype: int64
```

In Legislative District column we have more null values as compared to other columns i.e 649

Second column is Vehicle Location which is having 88 null values

In [10]:

```
df.isnull().sum().sum()
```

Out[10]:

```
np.int64(792)
```

Total number of null values present in our dataset is 792.

How should missing or zero values in the Base MSRP and Electric Range columns be handled?

In [11]:

```
df.head()
```

Out[11]:

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	E
0	5YJYGDEE8L	Thurston	Tumwater	WA	98501.0	2020	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	
1	5YJXCAE2XJ	Snohomish	Bothell	WA	98021.0	2018	TESLA	MODEL X	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	
2	5YJ3E1EBXK	King	Kent	WA	98031.0	2019	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	
3	7SAYGDEE4T	King	Issaquah	WA	98027.0	2026	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	
4	WAUUPBFF9G	King	Seattle	WA	98103.0	2016	AUDI	A3	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	

In [12]:

```
df['Electric Range'].unique()
```

Out[12]:

```
array([291., 238., 220.,   0.,  16., 215., 322.,  35.,  84.,  33., 208.,
        32.,  53., 266., 293., 125., 150.,  38.,  14.,  73., 107.,  39.,
        23., 239.,  40.,  37.,  34., 234.,  26.,  18.,  93., 153.,  87.,
        30., 249.,  19., 210.,  25.,  82., 308.,  17., 289.,  75.,  72.,
       103.,  97.,  42.,  21., 149.,  22.,  47., 259., 204.,  11., 126.,
        81., 203., 200., 151.,  27.,  13.,   6.,  83.,  28.,  29., 330.,
```

```
       110., 258., 337.,  15., 114., 222.,  20.,  12., 270., 265.,  62.,
       192., 233.,  48., 100.,  54., 170., 218.,  41.,  49.,  68.,  58.,
        43.,  76.,  10.,  60.,  36., 111., 245.,   8., 288.,  24.,  nan,
       124.,   9.,  59.,  44.,   1.,  55.,  56.,  46.,  31.,  51.,  95.,
        45.,  57.,  50.,  74.])
```

In [13]:

```
df.nunique()
```

Out[13]:

```
VIN (1-10)                                            16415
County                                                  242
City                                                    864
State                                                    51
Postal Code                                            1083
Model Year                                               22
Make                                                     47
Model                                                   183
Electric Vehicle Type                                     2
Clean Alternative Fuel Vehicle (CAFV) Eligibility         3
Electric Range                                          113
Legislative District                                     49
DOL Vehicle ID                                       270262
Vehicle Location                                       1080
Electric Utility                                         77
2020 Census Tract                                      2336
dtype: int64
```

Are there duplicate records in the dataset? If so, how should they be managed?

In [14]:

```
# Check duplicates by VIN
duplicates = df[df.duplicated(subset=["VIN (1-10)", "DOL Vehicle ID"], keep=False)]
```

In [15]:

```
duplicates
```

Out[15]:

VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	Electric Range	Legislative District	Veh

In [16]:

```
# Drop exact duplicates
df_cleaned = df.drop_duplicates(subset=["VIN (1-10)", "DOL Vehicle ID"], keep="first")
```

In [17]:

```
df_cleaned
```

Out[17]:

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type
0	5YJYGDEE8L	Thurston	Tumwater	WA	98501.0	2020	TESLA	MODEL Y	Battery Electric Vehicle (BEV)
1	5YJXCAE2XJ	Snohomish	Bothell	WA	98021.0	2018	TESLA	MODEL X	Battery Electric Vehicle (BEV)
2	5YJ3E1EBXK	King	Kent	WA	98031.0	2019	TESLA	MODEL 3	Battery Electric Vehicle (BEV)
3	7SAYGDEE4T	King	Issaquah	WA	98027.0	2026	TESLA	MODEL Y	Battery Electric Vehicle (BEV)
4	WAUUPBFF9G	King	Seattle	WA	98103.0	2016	AUDI	A3	Plug-in Hybrid Electric Vehicle (PHEV)
...	...	...	...	...	...	...	...	...	...
270257	1C4RJXN60R	Pierce	Joint Base Lewis Mcchord	WA	98433.0	2024	JEEP	WRANGLER	Plug-in Hybrid Electric Vehicle (PHEV)
270258	1C4JJXR66N	Mason	Hoodsport	WA	98548.0	2022	JEEP	WRANGLER	Plug-in Hybrid Electric Vehicle (PHEV)
270259	7SAYGDEEXP	Pierce	Tacoma	WA	98406.0	2023	TESLA	MODEL Y	Battery Electric Vehicle (BEV)
270260	5YJYGDEE2M	Snohomish	Bothell	WA	98021.0	2021	TESLA	MODEL Y	Battery Electric Vehicle (BEV)
270261	JN1BF0BA5P	Chelan	Wenatchee	WA	98801.0	2023	NISSAN	ARIYA HATCHBACK	Battery Electric Vehicle (BEV)

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type

270262 rows × 16 columns

In [18]:

```
total_duplicates = df.duplicated().sum()
```

In [19]:

```
total_duplicates
```

Out[19]:

```
np.int64(0)
```

In [20]:

```
# Count duplicates based on VIN and DOL Vehicle ID (unique identifiers)
vin_duplicates = df.duplicated(subset=["VIN (1-10)", "DOL Vehicle ID"]).sum()
print("Duplicate rows based on VIN and DOL Vehicle ID:", vin_duplicates)
```

```
Duplicate rows based on VIN and DOL Vehicle ID: 0
```

There are no duplicates if there would have been any duplicates then the below code will work:

df_cleaned = df_cleaned.groupby(["VIN (1-10)", "DOL Vehicle ID"]).apply(lambda x: x.ffill().bfill()).drop_duplicates()

How can VINs be anonymized while maintaining uniqueness?

VINs are unique identifiers for vehicles. If two rows share the same VIN (first 10 characters), they likely represent the same vehicle.

In [21]:

```
# Use a hashing algorithm (like SHA256 or MD5) to convert VINs into anonymized codes.
#This guarantees uniqueness but makes it impossible to reverse-engineer the original VIN

import hashlib
# Function to hash VIN
def anonymize_vin(vin):
    return hashlib.sha256(vin.encode()).hexdigest()[:10]  # shorten to 10 chars

# Apply anonymization
df["VIN_Anon"] = df["VIN (1-10)"].apply(anonymize_vin)

print(df[["VIN (1-10)", "VIN_Anon"]].head())
```

```
   VIN (1-10)    VIN_Anon
0  5YJYGDEE8L  f6c53bc06f
1  5YJXCAE2XJ  6222905c9f
2  5YJ3E1EBXK  78953a9f9d
3  7SAYGDEE4T  9a118e6006
4  WAUUPBFF9G  dbd150d329
```

How can Vehicle Location (GPS coordinates) be cleaned or converted for better readability?

In [22]:

```
lambda x: pd.Series(get_location(float(x.split(",")[1]), float(x.split(",")[0])))
```

Out[22]:

```
<function __main__.<lambda>(x)>
```

In [23]:

```
# First, install the geopy library
# !pip install geopy

# import the library
from geopy.geocoders import Nominatim
import pandas as pd  # Added import for pd

# Initialize geocoder
geolocator = Nominatim(user_agent="ev_analysis")

# Function to get city/county from coordinates
def get_location(lat, lon):
    try:
        location = geolocator.reverse((lat, lon), exactly_one=True)
        if location and location.raw.get("address"):
            address = location.raw["address"]
            city = address.get("city", address.get("town", address.get("village", "")))
            county = address.get("county", "")
            return city, county
    except:
        return None, None

# Modified function to safely parse location
def parse_location(loc_str):
    try:
        # Check if the string contains a comma and can be split
        if isinstance(loc_str, str) and "," in loc_str:
            parts = loc_str.split(",")
            if len(parts) >= 2:
                # Try to convert to float
                lon = float(parts[0].strip())
                lat = float(parts[1].strip())
                return get_location(lat, lon)
    except Exception as e:
        pass
    # Return None values if any error occurs
    return None, None

# Apply function with error handling
df[["City_Cleaned", "County_Cleaned"]] = df["Vehicle Location"].apply(
    lambda x: pd.Series(parse_location(x))
)

print(df[["Vehicle Location", "City_Cleaned", "County_Cleaned"]].head())
```

```
              Vehicle Location City_Cleaned County_Cleaned
0  POINT (-122.89165 47.03954)         None           None
```

```
1   POINT (-122.18384 47.8031)         None           None
2  POINT (-122.17743 47.41185)         None           None
3   POINT (-122.03439 47.5301)         None           None
4  POINT (-122.35436 47.67596)         None           None
```

Data Exploration

What are the top 5 most common EV makes and models in the dataset?

In [24]:

```
# Count frequency of each Make and Model combination
top_ev = df.groupby(["Make", "Model"]).size().reset_index(name="Count")

# Sort by count and get top 5
top_5_ev = top_ev.sort_values(by="Count", ascending=False).head(5)

print(top_5_ev)
```

```
          Make     Model  Count
159      TESLA   MODEL Y  57335
156      TESLA   MODEL 3  37413
136     NISSAN      LEAF  13503
157      TESLA   MODEL S   7758
48   CHEVROLET   BOLT EV   7708
```

In [25]:

```
import pandas as pd
# load data
df = pd.read_csv('D:\manish\Python\GROW AI\python\Electric_Vehicle_Population_Data.csv')

top_ev = df.groupby(['Make','Model']).size().reset_index(name='count')
```

In [26]:

```
top_ev
```

Out[26]:

	Make	Model	count
0	ACURA	ZDX	352
1	ALFA ROMEO	TONALE	100
2	AUDI	A3	533
3	AUDI	A6	49
4	AUDI	A7 E	12
...	...	...	...
178	VOLVO	V60	101
179	VOLVO	XC40	1387
180	VOLVO	XC60	1866
181	VOLVO	XC90	2313
182	WHEEGO ELECTRIC CARS	WHEEGO	2

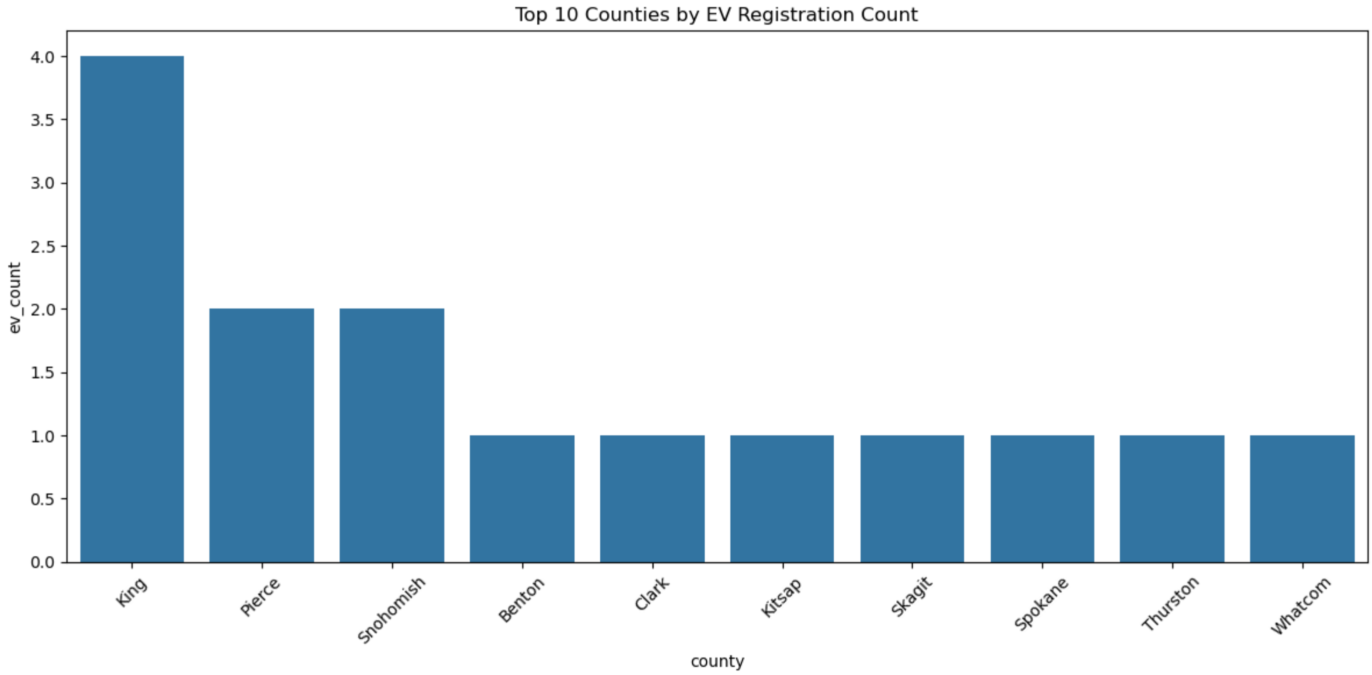
183 rows × 3 columns

What is the distribution of EVs by county? Which county has the most registrations

In [27]:

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.DataFrame({
    'county': ['King', 'Pierce', 'Snohomish', 'Clark', 'Spokane',
               'Thurston', 'Kitsap', 'Whatcom', 'Benton', 'Skagit',
               'King', 'Pierce', 'King', 'Snohomish', 'King'],
    'registration_id': range(1, 16)
})

# Group by county and count registrations
ev_by_county = df.groupby('county')['registration_id'].count().reset_index()
ev_by_county = ev_by_county.rename(columns={'registration_id': 'ev_count'})

# Sort by count in descending order
ev_by_county = ev_by_county.sort_values('ev_count', ascending=False)

# Display the county with the most registrations
print(f"County with most EV registrations: {ev_by_county.iloc[0]['county']} with {ev_by__

# Create a bar chart of the distribution
plt.figure(figsize=(12, 6))
sns.barplot(x='county', y='ev_count', data=ev_by_county.head(10))
plt.title('Top 10 Counties by EV Registration Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

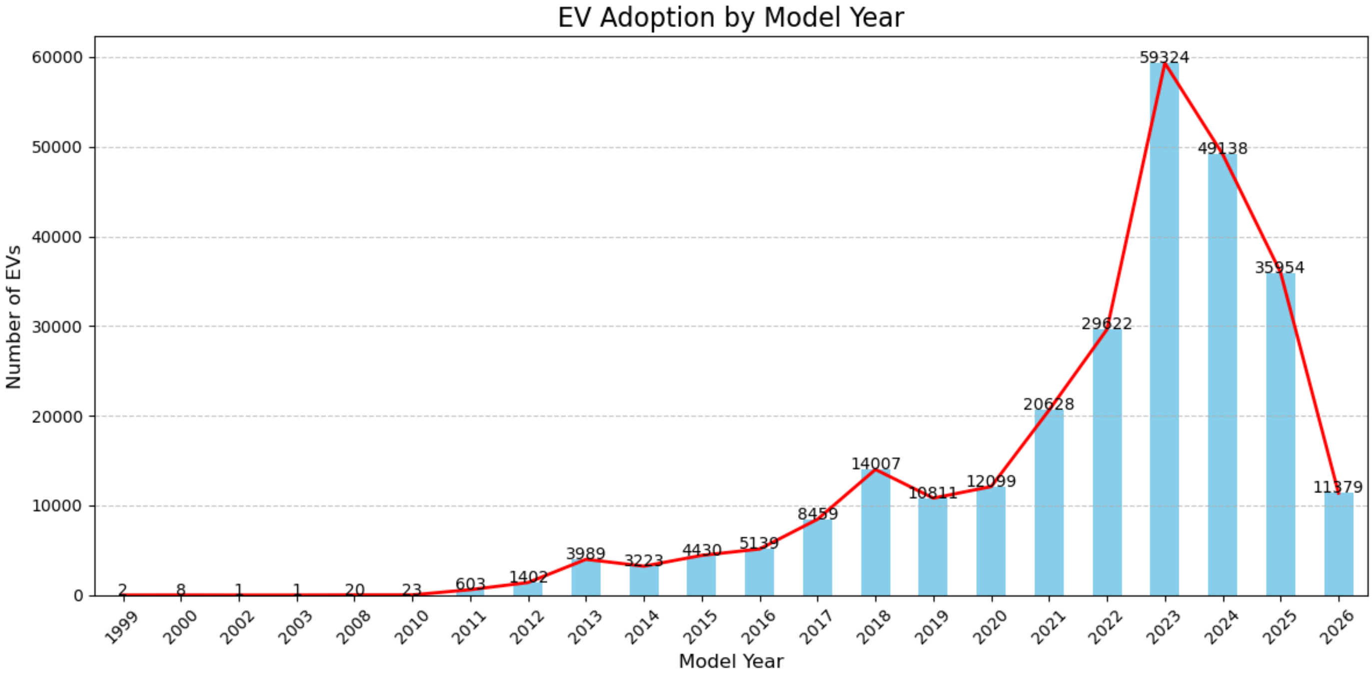
```
County with most EV registrations: King with 4 vehicles
```



How has EV adoption changed over different model years?

In [28]:

```
df1 = pd.read_csv('D:\manish\Python\GROW AI\python\Electric_Vehicle_Population_Data.csv'
# Count the number of EVs by model year
ev_by_year = df1['Model Year'].value_counts().sort_index()

# Create a bar chart to visualize the trend
plt.figure(figsize=(12, 6))
ev_by_year.plot(kind='bar', color='skyblue')
plt.title('EV Adoption by Model Year', fontsize=16)
plt.xlabel('Model Year', fontsize=12)
plt.ylabel('Number of EVs', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=45)

# Add a trend line
plt.plot(range(len(ev_by_year)), ev_by_year.values, 'r-', linewidth=2)

# Add value labels on top of each bar
for i, v in enumerate(ev_by_year):
    plt.text(i, v + 5, str(v), ha='center')

plt.tight_layout()
plt.show()

# Print summary statistics
print("EV Adoption by Model Year:")
print(ev_by_year)

# Calculate year-over-year growth rates
yoy_growth = ev_by_year.pct_change() * 100
print("\nYear-over-Year Growth Rate (%):")
print(yoy_growth.dropna())

# Calculate the compound annual growth rate (CAGR)
if len(ev_by_year) > 1:
    years = len(ev_by_year) - 1
    first_year_count = ev_by_year.iloc[0]
    last_year_count = ev_by_year.iloc[-1]
    if first_year_count > 0:  # Avoid division by zero
        cagr = ((last_year_count / first_year_count) ** (1 / years) - 1) * 100
        print(f"\nCompound Annual Growth Rate (CAGR): {cagr:.2f}%")
```




```
EV Adoption by Model Year:
Model Year
1999        2
2000        8
2002        1
2003        1
2008       20
2010       23
2011      603
2012     1402
2013     3989
2014     3223
2015     4430
2016     5139
2017     8459
2018    14007
2019    10811
2020    12099
2021    20628
2022    29622
2023    59324
2024    49138
2025    35954
2026    11379
Name: count, dtype: int64

Year-over-Year Growth Rate (%):
Model Year
2000     300.000000
2002     -87.500000
2003       0.000000
2008    1900.000000
2010      15.000000
2011    2521.739130
2012     132.504146
2013     184.522111
2014     -19.202808
2015      37.449581
```

```
2016     16.004515
2017     64.604009
2018     65.586949
2019    -22.817163
2020     11.913792
2021     70.493429
2022     43.600931
2023    100.270070
2024    -17.170117
2025    -26.830559
2026    -68.351227
Name: count, dtype: float64

Compound Annual Growth Rate (CAGR): 50.94%
```

What is the average electric range of EVs in the dataset?

In [29]:

```
df1 = pd.read_csv('D:\manish\Python\GROW AI\python\Electric_Vehicle_Population_Data.csv'
```

In [30]:

```
# Calculating the average electric range
average_range = df1['Electric Range'].mean()

# Displaying the result
print(f"The average electric range of EVs in the dataset is {average_range:.2f} miles")
```

```
The average electric range of EVs in the dataset is 40.39 miles
```

What percentage of EVs are eligible for Clean Alternative Fuel Vehicle (CAFV) incentives?

In [31]:

```
# Check the unique values in the CAFV eligibility column
print("Unique values in CAFV Eligibility column:", df1['Clean Alternative Fuel Vehicle (

# Count vehicles eligible for CAFV incentives (assuming 'Eligible' or similar value indi
# Replace 'Eligible' with the actual value that indicates eligibility in dataset
eligible_value = 'Clean Alternative Fuel Vehicle Eligible'
cafv_eligible = df1['Clean Alternative Fuel Vehicle (CAFV) Eligibility'].eq(eligible_val

# Calculate the total number of vehicles
total_vehicles = len(df1)

# Calculate the percentage
cafv_percentage = (cafv_eligible / total_vehicles) * 100

# Display the result
print(f"Percentage of EVs eligible for CAFV incentives: {cafv_percentage:.2f}%")
```

```
Unique values in CAFV Eligibility column: ['Clean Alternative Fuel Vehicle Eligible'
 'Eligibility unknown as battery range has not been researched'
 'Not eligible due to low battery range']
Percentage of EVs eligible for CAFV incentives: 28.25%
```

How does the electric range vary across different makes and models?

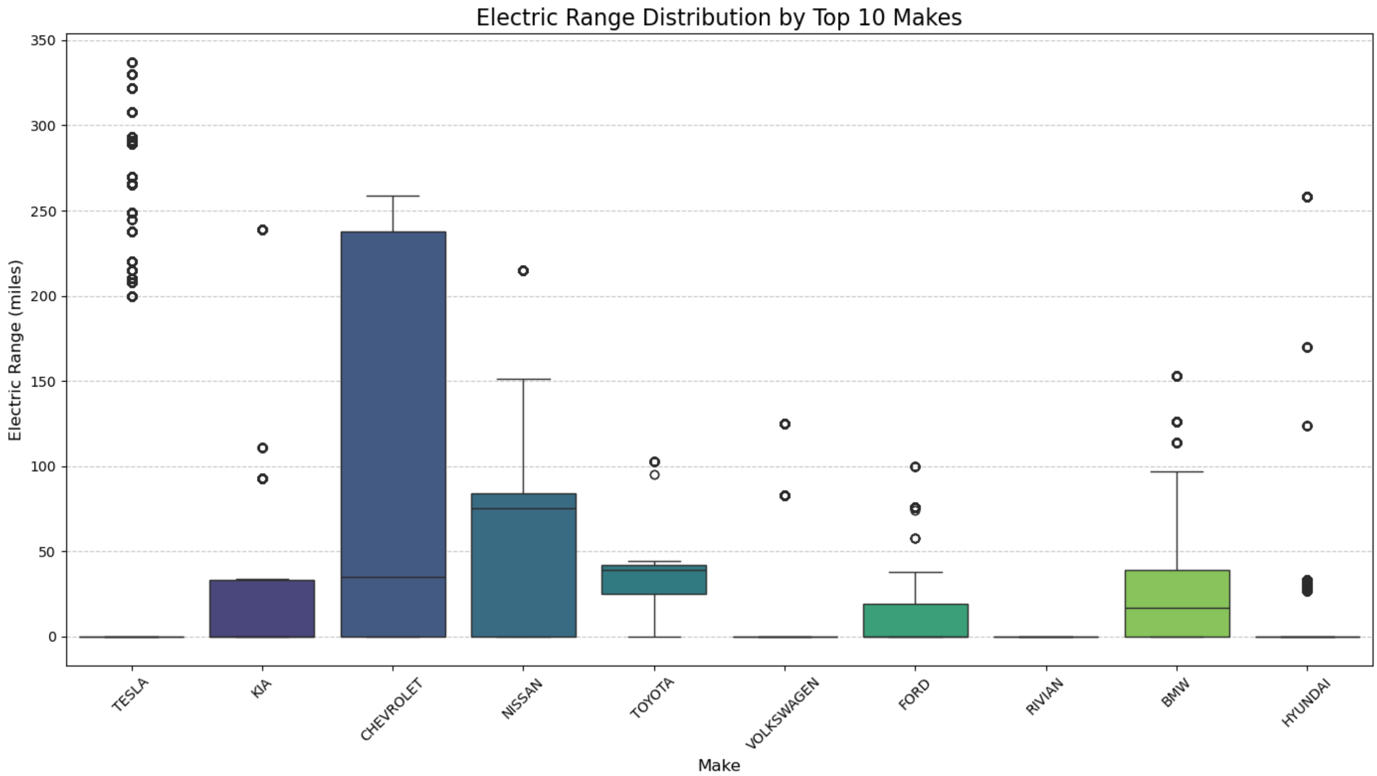
In [32]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# 1. Basic statistics by make
make_range_stats = df1.groupby('Make')['Electric Range'].agg(['mean', 'median', 'min', '

# Display the top 10 makes by average electric range
print("Top 10 Makes by Average Electric Range:")
print(make_range_stats.head(10))

# 2. Create a boxplot to visualize range distribution by make
plt.figure(figsize=(14, 8))
top_makes = df1['Make'].value_counts().head(10).index.tolist()
sns.boxplot(x='Make', y='Electric Range', data=df1[df1['Make'].isin(top_makes)], palette
plt.title('Electric Range Distribution by Top 10 Makes', fontsize=16)
plt.xlabel('Make', fontsize=12)
plt.ylabel('Electric Range (miles)', fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# 3. Top models by electric range
top_models = df1.groupby(['Make', 'Model'])['Electric Range'].mean().sort_values(ascendi
print("\nTop 15 Models by Average Electric Range:")
print(top_models)

# 4. Visualize top models by range
plt.figure(figsize=(14, 8))
top_models.plot(kind='bar', color='skyblue')
plt.title('Top 15 EV Models by Electric Range', fontsize=16)
plt.xlabel('Make and Model', fontsize=12)
plt.ylabel('Average Electric Range (miles)', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=45, ha='right')

# Add value labels on top of each bar
for i, v in enumerate(top_models):
    plt.text(i, v + 5, f"{v:.1f}", ha='center')

plt.tight_layout()
plt.show()

# 5. Create a heatmap for makes with multiple models
# Get makes with at least 2 models
makes_with_multiple_models = df1.groupby('Make')['Model'].nunique()
makes_with_multiple_models = makes_with_multiple_models[makes_with_multiple_models >= 2]

if makes_with_multiple_models:
    # Filter for these makes
    multi_model_data = df1[df1['Make'].isin(makes_with_multiple_models)]

    # Create pivot table: Make vs Model with Electric Range as values
    pivot_data = multi_model_data.pivot_table(
        index='Make',
        columns='Model',
        values='Electric Range',
```

```
        aggfunc='mean'
    )

    # Create heatmap
    plt.figure(figsize=(16, 10))
    sns.heatmap(pivot_data, annot=True, cmap='YlGnBu', fmt='.1f', linewidths=.5)
    plt.title('Average Electric Range by Make and Model', fontsize=16)
    plt.tight_layout()
    plt.show()
```

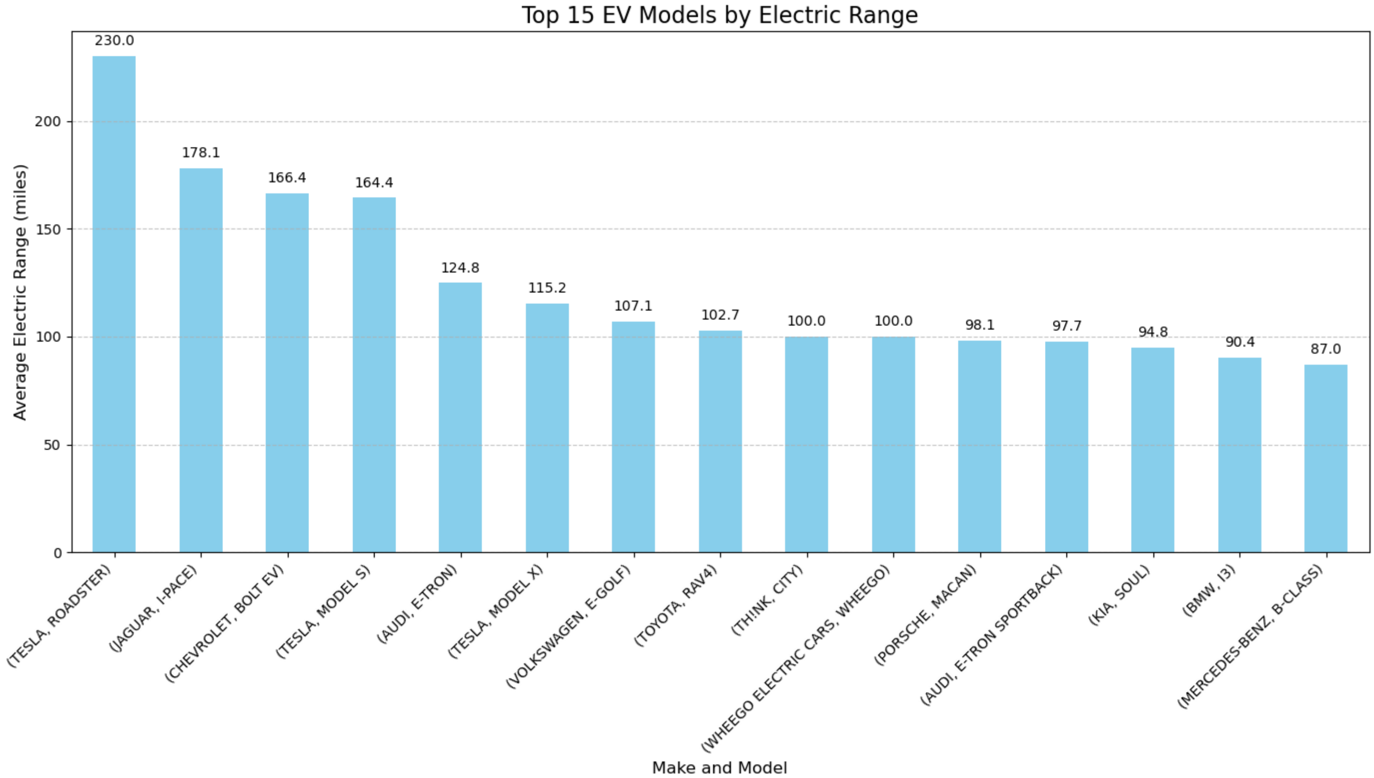
```
Top 10 Makes by Average Electric Range:
                            mean  median    min    max   count
Make
JAGUAR                178.100000   234.0    0.0  234.0     180
WHEEGO ELECTRIC CARS  100.000000   100.0  100.0  100.0       2
TH!NK                 100.000000   100.0  100.0  100.0       6
CHEVROLET              79.141709    35.0    0.0  259.0   19032
FIAT                   74.274533    84.0    0.0   87.0     856
NISSAN                 64.576207    75.0    0.0  215.0   15963
SMART                  61.750000    58.0    0.0   68.0     232
AZURE DYNAMICS         56.000000    56.0   56.0   56.0       4
TESLA                  53.120082     0.0    0.0  337.0  111049
PORSCHE                49.699374    13.0    0.0  308.0    1916
```

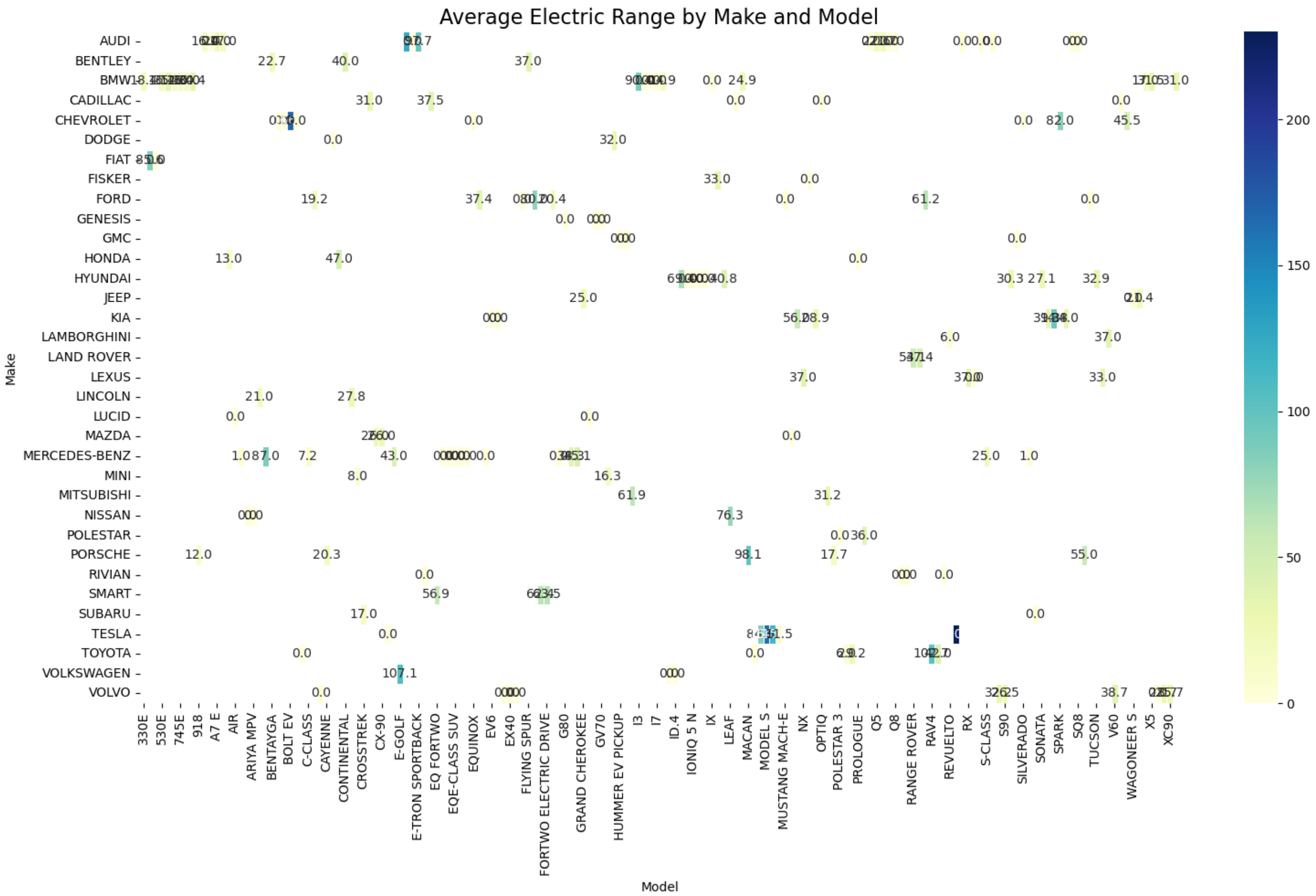
```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_1800\1105207110.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='Make', y='Electric Range', data=df1[df1['Make'].isin(top_makes)], palet
te='viridis')
```




```
Top 15 Models by Average Electric Range:
Make                  Model
TESLA                 ROADSTER            230.000000
JAGUAR                I-PACE              178.100000
CHEVROLET             BOLT EV             166.381681
```

```
TESLA                 MODEL S              164.350606
AUDI                  E-TRON               124.837093
TESLA                 MODEL X              115.182155
VOLKSWAGEN            E-GOLF               107.057471
TOYOTA                RAV4                 102.709091
TH!NK                 CITY                 100.000000
WHEEGO ELECTRIC CARS  WHEEGO               100.000000
PORSCHE               MACAN                 98.143791
AUDI                  E-TRON SPORTBACK      97.724138
KIA                   SOUL                  94.819320
BMW                   I3                    90.417289
MERCEDES-BENZ         B-CLASS               87.000000
Name: Electric Range, dtype: float64
```





Are there any regional trends in EV adoption (e.g., urban vs. rural areas)?

In [33]:

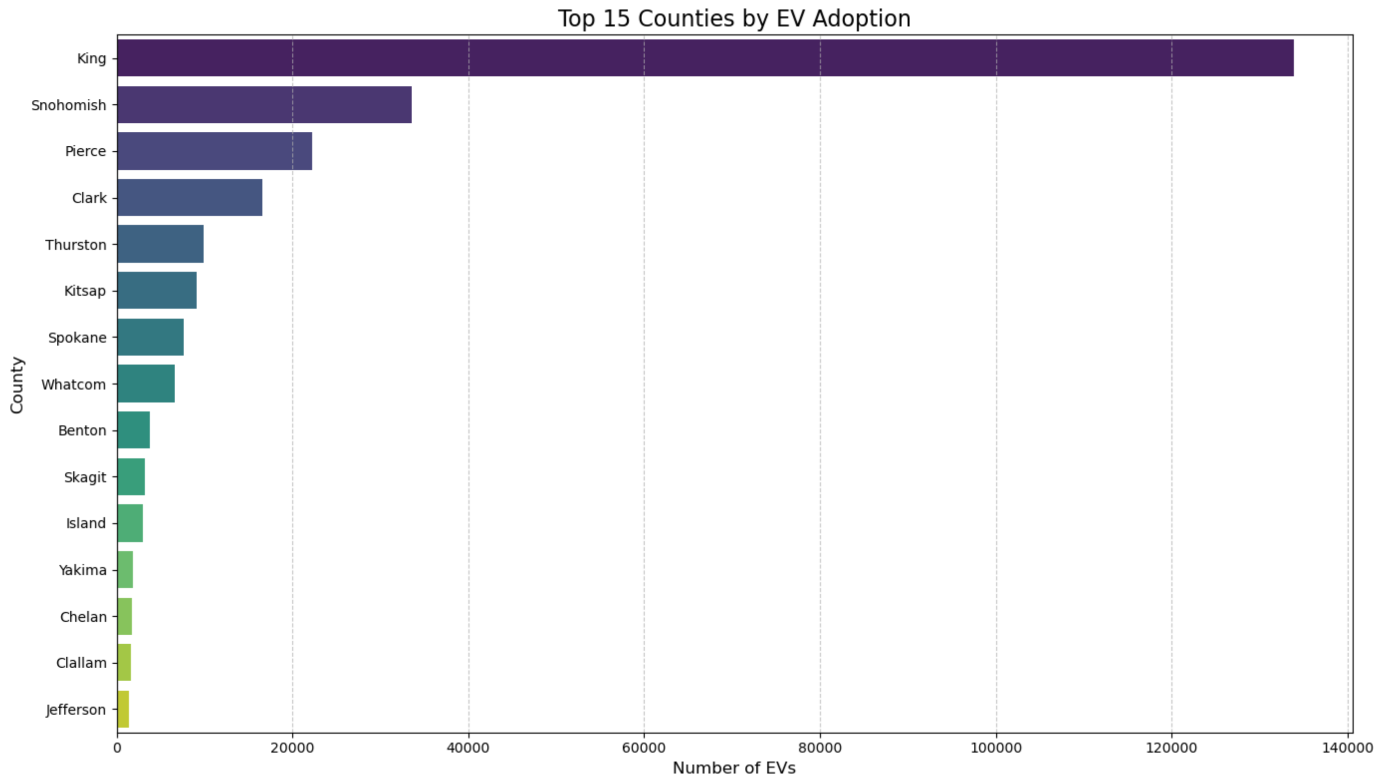
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Step 1: Check if we have location data
print("Available columns for regional analysis:", df1.columns.tolist())

# Assuming we have a 'County' or similar column
# If you have a different location column, replace 'County' with that column name
if 'County' in df1.columns:
    # Step 2: Count EVs by county
    ev_by_county = df1['County'].value_counts().reset_index()
    ev_by_county.columns = ['County', 'Number of EVs']

    # Display the counties with the most EVs
    print("\nTop 10 Counties by EV Count:")
    print(ev_by_county.head(10))

    # Step 3: Visualize EV distribution by county
    plt.figure(figsize=(14, 8))
    sns.barplot(x='Number of EVs', y='County', data=ev_by_county.head(15), palette='viri
    plt.title('Top 15 Counties by EV Adoption', fontsize=16)
    plt.xlabel('Number of EVs', fontsize=12)
    plt.ylabel('County', fontsize=12)
    plt.grid(axis='x', linestyle='--', alpha=0.7)
```

```
    plt.tight_layout()
    plt.show()

    # Step 4: If you have population data, calculate EV density
    # This requires joining with external population data
    # For demonstration, we'll create a mock population dataset
    # In a real scenario, you would import actual population data

    # Mock population data (replace with real data if available)
    # This assumes you have a way to classify counties as urban or rural
    urban_rural_classification = {
        # Example: 'County_Name': {'population': 100000, 'type': 'Urban'}
    }

    # If you have actual urban/rural classification data:
    if urban_rural_classification:
        # Create a DataFrame from the classification dictionary
        county_info = pd.DataFrame.from_dict(
            {k: {'population': v['population'], 'type': v['type']}
             for k, v in urban_rural_classification.items()},
            orient='index'
        ).reset_index()
        county_info.columns = ['County', 'Population', 'Area_Type']

        # Merge with EV data
        county_ev_data = pd.merge(ev_by_county, county_info, on='County', how='inner')

        # Calculate EVs per 1000 residents
        county_ev_data['EVs_per_1000'] = (county_ev_data['Number of EVs'] / county_ev_da

        # Compare urban vs rural adoption
        urban_rural_comparison = county_ev_data.groupby('Area_Type').agg({
            'Number of EVs': 'sum',
            'Population': 'sum',
            'EVs_per_1000': 'mean'
        }).reset_index()

        print("\nUrban vs Rural EV Adoption:")
        print(urban_rural_comparison)

        # Visualize urban vs rural comparison
        plt.figure(figsize=(12, 6))
        sns.barplot(x='Area_Type', y='EVs_per_1000', data=urban_rural_comparison, palett
        plt.title('EV Adoption Rate: Urban vs Rural Areas', fontsize=16)
        plt.xlabel('Area Type', fontsize=12)
        plt.ylabel('EVs per 1,000 Residents', fontsize=12)
        plt.grid(axis='y', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()
else:
    # If we don't have direct county/location data we can check with check for zip codes
    if 'ZIP Code' in df1.columns or 'Zip' in df1.columns or 'Postal Code' in df1.columns
        # Determine which zip code column to use
        zip_col = [col for col in ['ZIP Code', 'Zip', 'Postal Code'] if col in df1.colum

        print(f"\nAnalyzing regional trends using {zip_col}")

        # Count EVs by zip code
        ev_by_zip = df1[zip_col].value_counts().reset_index()
```

```
        ev_by_zip.columns = ['ZIP Code', 'Number of EVs']

        print("\nTop 10 ZIP Codes by EV Count:")
        print(ev_by_zip.head(10))

        # Visualize top zip codes
        plt.figure(figsize=(14, 8))
        sns.barplot(x='Number of EVs', y='ZIP Code', data=ev_by_zip.head(15), palette='v
        plt.title('Top 15 ZIP Codes by EV Adoption', fontsize=16)
        plt.xlabel('Number of EVs', fontsize=12)
        plt.ylabel('ZIP Code', fontsize=12)
        plt.grid(axis='x', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()

        print("\nNote: To properly classify ZIP codes as urban or rural, you would need
    else:
        print("\nNo location data (County, City, or ZIP Code) found in the dataset to an
        print("To analyze urban vs. rural adoption patterns, you need location data that
```

```
Available columns for regional analysis: ['VIN (1-10)', 'County', 'City', 'State', 'Post
al Code', 'Model Year', 'Make', 'Model', 'Electric Vehicle Type', 'Clean Alternative Fue
l Vehicle (CAFV) Eligibility', 'Electric Range', 'Legislative District', 'DOL Vehicle I
D', 'Vehicle Location', 'Electric Utility', '2020 Census Tract']

Top 10 Counties by EV Count:
      County  Number of EVs
0       King         133903
1  Snohomish          33531
2     Pierce          22213
3      Clark          16553
4   Thurston           9852
5     Kitsap           9057
6    Spokane           7593
7    Whatcom           6620
8     Benton           3792
9     Skagit           3166
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_1800\3802896159.py:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Number of EVs', y='County', data=ev_by_county.head(15), palette='viridi
s')
```



DATA VISUALISATION

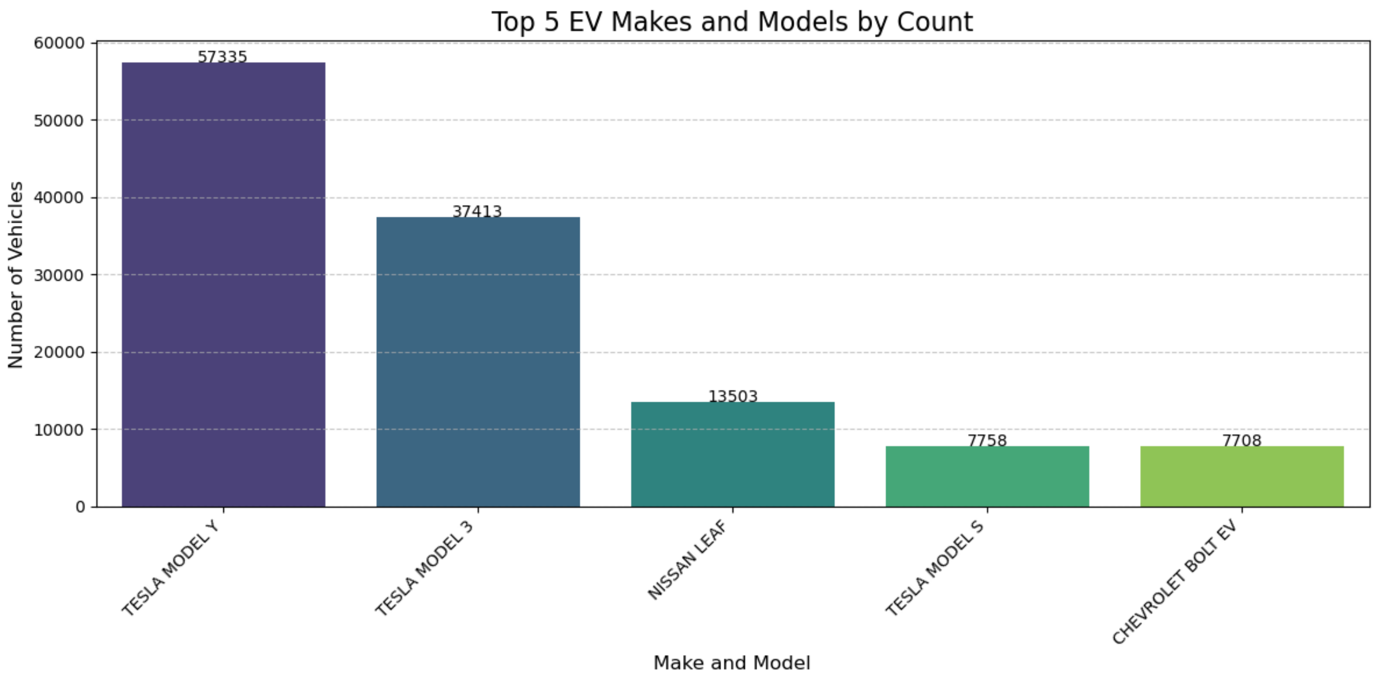
Create a bar chart showing the top 5 EV makes and models by count.

In [34]:

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

# Count EVs by make and model
make_model_counts = df1.groupby(['Make', 'Model']).size().reset_index(name='Count')

# Sort by count in descending order
make_model_counts = make_model_counts.sort_values('Count', ascending=False)

# Get the top 5 make-model combinations
top_5_make_models = make_model_counts.head(5)

# Create a new column combining make and model for display
top_5_make_models['Make_Model'] = top_5_make_models['Make'] + ' ' + top_5_make_models['M

# Create the bar chart
plt.figure(figsize=(12, 6))
bars = sns.barplot(x='Make_Model', y='Count', data=top_5_make_models, palette='viridis')

# Add title and labels
plt.title('Top 5 EV Makes and Models by Count', fontsize=16)
plt.xlabel('Make and Model', fontsize=12)
plt.ylabel('Number of Vehicles', fontsize=12)
plt.xticks(rotation=45, ha='right')

# Add value labels on top of each bar
for i, v in enumerate(top_5_make_models['Count']):
```

```
    plt.text(i, v + 0.5, str(v), ha='center')

# Add grid lines for better readability
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()

# Print the data for reference
print("Top 5 EV Makes and Models by Count:")
print(top_5_make_models[['Make', 'Model', 'Count']])

# Calculate percentage of total
total_evs = len(df1)
top_5_make_models['Percentage'] = (top_5_make_models['Count'] / total_evs * 100).round(2
print("\nPercentage of Total EVs:")
print(top_5_make_models[['Make_Model', 'Count', 'Percentage']])
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_1800\2335216976.py:15: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  top_5_make_models['Make_Model'] = top_5_make_models['Make'] + ' ' + top_5_make_models
['Model']
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_1800\2335216976.py:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  bars = sns.barplot(x='Make_Model', y='Count', data=top_5_make_models, palette='viridi
s')
```




```
Top 5 EV Makes and Models by Count:
          Make    Model  Count
```

```
159      TESLA  MODEL Y  57335
156      TESLA  MODEL 3  37413
136     NISSAN     LEAF  13503
157      TESLA  MODEL S   7758
48   CHEVROLET  BOLT EV   7708

Percentage of Total EVs:
            Make_Model  Count  Percentage
159      TESLA MODEL Y  57335       21.21
156      TESLA MODEL 3  37413       13.84
136        NISSAN LEAF  13503        5.00
157      TESLA MODEL S   7758        2.87
48   CHEVROLET BOLT EV   7708        2.85
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_1800\2335216976.py:46: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  top_5_make_models['Percentage'] = (top_5_make_models['Count'] / total_evs * 100).round
(2)
```

Use a heatmap or choropleth map to visualize EV distribution by county.

In [35]:

```
# pip install geopandas
```

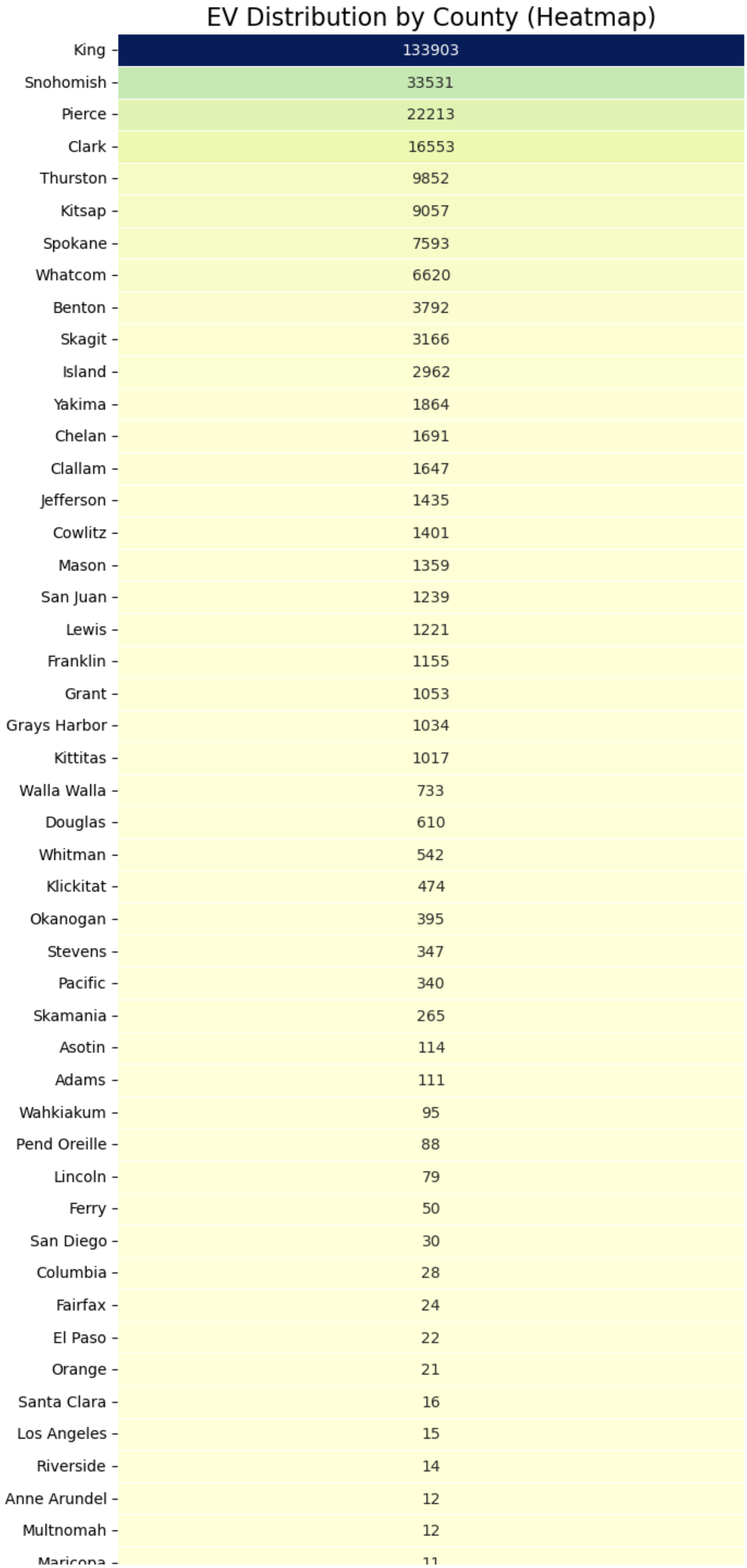
In [36]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import geopandas as gpd
from matplotlib.colors import LinearSegmentedColormap

# Step 1: Count EVs by county
if 'County' in df1.columns:
    # Count EVs by county
    ev_by_county = df1['County'].value_counts().reset_index()
    ev_by_county.columns = ['County', 'Number of EVs']

    print("EV Distribution by County:")
    print(ev_by_county.head(10))

    # Step 2: Create a heatmap-style visualization
    # First, ensure county names are standardized (e.g., remove "County" suffix if prese
    ev_by_county['County'] = ev_by_county['County'].str.replace(' County', '', regex=Fal

    # Create a pivot table for the heatmap
    # We'll use a dummy column since we only have one value per county
    ev_by_county['Dummy'] = 'EV Count'
    heatmap_data = ev_by_county.pivot(index='County', columns='Dummy', values='Number of

    # Sort by EV count for better visualization
    heatmap_data = heatmap_data.sort_values('EV Count', ascending=False)

    # Create a heatmap
```

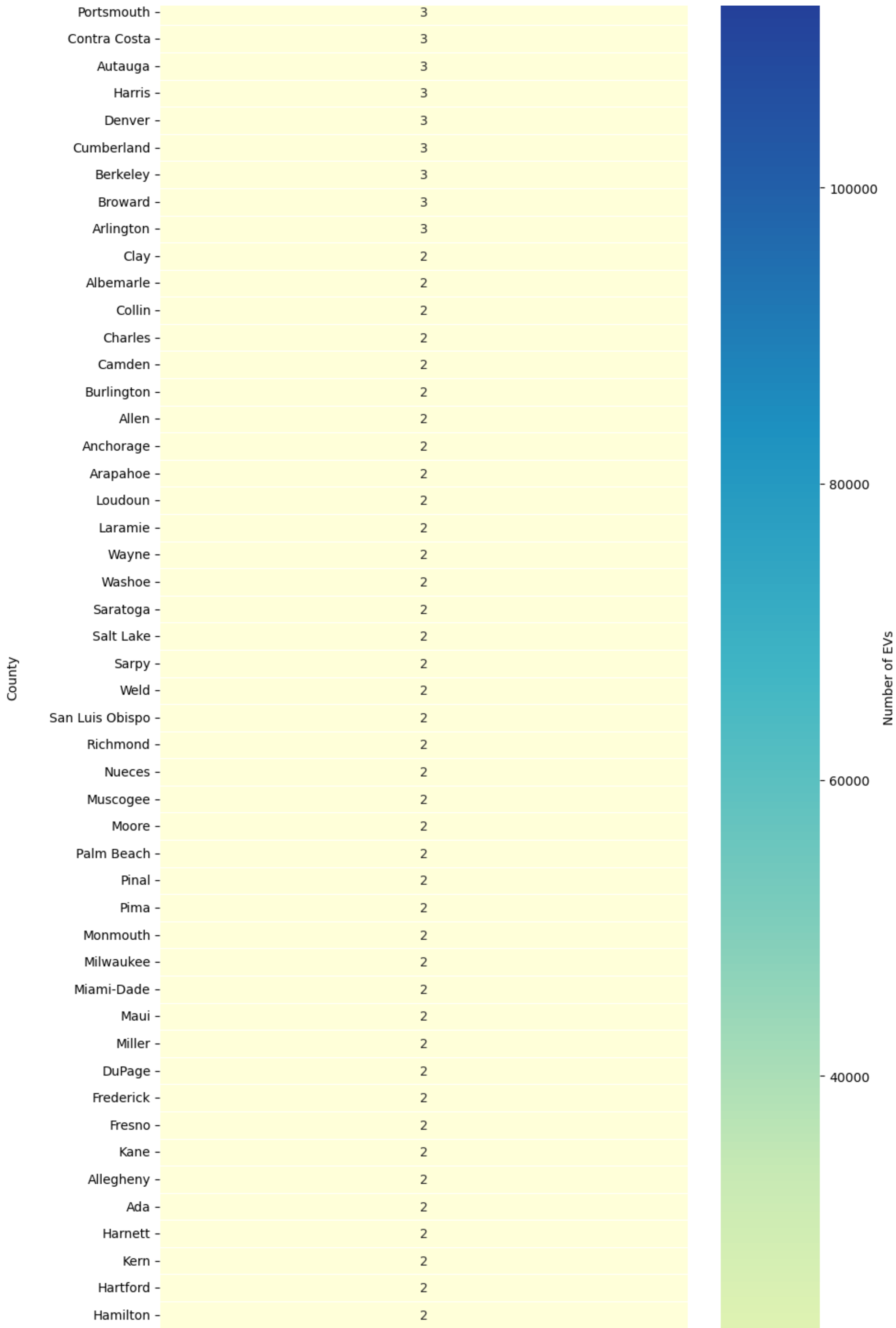
```
    plt.figure(figsize=(10, max(8, len(ev_by_county) * 0.3)))  # Adjust height based on
    sns.heatmap(heatmap_data, annot=True, fmt='d', cmap='YlGnBu',
                linewidths=.5, cbar_kws={'label': 'Number of EVs'})
    plt.title('EV Distribution by County (Heatmap)', fontsize=16)
    plt.tight_layout()
    plt.show()

    # Step 3: Create a choropleth map
    try:
        # Load US counties shapefile
        # You may need to download this file or use a different source
        # This example uses the US Census Bureau's county shapefile
        counties = gpd.read_file('https://raw.githubusercontent.com/plotly/datasets/mast

        # If the above URL doesn't work, you can use:
        # counties = gpd.read_file(gpd.datasets.get_path('us_counties'))

        # Alternatively, for a specific state, you might use:
        # counties = gpd.read_file('path_to_state_county_shapefile.shp')

        # Standardize county names for joining
        counties['NAME'] = counties['NAME'].str.upper()
        ev_by_county['County'] = ev_by_county['County'].str.upper()

        # Merge EV data with geographic data
        # Note: The actual column names in your shapefile may differ
        merged_data = counties.merge(ev_by_county, left_on='NAME', right_on='County', ho

        # Fill NaN values with 0 (counties with no EVs in the dataset)
        merged_data['Number of EVs'] = merged_data['Number of EVs'].fillna(0)

        # Create the choropleth map
        fig, ax = plt.subplots(1, 1, figsize=(15, 10))

        # Create a custom colormap
        cmap = LinearSegmentedColormap.from_list("", ["#f7fbff", "#08519c"])

        # Plot the choropleth
        merged_data.plot(column='Number of EVs',
                         ax=ax,
                         legend=True,
                         cmap=cmap,
                         legend_kwds={'label': "Number of EVs by County",
                                      'orientation': "horizontal"})

        # Add title and remove axis
        ax.set_title('EV Distribution by County', fontsize=16)
        ax.set_axis_off()

        plt.tight_layout()
        plt.show()

    except Exception as e:
        print(f"Could not create choropleth map: {e}")
        print("To create a choropleth map, you need to install geopandas and have access
        print("Alternative approach: Consider using Folium or Plotly for interactive map

        # Alternative: Create a simple bar chart of top counties
        plt.figure(figsize=(14, 8))
```

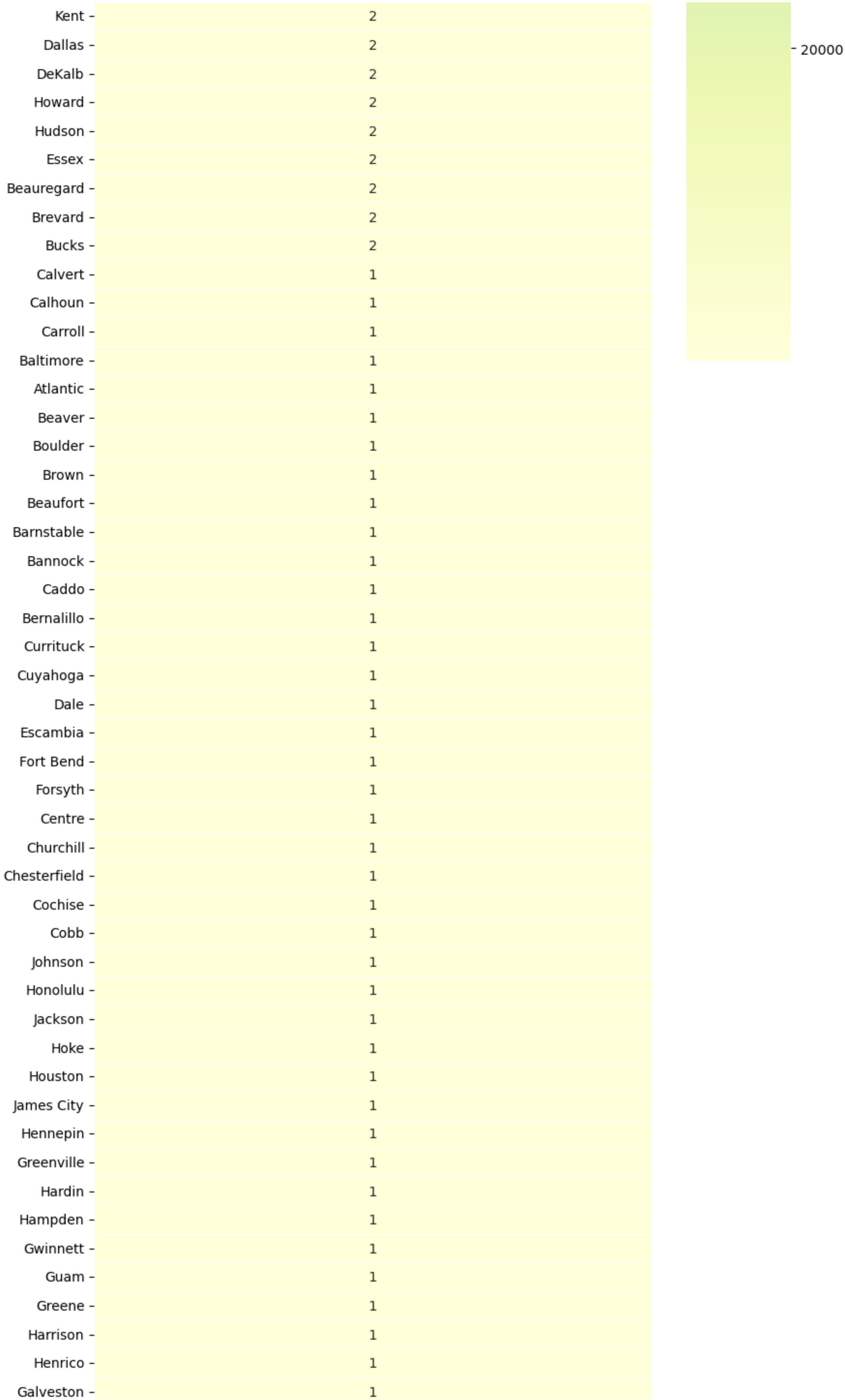
```
        top_counties = ev_by_county.head(15)
        sns.barplot(x='Number of EVs', y='County', data=top_counties, palette='YlGnBu')
        plt.title('Top 15 Counties by EV Count', fontsize=16)
        plt.xlabel('Number of EVs', fontsize=12)
        plt.ylabel('County', fontsize=12)
        plt.grid(axis='x', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()

        # Provide code for interactive map using Plotly
        print("\nHere's code for an interactive choropleth map using Plotly:")
        print("""
import plotly.express as px

# Load county FIPS codes (you'll need to create or obtain this mapping)
# county_fips = pd.read_csv('county_fips.csv')  # Map county names to FIPS codes
# ev_with_fips = pd.merge(ev_by_county, county_fips, on='County', how='left')

# For this example, we'll use a sample dataset with FIPS codes
fig = px.choropleth(ev_by_county,
                    geojson=counties,
                    locations='FIPS',  # FIPS code column
                    color='Number of EVs',
                    color_continuous_scale="Viridis",
                    scope="usa",
                    labels={'Number of EVs': 'EV Count'}
                   )
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
        """)

else:
    print("No 'County' column found in the dataset.")
    print("Available columns:", df1.columns.tolist())
    print("\nTo create a county-level visualization, you need county data in your datase
    print("If you have ZIP codes or other location data, you could aggregate to county l
```

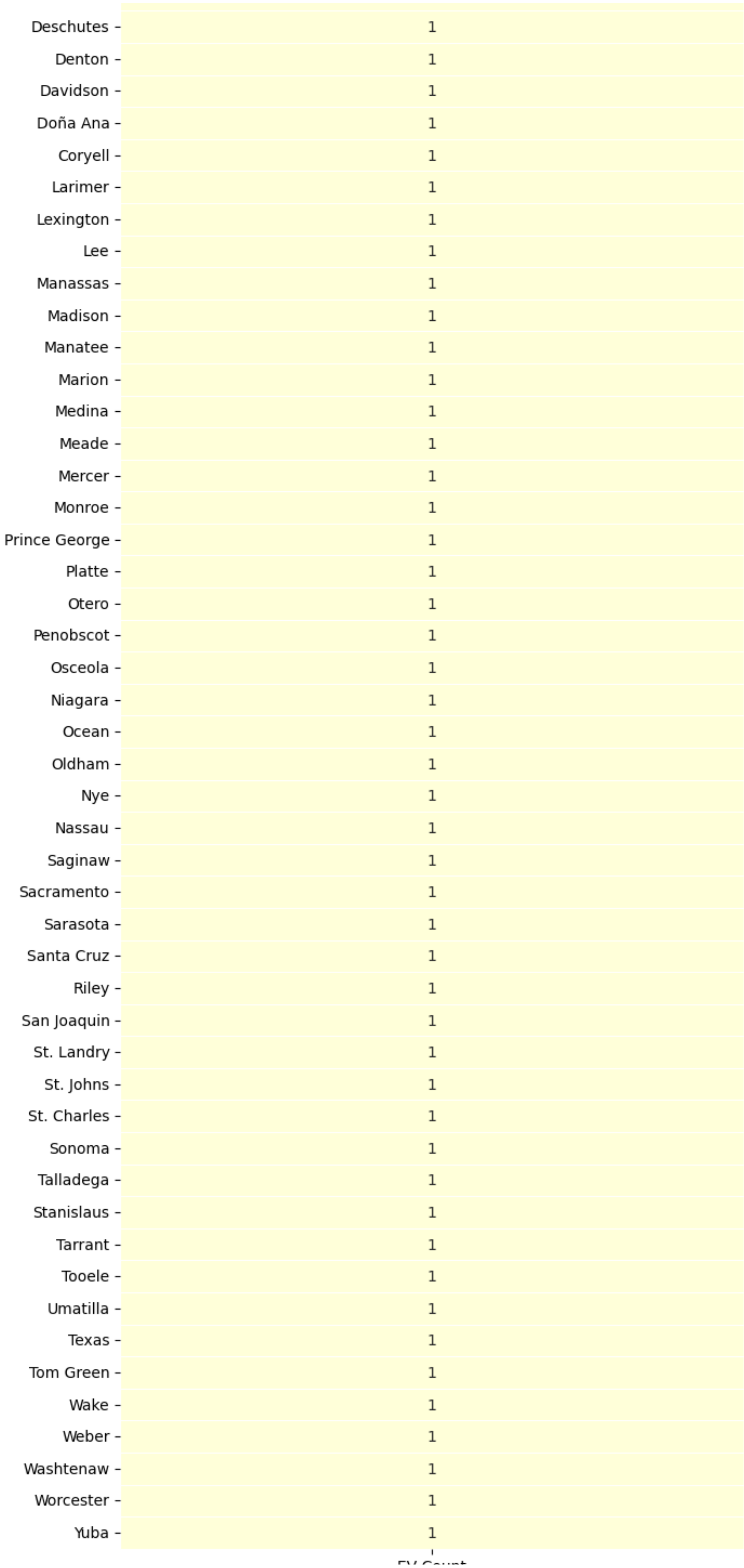
```
EV Distribution by County:
      County  Number of EVs
0       King         133903
1  Snohomish          33531
2     Pierce          22213
3      Clark          16553
4   Thurston           9852
5     Kitsap           9057
6    Spokane           7593
7    Whatcom           6620
8     Benton           3792
9     Skagit           3166
```


EV Distribution by County (Heatmap)
King 133903
Snohomish 33531
Pierce 22213
Clark 16553
Thurston 9852
Kitsap 9057
Spokane 7593
Whatcom 6620
Benton 3792
Skagit 3166
Island 2962
Yakima 1864
Chelan 1691
Clallam 1647
Jefferson 1435
Cowlitz 1401
Mason 1359
San Juan 1239
Lewis 1221
Franklin 1155
Grant 1053
Grays Harbor 1034
Kittitas 1017
Walla Walla 733
Douglas 610
Whitman 542
Klickitat 474
Okanogan 395
Stevens 347
Pacific 340
Skamania 265
Asotin 114
Adams 111
Wahkiakum 95
Pend Oreille 88
Lincoln 79
Ferry 50
San Diego 30
Columbia 28
Fairfax 24
El Paso 22
Orange 21
Santa Clara 16
Los Angeles 15
Riverside 14
Anne Arundel 12
Multnomah 12


Alameda 11
Duval 8
Bexar 8
Lake 8
Montgomery 8
Virginia Beach 7
Kings 7
Alexandria 7
Macomb 6
Garfield 6
Chesapeake 6
Leavenworth 6
Monterey 6
Prince George's 6
Hillsborough 5
District of Columbia 5
Clackamas 5
Harford 5
San Bernardino 5
San Mateo 5
Rockingham 5
New London 5
Middlesex 5
Prince William 5
York 4
Williamson 4
Travis 4
Norfolk 4
Okaloosa 4
Ventura 4
Stafford 4
Cook 4
New York 4
Kootenai 4
Bell 4
St. Louis 3
Washington 3
Newport 3
Polk 3
Placer 3
Marin 3
Pulaski 3
San Francisco 3
Shelby 3
Suffolk 3
St. Mary's 3
Solano 3
St. Clair 3
120000


Portsmouth 3
Contra Costa 3
Autauga 3
Harris 3
Denver 3
Cumberland 3
Berkeley 3
Broward 3
Arlington 3
Clay 2
Albemarle 2
Collin 2
Charles 2
Camden 2
Burlington 2
Allen 2
Anchorage 2
Arapahoe 2
Loudoun 2
Laramie 2
Wayne 2
Washoe 2
Saratoga 2
Salt Lake 2
Sarpy 2
Weld 2
San Luis Obispo 2
Richmond 2
Nueces 2
Muscogee 2
Moore 2
Palm Beach 2
Pinal 2
Pima 2
Monmouth 2
Milwaukee 2
Miami-Dade 2
Maui 2
Miller 2
DuPage 2
Frederick 2
Fresno 2
Kane 2
Allegheny 2
Ada 2
Harnett 2
Kern 2
Hartford 2
Hamilton 2
County
100000
80000
60000
40000
Number of EVs

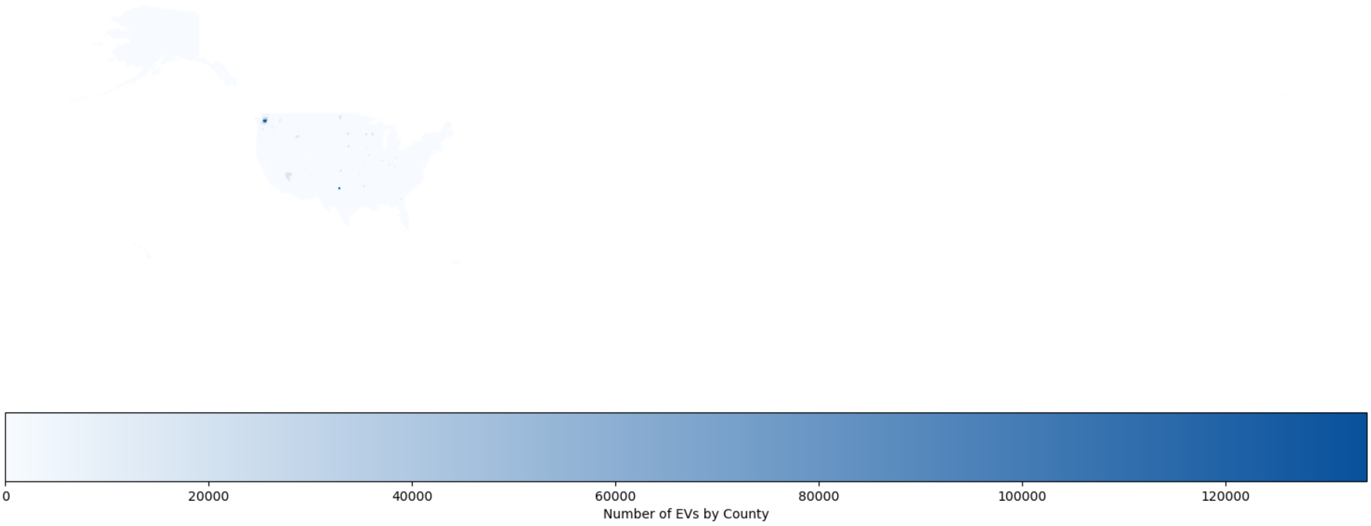

Kent 2
Dallas 2
DeKalb 2
Howard 2
Hudson 2
Essex 2
Beauregard 2
Brevard 2
Bucks 2
Calvert 1
Calhoun 1
Carroll 1
Baltimore 1
Atlantic 1
Beaver 1
Boulder 1
Brown 1
Beaufort 1
Barnstable 1
Bannock 1
Caddo 1
Bernalillo 1
Currituck 1
Cuyahoga 1
Dale 1
Escambia 1
Fort Bend 1
Forsyth 1
Centre 1
Churchill 1
Chesterfield 1
Cochise 1
Cobb 1
Johnson 1
Honolulu 1
Jackson 1
Hoke 1
Houston 1
James City 1
Hennepin 1
Greenville 1
Hardin 1
Hampden 1
Gwinnett 1
Guam 1
Greene 1
Harrison 1
Henrico 1
Galveston 1
20000



Deschutes 1
Denton 1
Davidson 1
Doña Ana 1
Coryell 1
Larimer 1
Lexington 1
Lee 1
Manassas 1
Madison 1
Manatee 1
Marion 1
Medina 1
Meade 1
Mercer 1
Monroe 1
Prince George 1
Platte 1
Otero 1
Penobscot 1
Osceola 1
Niagara 1
Ocean 1
Oldham 1
Nye 1
Nassau 1
Saginaw 1
Sacramento 1
Sarasota 1
Santa Cruz 1
Riley 1
San Joaquin 1
St. Landry 1
St. Johns 1
St. Charles 1
Sonoma 1
Talladega 1
Stanislaus 1
Tarrant 1
Tooele 1
Umatilla 1
Texas 1
Tom Green 1
Wake 1
Weber 1
Washtenaw 1
Worcester 1
Yuba 1

EV Count
Dummy

EV Distribution by County

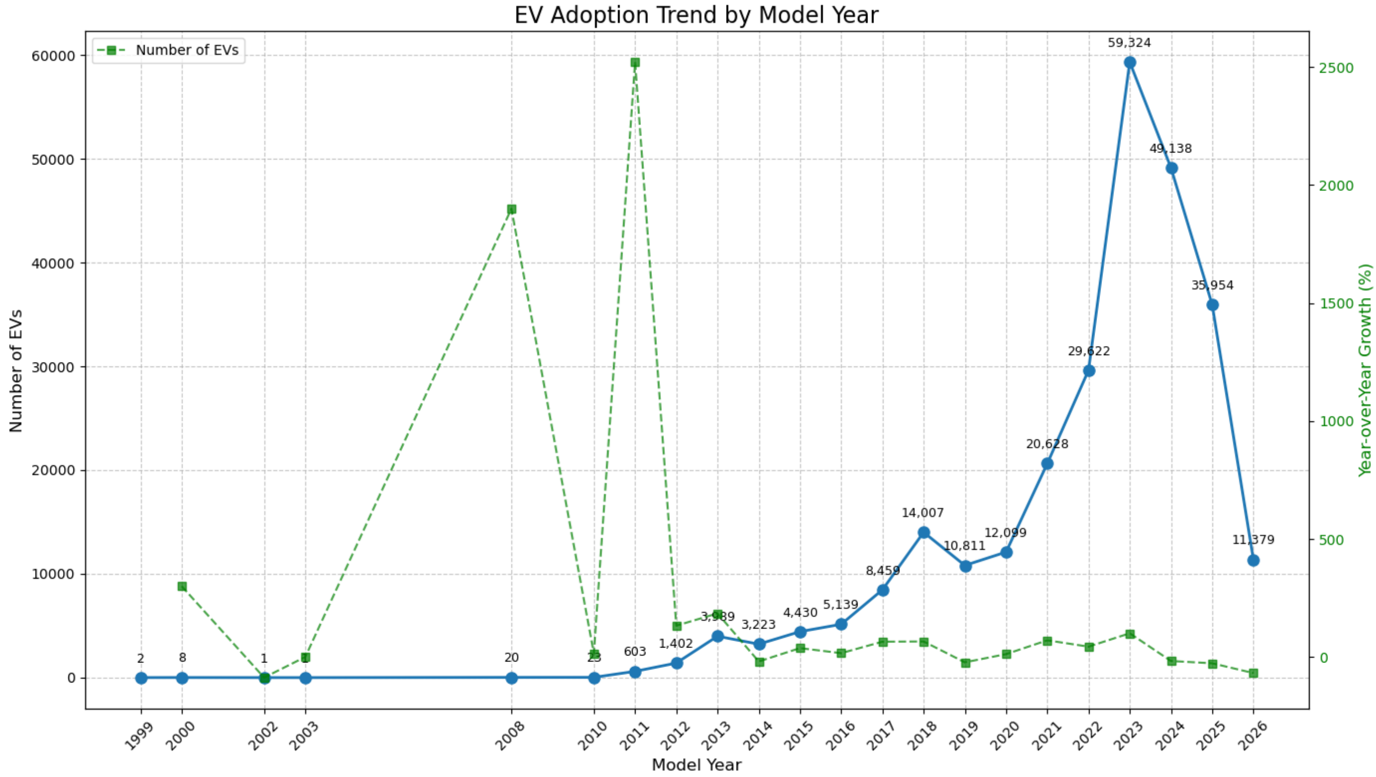


Create a line graph showing the trend of EV adoption by model year.

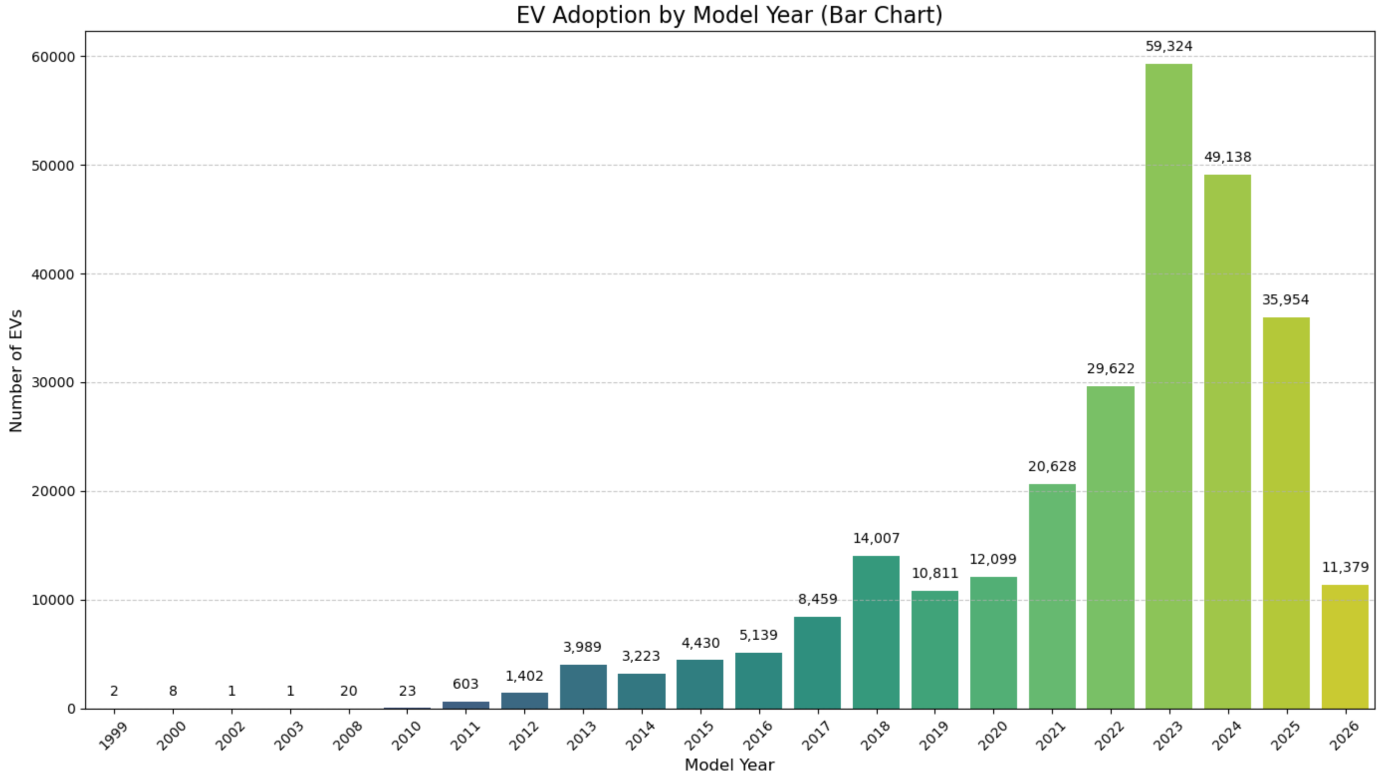
In [37]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Count EVs by model year
ev_by_year = df1['Model Year'].value_counts().sort_index()

# Convert to DataFrame for easier manipulation
ev_year_df = ev_by_year.reset_index()
ev_year_df.columns = ['Model Year', 'Number of EVs']

# Create the line graph
plt.figure(figsize=(14, 8))

# Plot the line with markers
plt.plot(ev_year_df['Model Year'], ev_year_df['Number of EVs'],
         marker='o', linestyle='-', linewidth=2, markersize=8, color='#1f77b4')

# Add data points
for x, y in zip(ev_year_df['Model Year'], ev_year_df['Number of EVs']):
    plt.text(x, y + max(ev_year_df['Number of EVs'])*0.02, f'{y:,}',
             ha='center', va='bottom', fontsize=9)

# Add title and labels
plt.title('EV Adoption Trend by Model Year', fontsize=16)
plt.xlabel('Model Year', fontsize=12)
plt.ylabel('Number of EVs', fontsize=12)

# Add grid for better readability
plt.grid(True, linestyle='--', alpha=0.7)

# Set x-axis ticks to show all years
plt.xticks(ev_year_df['Model Year'], rotation=45)

# Calculate year-over-year growth rates
```

```
ev_year_df['YoY Growth'] = ev_year_df['Number of EVs'].pct_change() * 100

# Add a second y-axis for growth rate
ax2 = plt.gca().twinx()
ax2.plot(ev_year_df['Model Year'][1:], ev_year_df['YoY Growth'][1:],
         marker='s', linestyle='--', color='green', alpha=0.7)
ax2.set_ylabel('Year-over-Year Growth (%)', color='green', fontsize=12)
ax2.tick_params(axis='y', labelcolor='green')

# Add legend
plt.legend(['Number of EVs', 'YoY Growth (%)'], loc='upper left')

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()

# Print the data for reference
print("EV Adoption by Model Year:")
print(ev_year_df)

# Calculate CAGR (Compound Annual Growth Rate)
if len(ev_year_df) > 1:
    first_year = ev_year_df['Model Year'].min()
    last_year = ev_year_df['Model Year'].max()
    first_year_count = ev_year_df[ev_year_df['Model Year'] == first_year]['Number of EVs
    last_year_count = ev_year_df[ev_year_df['Model Year'] == last_year]['Number of EVs']

    # Only calculate if first year count is greater than 0
    if first_year_count > 0:
        years = last_year - first_year
        cagr = (((last_year_count / first_year_count) ** (1 / years)) - 1) * 100
        print(f"\nCompound Annual Growth Rate (CAGR) from {first_year} to {last_year}: {

# Create a bar chart version for comparison
plt.figure(figsize=(14, 8))
sns.barplot(x='Model Year', y='Number of EVs', data=ev_year_df, palette='viridis')

# Add value labels on top of each bar
for i, v in enumerate(ev_year_df['Number of EVs']):
    plt.text(i, v + max(ev_year_df['Number of EVs'])*0.02, f'{v:,}', ha='center')

plt.title('EV Adoption by Model Year (Bar Chart)', fontsize=16)
plt.xlabel('Model Year', fontsize=12)
plt.ylabel('Number of EVs', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
EV Adoption by Model Year:
    Model Year  Number of EVs   YoY Growth
0         1999              2          NaN
1         2000              8   300.000000
2         2002              1   -87.500000
3         2003              1     0.000000
4         2008             20  1900.000000
5         2010             23    15.000000
6         2011            603  2521.739130
7         2012           1402   132.504146
8         2013           3989   184.522111
9         2014           3223   -19.202808
10        2015           4430    37.449581
11        2016           5139    16.004515
12        2017           8459    64.604009
13        2018          14007    65.586949
14        2019          10811   -22.817163
15        2020          12099    11.913792
16        2021          20628    70.493429
17        2022          29622    43.600931
18        2023          59324   100.270070
19        2024          49138   -17.170117
20        2025          35954   -26.830559
21        2026          11379   -68.351227

Compound Annual Growth Rate (CAGR) from 1999 to 2026: 37.75%
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_1800\1767253685.py:74: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Model Year', y='Number of EVs', data=ev_year_df, palette='viridis')
```



Plot a pie chart showing the proportion of CAFV-eligible vs. non-eligible EVs.

In [38]:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Check if the CAFV eligibility column exists
# Common column names for this information
possible_columns = [
    'Clean Alternative Fuel Vehicle Eligible',
       'Eligibility unknown as battery range has not been researched']

# Find the correct column name
cafv_column = None
for col in possible_columns:
    if col in df1.columns:
        cafv_column = col
        break

if cafv_column:
    # Check the unique values in the column
    print(f"Unique values in {cafv_column}:")
    print(df1[cafv_column].value_counts())

    # Count the number of eligible and non-eligible vehicles
    cafv_counts = df1[cafv_column].value_counts()

    # Determine which values indicate eligibility
    # This depends on how the data is coded in your dataset
    # Common values might be 'Eligible', 'Yes', 'Clean Alternative Fuel Vehicle', etc.

    # For this example, we'll assume 'Eligible' indicates eligibility
    # Adjust this based on your actual data values
```

```python
    eligible_values = ['Clean Alternative Fuel Vehicle Eligible',
       'Eligibility unknown as battery range has not been researched',]

    # Create a simplified DataFrame for the pie chart
    # Group all eligible values together and all non-eligible values together
    eligibility_data = pd.DataFrame({
        'Status': ['Eligible', 'Not Eligible'],
        'Count': [
            df1[cafv_column].isin(eligible_values).sum(),
            (~df1[cafv_column].isin(eligible_values)).sum()
        ]
    })

    # Calculate percentages
    eligibility_data['Percentage'] = (eligibility_data['Count'] / eligibility_data['Coun

    # Create labels with percentages
    labels = [f"{status} ({count:,}, {pct}%)"
              for status, count, pct in zip(eligibility_data['Status'],
                                            eligibility_data['Count'],
                                            eligibility_data['Percentage'])]

    # Create the pie chart
    plt.figure(figsize=(10, 8))

    # Plot the pie chart with a slight explosion for the eligible segment
    explode = (0.1, 0)  # explode the 1st slice (Eligible)

    plt.pie(eligibility_data['Count'],
            explode=explode,
            labels=labels,
            autopct='',  # We're using custom labels with percentages
            startangle=90,
            shadow=True,
            colors=['#66b3ff', '#ff9999'],
            wedgeprops={'edgecolor': 'white', 'linewidth': 1.5})

    # Equal aspect ratio ensures that pie is drawn as a circle
    plt.axis('equal')

    # Add title
    plt.title('Proportion of CAFV-Eligible vs. Non-Eligible EVs', fontsize=16)

    # Add a legend
    plt.legend(eligibility_data['Status'], loc="best")

    # Show the plot
    plt.tight_layout()
    plt.show()

    # Print the summary data
    print("\nCAFV Eligibility Summary:")
    print(eligibility_data)

    # Additional analysis: Check if eligibility varies by make
    print("\nCAFV Eligibility by Make:")
    make_eligibility = df1.groupby('Make')[cafv_column].apply(
        lambda x: (x.isin(eligible_values).sum() / len(x) * 100).round(1)
    ).sort_values(ascending=False)
```

```
    print(make_eligibility)

    # Create a bar chart showing eligibility percentage by make
    plt.figure(figsize=(14, 8))
    make_eligibility.plot(kind='bar', color='skyblue')
    plt.title('Percentage of CAFV-Eligible EVs by Make', fontsize=16)
    plt.xlabel('Make', fontsize=12)
    plt.ylabel('Percentage Eligible (%)', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(rotation=45, ha='right')

    # Add value labels on top of each bar
    for i, v in enumerate(make_eligibility):
        plt.text(i, v + 1, f"{v}%", ha='center')

    plt.tight_layout()
    plt.show()

else:
    print("No CAFV eligibility column found in the dataset.")
    print("Available columns:", df1.columns.tolist())
```

```
No CAFV eligibility column found in the dataset.
Available columns: ['VIN (1-10)', 'County', 'City', 'State', 'Postal Code', 'Model Yea
r', 'Make', 'Model', 'Electric Vehicle Type', 'Clean Alternative Fuel Vehicle (CAFV) Eli
gibility', 'Electric Range', 'Legislative District', 'DOL Vehicle ID', 'Vehicle Locatio
n', 'Electric Utility', '2020 Census Tract']
```

In [39]:

```
!pip install folium
```

```
Requirement already satisfied: folium in c:\users\lenovo\anaconda3\lib\site-packages (0.
20.0)
Requirement already satisfied: branca>=0.6.0 in c:\users\lenovo\anaconda3\lib\site-packa
ges (from folium) (0.8.2)
Requirement already satisfied: jinja2>=2.9 in c:\users\lenovo\anaconda3\lib\site-package
s (from folium) (3.1.6)
Requirement already satisfied: numpy in c:\users\lenovo\anaconda3\lib\site-packages (fro
m folium) (2.1.3)
Requirement already satisfied: requests in c:\users\lenovo\anaconda3\lib\site-packages
(from folium) (2.32.3)
Requirement already satisfied: xyzservices in c:\users\lenovo\anaconda3\lib\site-package
s (from folium) (2022.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\lenovo\anaconda3\lib\site-pac
kages (from jinja2>=2.9->folium) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\lenovo\anaconda3\lib
\site-packages (from requests->folium) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\lenovo\anaconda3\lib\site-packag
es (from requests->folium) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\lenovo\anaconda3\lib\site-
packages (from requests->folium) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\lenovo\anaconda3\lib\site-
packages (from requests->folium) (2025.8.3)
```

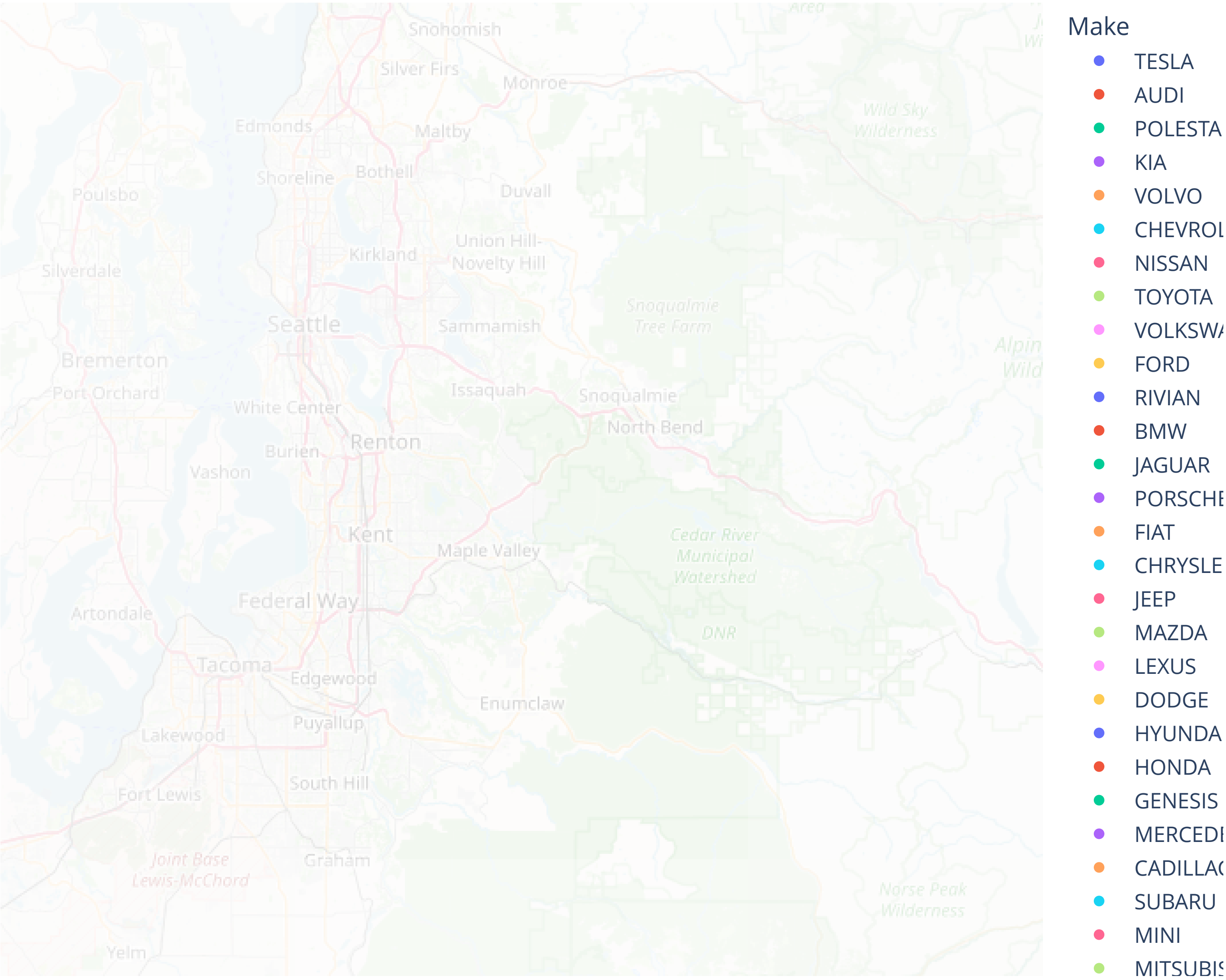
Use a geospatial map to display EV registrations based on vehicle location.

In [ ]:

```python
# Import the necessary libraries first
import pandas as pd
import plotly.express as px  # This import was missing
import folium
from folium.plugins import MarkerCluster  # This import might also be needed

# Rest of your code remains the same
# We're using the right ones
print(df1.columns.tolist())

# We need to extract latitude and longitude from 'Vehicle Location' column
# Assuming 'Vehicle Location' contains coordinates in format "POINT (longitude latitude)
# Extract coordinates from 'Vehicle Location'
coords = df1['Vehicle Location'].str.extract(r'POINT \(([-\d.]+) ([-\d.]+)\)')
# Rename the extracted columns
coords.columns = ['Longitude', 'Latitude']
# Convert to numeric and assign to df1
df1['Longitude'] = pd.to_numeric(coords['Longitude'])
df1['Latitude'] = pd.to_numeric(coords['Latitude'])

# Method 1: Using Plotly Express (interactive and works well in Jupyter)
fig = px.scatter_mapbox(df1,
                        lat='Latitude',
                        lon='Longitude',
                        hover_name='City',
                        hover_data=['Make', 'Model'],
                        color='Make',
                        zoom=8,
                        height=600,
                        width=800,
                        title='EV Registrations by Location')

fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":50,"l":0,"b":0})
fig.show()

# Method 2: Using Folium (more customizable)
# Filter out rows with NaN values in Latitude or Longitude
# This is the key fix to address the error
df_clean = df1.dropna(subset=['Latitude', 'Longitude'])

# Create a map centered at the mean of your data points
m = folium.Map(location=[df_clean['Latitude'].mean(), df_clean['Longitude'].mean()],
               zoom_start=10)

# Add a marker cluster to make the map more readable with many points
marker_cluster = MarkerCluster().add_to(m)

# Add markers for each EV registration
for idx, row in df_clean.iterrows():  # Using the filtered dataframe
    folium.Marker(
        location=[row['Latitude'], row['Longitude']],
        popup=f"Make: {row['Make']}<br>Model: {row['Model']}<br>City: {row['City']}",
        tooltip=row['City']
    ).add_to(marker_cluster)

# Display the map
m
```

```
['VIN (1-10)', 'County', 'City', 'State', 'Postal Code', 'Model Year', 'Make', 'Model',
'Electric Vehicle Type', 'Clean Alternative Fuel Vehicle (CAFV) Eligibility', 'Electric
Range', 'Legislative District', 'DOL Vehicle ID', 'Vehicle Location', 'Electric Utilit
y', '2020 Census Tract']
```

EV Registrations by Location



In [ ]:

```python
# pip install scikit-learn
```

What independent variables (features) can be used to predict Electric Range? (e.g., Model Year, Base MSRP, Make)

In [ ]:

Linear Regression

[27]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

[6]:
```python
df1 = pd.read_csv('D:\manish\Python\GROW AI\python\Electric_Vehicle_Population_Data.csv')
df1
```

[6]:

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	Electric Range	Legislative District	DOL Vehicle ID	Vehicle Location	Electric Utility	2020 Census Tract
0	5YJYGDEE8L	Thurston	Tumwater	WA	98501.0	2020	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	291.0	35.0	124633715	POINT (-122.89165 47.03954)	PUGET SOUND ENERGY INC	5.306701e+10
1	5YJXCAE2XJ	Snohomish	Bothell	WA	98021.0	2018	TESLA	MODEL X	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	238.0	1.0	474826075	POINT (-122.18384 47.8031)	PUGET SOUND ENERGY INC	5.306105e+10
2	5YJ3E1EBXK	King	Kent	WA	98031.0	2019	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	220.0	47.0	280307233	POINT (-122.17743 47.41185)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	5.303303e+10
3	7SAYGDEE4T	King	Issaquah	WA	98027.0	2026	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	0.0	41.0	280786565	POINT (-122.03439 47.5301)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	5.303302e+10
4	WAUUPBFF9G	King	Seattle	WA	98103.0	2016	AUDI	A3	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	16.0	43.0	198988891	POINT (-122.35436 47.67596)	CITY OF SEATTLE - (WA)|CITY OF TACOMA - (WA)	5.303300e+10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
270257	1C4RJXN60R	Pierce	Joint Base Lewis Mcchord	WA	98433.0	2024	JEEP	WRANGLER	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	21.0	28.0	266021122	POINT (-122.60761 47.08798)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)	5.305307e+10
270258	1C4JJXR66N	Mason	Hoodsport	WA	98548.0	2022	JEEP	WRANGLER	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	22.0	35.0	282482938	POINT (-123.14135 47.40639)	BONNEVILLE POWER ADMINISTRATION||CITY OF TACOM...	5.304596e+10
270259	7SAYGDEEXP	Pierce	Tacoma	WA	98406.0	2023	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	0.0	27.0	228485085	POINT (-122.52082 47.26887)	BONNEVILLE POWER ADMINISTRATION||CITY OF TACOM...	5.305306e+10
270260	5YJYGDEE2M	Snohomish	Bothell	WA	98021.0	2021	TESLA	MODEL Y	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	0.0	1.0	282699217	POINT (-122.18384 47.8031)	PUGET SOUND ENERGY INC	5.306105e+10
270261	JN1BF0BA5P	Chelan	Wenatchee	WA	98801.0	2023	NISSAN	ARIYA HATCHBACK	Battery Electric Vehicle (BEV)	Eligibility unknown as battery range has not b...	0.0	12.0	261475224	POINT (-120.30521 47.41493)	PUD NO 1 OF CHELAN COUNTY	5.300796e+10

270262 rows × 16 columns

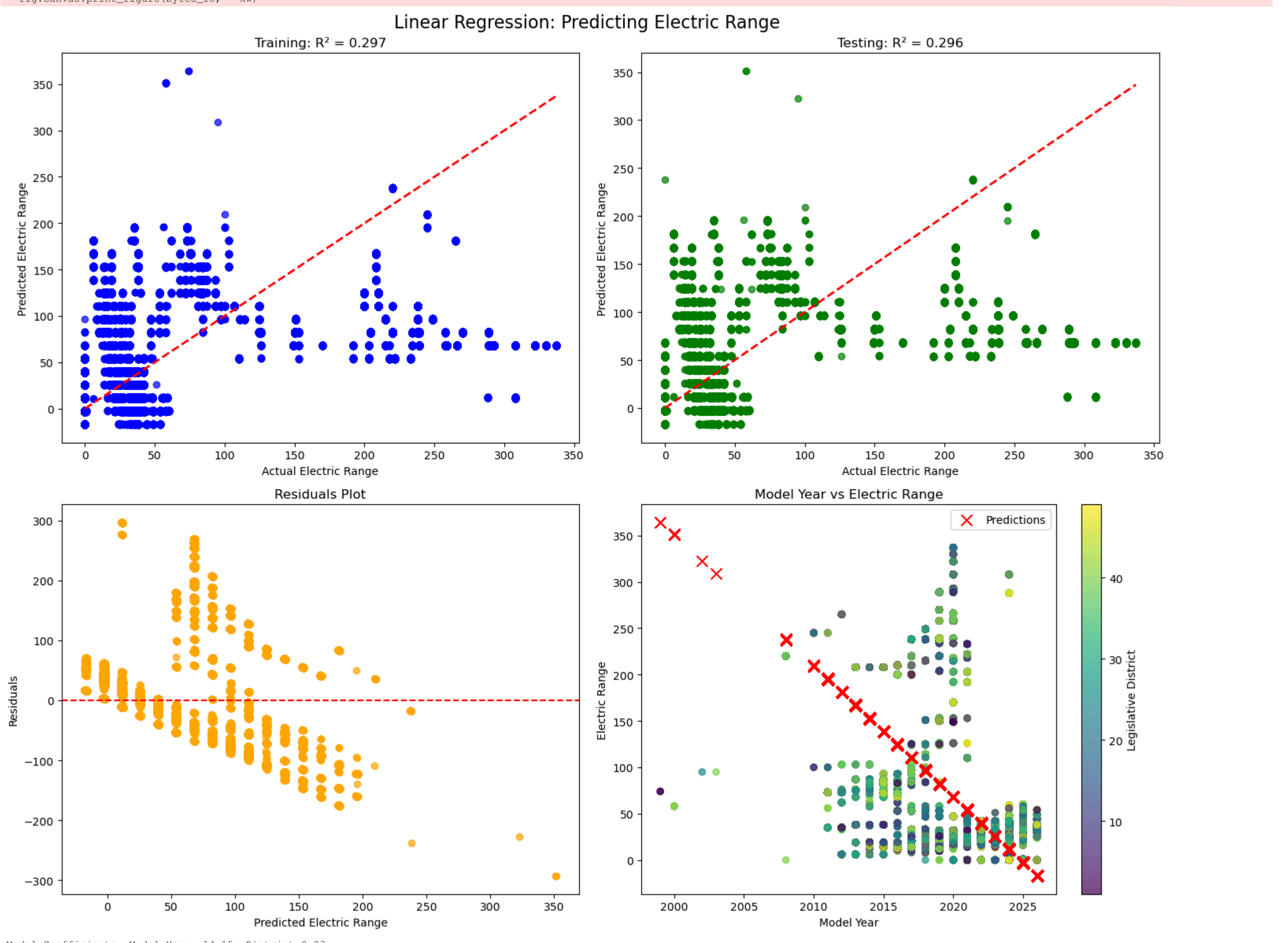
How can we use Linear Regression to predict the Electric Range of a vehicle?

[8]:
```python
d1 = df[['Model Year', 'Electric Range', 'Legislative District']].dropna()
X = d1[['Model Year', 'Legislative District']]
y = d1['Electric Range']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("R²:", r2_score(y_test, y_pred))
print("Predictions:", y_pred)
```

```
Coefficients: [-14.14570503   0.02866648]
Intercept: 28641.659603274988
R²: 0.295898700156663
Predictions: [ 12.04262164  40.36269818  25.24233268 ... -16.22012193  -2.21774932
  39.30203825]
```

[10]:
```python
d1 = df[['Model Year', 'Electric Range', 'Legislative District']].dropna()
X = d1[['Model Year', 'Legislative District']]
y = d1['Electric Range']

# Train model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = LinearRegression().fit(X_train, y_train)
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Create visualizations
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
fig.suptitle('Linear Regression: Predicting Electric Range', fontsize=16)

# 1. Actual vs Predicted (Training)
axes[0,0].scatter(y_train, y_pred_train, alpha=0.7, color='blue')
axes[0,0].plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'r--', lw=2)
axes[0,0].set_xlabel('Actual Electric Range')
axes[0,0].set_ylabel('Predicted Electric Range')
axes[0,0].set_title(f'Training: R² = {r2_score(y_train, y_pred_train):.3f}')

# 2. Actual vs Predicted (Testing)
axes[0,1].scatter(y_test, y_pred_test, alpha=0.7, color='green')
axes[0,1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
axes[0,1].set_xlabel('Actual Electric Range')
axes[0,1].set_ylabel('Predicted Electric Range')
axes[0,1].set_title(f'Testing: R² = {r2_score(y_test, y_pred_test):.3f}')

# 3. Residuals Plot
residuals = y_test - y_pred_test
axes[1,0].scatter(y_pred_test, residuals, alpha=0.7, color='orange')
axes[1,0].axhline(y=0, color='r', linestyle='--')
axes[1,0].set_xlabel('Predicted Electric Range')
axes[1,0].set_ylabel('Residuals')
axes[1,0].set_title('Residuals Plot')

# 4. Model Year vs Electric Range with regression line
scatter = axes[1,1].scatter(d1['Model Year'], d1['Electric Range'],
                            c=d1['Legislative District'], cmap='viridis', alpha=0.7)
model_year_line = axes[1,1].scatter(X['Model Year'], model.predict(X),
                                    color='red', s=100, marker='x', label='Predictions')
axes[1,1].set_xlabel('Model Year')
axes[1,1].set_ylabel('Electric Range')
axes[1,1].set_title('Model Year vs Electric Range')
plt.colorbar(scatter, ax=axes[1,1], label='Legislative District')
axes[1,1].legend()

plt.tight_layout()
plt.show()

# Print model statistics
print(f"Model Coefficients: Model Year={model.coef_[0]:.2f}, District={model.coef_[1]:.2f}")
print(f"Intercept: {model.intercept_:.2f}")
print(f"Test R²: {r2_score(y_test, y_pred_test):.3f}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred_test)):.2f}")
```

```
C:\Users\LENOVO\anaconda3\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Creating legend with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```




```
Model Coefficients: Model Year=-14.15, District=0.03
Intercept: 28641.66
Test R²: 0.296
RMSE: 66.52
```

What independent variables (features) can be used to predict Electric Range? (e.g., Model Year, Base MSRP, Make)

[11]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder

# Potential numeric features for Electric Range prediction
numeric_features = ['Model Year', 'Legislative District', 'Postal Code']
categorical_features = ['Make', 'Model', 'Electric Vehicle Type', 'Clean Alternative Fuel Vehicle (CAFV) Eligibility']

# Create feature matrix (encode categoricals, handle missing values)
df_numeric = df[numeric_features + ['Electric Range']].dropna()
X_numeric = df_numeric[numeric_features]

# Encode categorical features (one-hot for small dataset)
df_cat = pd.get_dummies(df[categorical_features + ['Electric Range']].dropna(), drop_first=True)
X_cat = df_cat.drop('Electric Range', axis=1)

print("Available Numeric Features:", numeric_features)
print("Available Categorical Features:", categorical_features)
print("\nDataset shape after cleaning:", df_numeric.shape)
```

```
Available Numeric Features: ['Model Year', 'Legislative District', 'Postal Code']
Available Categorical Features: ['Make', 'Model', 'Electric Vehicle Type', 'Clean Alternative Fuel Vehicle (CAFV) Eligibility']

Dataset shape after cleaning: (269608, 4)
```

How do we handle categorical variables like Make and Model in regression analysis?

[14]:
```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

df_clean = df1[['Make', 'Model', 'Electric Range', 'Model Year']].dropna()

# METHOD 1: ONE-HOT ENCODING (Recommended for nominal categoricals)
X_cat = df_clean[['Make', 'Model']]
y = df_clean['Electric Range']

ohe = OneHotEncoder(drop='first', sparse_output=False)  # drop='first' avoids multicollinearity
X_ohe = ohe.fit_transform(X_cat)
ohe_feature_names = ohe.get_feature_names_out(['Make', 'Model'])
X_ohe_df = pd.DataFrame(X_ohe, columns=ohe_feature_names)

print("One-Hot Encoded features shape:", X_ohe_df.shape)
print("Sample columns:", X_ohe_df.columns[:5].tolist())
```

```
One-Hot Encoded features shape: (270257, 228)
Sample columns: ['Make_ALFA ROMEO', 'Make_AUDI', 'Make_AZURE DYNAMICS', 'Make_BENTLEY', 'Make_BMW']
```

[17]:
```python
# Combine with numeric features
X_numeric = df_clean[['Model Year']]
X_final = pd.concat([X_numeric.reset_index(drop=True), X_ohe_df.reset_index(drop=True)], axis=1)

# Train-test split and model
X_train, X_test, y_train, y_test = train_test_split(X_final, y.reset_index(drop=True),
                                                    test_size=0.3, random_state=42)
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

print(f"R² Score: {r2_score(y_test, y_pred):.3f}")
print("Top Make coefficients:")
make_coefs = {col: coef for col, coef in zip(X_final.columns, model.coef_) if 'Make_' in col}
print(sorted(make_coefs.items(), key=lambda x: x[1], reverse=True)[:3])

# METHOD 2: LABEL ENCODING (Alternative, simpler but assumes order)
le_make = LabelEncoder()
le_model = LabelEncoder()
df_clean['Make_encoded'] = le_make.fit_transform(df_clean['Make'])
df_clean['Model_encoded'] = le_model.fit_transform(df_clean['Model'])

X_label = df_clean[['Model Year', 'Make_encoded', 'Model_encoded']]
X_train_label, X_test_label, y_train, y_test = train_test_split(X_label, y, test_size=0.3, random_state=42)
model_label = LinearRegression().fit(X_train_label, y_train)
print(f"Label Encoding R²: {r2_score(y_test, model_label.predict(X_test_label)):.3f}")
```

```
R² Score: 0.513
Top Make coefficients:
[('Make_JAGUAR', np.float64(67.72202630810501)), ('Make_LAND ROVER', np.float64(38.050676780855014)), ('Make_BENTLEY', np.float64(37.17683097158081))]
Label Encoding R²: 0.304
```

What is the R² score of the model, and what does it indicate about prediction accuracy?

[18]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

d1 = df1[['Model Year', 'Electric Range', 'Legislative District']].dropna()
X = d1[['Model Year', 'Legislative District']]
y = d1['Electric Range']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)

print(f"R² Score: {r2:.4f}")
```

```
R² Score: 0.2959
```

What steps are needed to improve the accuracy of the Linear Regression model?

[20]:
```python
# Add categorical features via one-hot encoding
df_clean = df1[['Model Year', 'Make', 'Electric Vehicle Type', 'Electric Range']].dropna()

# One-hot encode Make and Vehicle Type
X = pd.get_dummies(df_clean[['Model Year', 'Make', 'Electric Vehicle Type']], drop_first=True)
y = df_clean['Electric Range']
```

[21]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score

# Enhanced feature set
features = ['Model Year', 'Make', 'Electric Vehicle Type',
            'Clean Alternative Fuel Vehicle (CAFV) Eligibility', 'Legislative District']
df_enhanced = pd.get_dummies(df[features + ['Electric Range']].dropna(), drop_first=True)
X = df_enhanced.drop('Electric Range', axis=1)
y = df_enhanced['Electric Range']

# Scale features and split
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Compare models
models = {
    'Linear': LinearRegression(),
    'Ridge': Ridge(alpha=1.0),
    'Lasso': Lasso(alpha=0.1)
}

for name, model in models.items():
    model.fit(X_train, y_train)
    r2 = r2_score(y_test, model.predict(X_test))
    cv_score = cross_val_score(model, X_train, y_train, cv=3).mean()
    print(f"{name} - Test R²: {r2:.3f}, CV R²: {cv_score:.3f}")
```

```
Linear - Test R²: 0.914, CV R²: 0.915
Ridge - Test R²: 0.914, CV R²: 0.915
Lasso - Test R²: 0.914, CV R²: 0.914
```

Can we use this model to predict the range of new EV models based on their specifications?

[26]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score
import warnings; warnings.filterwarnings('ignore')


print(f"Original shape: {df.shape}")
print("NaN in Electric Range:", df['Electric Range'].isna().sum())

# **CRITICAL FIX 1: Remove rows where target (y) is NaN**
df_clean = df.dropna(subset=['Electric Range'])  # Drop rows with NaN target
print(f"Clean shape (y fixed): {df_clean.shape}")

# Define features
numeric_features = ['Model Year', 'Legislative District']
categorical_features = ['Make', 'Electric Vehicle Type',
                        'Clean Alternative Fuel Vehicle (CAFV) Eligibility']

# **CRITICAL FIX 2: Select only available columns**
available_numeric = [col for col in numeric_features if col in df_clean.columns]
available_categorical = [col for col in categorical_features if col in df_clean.columns]
print(f"Using numeric: {available_numeric}")
print(f"Using categorical: {available_categorical}")

# Prepare X and y
X = df_clean[available_numeric + available_categorical]
y = df_clean['Electric Range']  # Now guaranteed no NaN

# **FIXED Pipeline with imputation**
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline([
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler())
        ]), available_numeric),

        ('cat', Pipeline([
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('encoder', OneHotEncoder(drop='first', handle_unknown='ignore'))
        ]), available_categorical)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', Ridge(alpha=1.0))
])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

r2 = r2_score(y_test, y_pred)
print(f"✅ SUCCESS - Test R²: {r2:.3f}")
print(f"Train R²: {r2_score(y_train, pipeline.predict(X_train)):.3f}")
```

```
Original shape: (270262, 16)
NaN in Electric Range: 5
Clean shape (y fixed): (270257, 16)
Using numeric: ['Model Year', 'Legislative District']
Using categorical: ['Make', 'Electric Vehicle Type', 'Clean Alternative Fuel Vehicle (CAFV) Eligibility']
✅ SUCCESS - Test R²: 0.914
Train R²: 0.915
```

[28]:
```
pip install nbconvert
```

```
Requirement already satisfied: nbconvert in c:\users\lenovo\anaconda3\lib\site-packages (7.16.6)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: beautifulsoup4 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (4.12.3)
Requirement already satisfied: bleach!=5.0.0 in c:\users\lenovo\anaconda3\lib\site-packages (from bleach[css]!=5.0.0->nbconvert) (6.2.0)
Requirement already satisfied: defusedxml in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (3.1.6)
Requirement already satisfied: jupyter-core>=4.7 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (0.3.0)
Requirement already satisfied: markupsafe>=2.0 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (3.0.2)
Requirement already satisfied: mistune<4,>=2.0.3 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (3.1.2)
Requirement already satisfied: nbclient>=0.5.0 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (0.10.2)
Requirement already satisfied: nbformat>=5.7 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (5.10.4)
Requirement already satisfied: packaging in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (24.2)
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (2.19.1)
Requirement already satisfied: traitlets>=5.1 in c:\users\lenovo\anaconda3\lib\site-packages (from nbconvert) (5.14.3)
Requirement already satisfied: webencodings in c:\users\lenovo\anaconda3\lib\site-packages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert) (0.5.1)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in c:\users\lenovo\anaconda3\lib\site-packages (from bleach[css]!=5.0.0->nbconvert) (1.4.0)
Requirement already satisfied: platformdirs>=2.5 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-core>=4.7->nbconvert) (4.3.7)
Requirement already satisfied: pywin32>=300 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-core>=4.7->nbconvert) (308)
Requirement already satisfied: jupyter-client>=6.1.12 in c:\users\lenovo\anaconda3\lib\site-packages (from nbclient>=0.5.0->nbconvert) (8.6.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.9.0.post0)
Requirement already satisfied: pyzmq>=23.0 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (26.2.0)
Requirement already satisfied: tornado>=6.2 in c:\users\lenovo\anaconda3\lib\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.5.1)
Requirement already satisfied: fastjsonschema>=2.15 in c:\users\lenovo\anaconda3\lib\site-packages (from nbformat>=5.7->nbconvert) (2.20.0)
Requirement already satisfied: jsonschema>=2.6 in c:\users\lenovo\anaconda3\lib\site-packages (from nbformat>=5.7->nbconvert) (4.23.0)
Requirement already satisfied: attrs>=22.2.0 in c:\users\lenovo\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (24.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in c:\users\lenovo\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in c:\users\lenovo\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in c:\users\lenovo\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.22.3)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (1.17.0)
Requirement already satisfied: soupsieve>1.2 in c:\users\lenovo\anaconda3\lib\site-packages (from beautifulsoup4->nbconvert) (2.5)
```

[ ]: