In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
import plotly.express as px
```

In [2]:
```
df = pd.read_csv('dataset.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	Electric Range	Base MSRP	Legislative District	Vel
0	JTMEB3FV6N	Monroe	Key West	FL	33040	2022	TOYOTA	RAV4 PRIME	Plug-in Hybrid Electric Vehicle (PHEV)	Clean Alternative Fuel Vehicle Eligible	42	0	NaN	198
1	1G1RD6E45D	Clark	Laughlin	NV	89029	2013	CHEVROLET	VOLT	Plug-in Hybrid Electric Vehicle (PHEV)	Clean Alternative Fuel Vehicle Eligible	38	0	NaN	5
2	JN1AZ0CP8B	Yakima	Yakima	WA	98901	2011	NISSAN	LEAF	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	73	0	15.0	218
3	1G1FW6S08H	Skagit	Concrete	WA	98237	2017	CHEVROLET	BOLT EV	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	238	0	39.0	186
4	3FA6P0SU1K	Snohomish	Everett	WA	98201	2019	FORD	FUSION	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	26	0	38.0	2

In [4]:
```
df.shape
```

Out[4]:
```
(112634, 17)
```

In [5]:
```
df.describe()
```

Out[5]:

	Postal Code	Model Year	Electric Range	Base MSRP	Legislative District	DOL Vehicle ID	2020 Census Tract
count	112634.000000	112634.000000	112634.000000	112634.000000	112348.000000	1.126340e+05	1.126340e+05
mean	98156.226850	2019.003365	87.812987	1793.439681	29.805604	1.994567e+08	5.296650e+10
std	2648.733064	2.892364	102.334216	10783.753486	14.700545	9.398427e+07	1.699104e+09
min	1730.000000	1997.000000	0.000000	0.000000	1.000000	4.777000e+03	1.101001e+09
25%	98052.000000	2017.000000	0.000000	0.000000	18.000000	1.484142e+08	5.303301e+10
50%	98119.000000	2020.000000	32.000000	0.000000	34.000000	1.923896e+08	5.303303e+10
75%	98370.000000	2022.000000	208.000000	0.000000	43.000000	2.191899e+08	5.305307e+10
max	99701.000000	2023.000000	337.000000	845000.000000	49.000000	4.792548e+08	5.603300e+10

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

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

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112634 entries, 0 to 112633
Data columns (total 17 columns):
 #   Column                                             Non-Null Count   Dtype  
---  ------                                             --------------   -----  
 0   VIN (1-10)                                         112634 non-null  object 
 1   County                                             112634 non-null  object 
 2   City                                               112634 non-null  object 
 3   State                                              112634 non-null  object 
 4   Postal Code                                        112634 non-null  int64  
 5   Model Year                                         112634 non-null  int64  
 6   Make                                               112634 non-null  object 
 7   Model                                              112614 non-null  object 
 8   Electric Vehicle Type                              112634 non-null  object 
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  112634 non-null  object 
 10  Electric Range                                     112634 non-null  int64  
 11  Base MSRP                                          112634 non-null  int64  
 12  Legislative District                               112348 non-null  float64
 13  DOL Vehicle ID                                     112634 non-null  int64  
 14  Vehicle Location                                   112610 non-null  object 
 15  Electric Utility                                   112191 non-null  object 
 16  2020 Census Tract                                  112634 non-null  int64  
dtypes: float64(1), int64(6), object(10)
memory usage: 14.6+ MB
```

In [8]:
```
# fiding the duplicate columns.
df.duplicated().sum()
```

Out[8]:
```
0
```

In [9]:
```
# fiding the null values in the each columns with count
null_count = df.isna().sum()

print("Count of NaN or null values in each column:")
print(null_count)
```

```
Count of NaN or null values in each column:
VIN (1-10)                                             0
County                                                 0
City                                                   0
State                                                  0
Postal Code                                            0
Model Year                                             0
Make                                                   0
Model                                                 20
Electric Vehicle Type                                  0
Clean Alternative Fuel Vehicle (CAFV) Eligibility      0
Electric Range                                         0
Base MSRP                                              0
Legislative District                                 286
DOL Vehicle ID                                         0
Vehicle Location                                      24
Electric Utility                                     443
2020 Census Tract                                      0
dtype: int64
```

In [10]:
```
rows_with_null = df[df.isna().any(axis=1)]

rows_with_null.head(10)
```

Out[10]:

	VIN (1-10)	County	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	Electric Range	Base MSRP	Legislative District
0	JTMEB3FV6N	Monroe	Key West	FL	33040	2022	TOYOTA	RAV4 PRIME	Plug-in Hybrid Electric Vehicle (PHEV)	Clean Alternative Fuel Vehicle Eligible	42	0	NaN
1	1G1RD6E45D	Clark	Laughlin	NV	89029	2013	CHEVROLET	VOLT	Plug-in Hybrid Electric Vehicle (PHEV)	Clean Alternative Fuel Vehicle Eligible	38	0	NaN
									Battery Electric	Clean Alternative			

In [11]:
```
# Droping rows with any NaN values
df = df.dropna(axis=0)
```

In [12]:
```
#Note: we can see the shape of the data frame has been changed because NaN values are removed.
df.shape
```

Out[12]:
```
(112152, 17)
```

In [13]:
```
# checking  the null values in the each columns with count
null_count = df.isna().sum()

print("Count of NaN or null values in each column:")
print(null_count)
```

```
Count of NaN or null values in each column:
VIN (1-10)                                         0
County                                             0
City                                               0
State                                              0
Postal Code                                        0
Model Year                                         0
Make                                               0
Model                                              0
Electric Vehicle Type                              0
Clean Alternative Fuel Vehicle (CAFV) Eligibility  0
Electric Range                                     0
Base MSRP                                          0
Legislative District                               0
DOL Vehicle ID                                     0
Vehicle Location                                   0
Electric Utility                                   0
2020 Census Tract                                  0
dtype: int64
```

In [14]:
```
df.describe()
```

Out[14]:

	Postal Code	Model Year	Electric Range	Base MSRP	Legislative District	DOL Vehicle ID	2020 Census Tract
count	112152.000000	112152.000000	112152.000000	112152.000000	112152.000000	1.121520e+05	1.121520e+05
mean	98258.856659	2019.004494	87.829651	1793.882320	29.817703	1.994712e+08	5.303958e+10
std	302.889935	2.891859	102.336645	10785.259118	14.698726	9.401842e+07	1.617788e+07
min	98001.000000	1997.000000	0.000000	0.000000	1.000000	4.777000e+03	5.300195e+10
25%	98052.000000	2017.000000	0.000000	0.000000	18.000000	1.484164e+08	5.303301e+10
50%	98121.000000	2020.000000	32.000000	0.000000	34.000000	1.923916e+08	5.303303e+10
75%	98370.000000	2022.000000	208.000000	0.000000	43.000000	2.191885e+08	5.305307e+10
max	99403.000000	2023.000000	337.000000	845000.000000	49.000000	4.792548e+08	5.307794e+10

In [ ]:

TASK_1 (Univariate and Bivariate)

Non - Visual Analysis

In [15]:
```
# Find nunique, unique values, and value counts
nunique_val, unique_vals, value_counts = df['County'].nunique(), df['County'].unique(), df['County'].value_cour

print("Number of unique values:", nunique_val)
print("Value counts:\n", value_counts)
print("Unique values:", unique_vals)
```

```
Number of unique values: 39
Value counts:
 King            58980
Snohomish       12412
Pierce           8525
Clark            6681
Thurston         4109
Kitsap           3828
Whatcom          2839
Spokane          2785
Benton           1376
Island           1298
Skagit           1228
Clallam           728
San Juan          717
Jefferson         698
Chelan            654
Yakima            617
Cowlitz           569
Mason             547
Lewis             431
Grays Harbor      402
Kittitas          392
Franklin          365
Grant             335
Walla Walla       312
Douglas           221
Whitman           177
Klickitat         175
Okanogan          149
Pacific           145
Skamania          139
Stevens            91
Asotin             48
Wahkiakum          39
Adams              34
Pend Oreille       32
Lincoln            30
Ferry              27
Columbia           13
Garfield            4
Name: County, dtype: int64
Unique values: ['Yakima' 'Skagit' 'Snohomish' 'Island' 'Thurston' 'Grant' 'King' 'Kitsap'
 'Whitman' 'Spokane' 'Cowlitz' 'Pierce' 'Kittitas' 'Grays Harbor' 'Clark'
 'Chelan' 'Whatcom' 'Benton' 'Walla Walla' 'Mason' 'San Juan' 'Lewis'
 'Jefferson' 'Clallam' 'Douglas' 'Klickitat' 'Skamania' 'Adams' 'Franklin'
 'Okanogan' 'Stevens' 'Asotin' 'Ferry' 'Pacific' 'Columbia' 'Wahkiakum'
 'Lincoln' 'Pend Oreille' 'Garfield']
```

In [16]:
```
nunique_val, unique_vals, value_counts = df['City'].nunique(), df['City'].unique(), df['City'].value_counts()

print("Number of unique values:", nunique_val)
print("Value counts:\n", value_counts)
print("Unique values:", unique_vals)
```

```
Number of unique values: 435
Value counts:
 Seattle           20295
Bellevue           5919
Redmond            4199
Vancouver          4013
Kirkland           3598
                  ...
Walla Walla Co        1
Clallam Bay           1
Malott                1
Rockport              1
Uniontown             1
Name: City, Length: 435, dtype: int64
Unique values: ['Yakima' 'Concrete' 'Everett' 'Bothell' 'Mukilteo' 'Clinton' 'Anacortes'
 'Lacey' 'Moses Lake' 'Rochester' 'Burlington' 'Marysville' 'Lynnwood'
 'Edmonds' 'Olympia' 'Seattle' 'Auburn' 'Langley' 'Snohomish' 'Bremerton'
 'Pullman' 'Spokane' 'Suquamish' 'Monroe' 'Keyport' 'Maple Valley' 'Kent'
 'Lake Forest Park' 'Poulsbo' 'Redmond' 'Issaquah' 'Longview' 'Tacoma'
 'Ellensburg' 'Burien' 'Gig Harbor' 'South Hill' 'Sammamish' 'Westport'
 'Vancouver' 'Airway Heights' 'Mercer Island' 'Stanwood' 'Tumwater'
 'Bainbridge Island' 'Entiat' 'Lakewood' 'Lake Tapps' 'Bellevue'
 'Kirkland' 'Newcastle' 'Port Orchard' 'Bellingham' 'Richland'
 'Camano Island' 'Wenatchee' 'Lake Stevens' 'Roy' 'Des Moines' 'Renton'
 'Camas' 'Kennewick' 'Battle Ground' 'Bonney Lake' 'Walla Walla'
 'North Bend' 'Mount Vernon' 'Woodland' 'Woodinville' 'Allyn' 'Brier'
 'Snoqualmie' 'Fall City' 'Puyallup' 'Friday Harbor' 'Point Roberts'
 'Dupont' 'Castle Rock' 'Blaine' 'Morton' 'Port Townsend' 'Roslyn'
 'Kenmore' 'Covington' 'Federal Way' 'Silverdale' 'Medina' 'Shoreline'
 'Enumclaw' 'Sequim' 'Orondo' 'Grandview' 'Mill Creek' 'Zillah' 'Edgewood'
 'Vashon' 'White Salmon' 'Normandy Park' 'Fircrest' 'East Wenatchee'
 'Peshastin' 'Grapeview' 'Steilacoom' 'Sumner' 'Greenacres' 'Shelton'
 'Chehalis' 'Pacific Beach' 'Everson' 'Black Diamond' 'North Bonneville'
 'Coupeville' 'Seabeck' 'Arlington' 'Palouse' 'Bow' 'Lakebay'
 'University Place' 'Clyde Hill' 'Cle Elum' 'Yacolt' 'Oak Harbor'
 'Goldendale' 'Port Hadlock' 'Acme' 'Ritzville' 'Union' 'Orting' 'Tahuya'
 'Fox Island' 'Moxee' 'Port Angeles' 'Spanaway' 'Lopez Island'
 'Hunts Point' 'Leavenworth' 'Seatac' 'Stevenson' 'Pasco' 'Yelm'
 'Tonasket' 'Liberty Lake' 'Hansville' 'Eastsound' 'Nordland' 'Touchet'
 'Spokane Valley' 'Tukwila' 'Nine Mile Falls' 'Selah' 'Fife' 'Lynden'
 'Aberdeen' 'Anderson Island' 'Orcas Is' 'Kingston' 'Randle'
 'Sedro-Woolley' 'Carnation' 'Belfair' 'Cheney' 'Elma' 'Olalla'
 'Granite Falls' 'Ephrata' 'Preston' 'Ridgefield' 'Mccleary' 'Ferndale'
 'Mountlake Terrace' 'Freeland' 'Yarrow Point' 'Rainier' 'Sunnyside'
 'Salkum' 'Colville' 'Duvall' 'Otis Orchards' 'Twisp' 'Eatonville'
 'Chattaroy' 'Ocean Shores' 'Washougal' 'Port Ludlow' 'Benton City'
 'Clarkston' 'Ravensdale' 'Kelso' 'Curlew' 'Deming' 'Prosser' 'Milton'
 'Artondale' 'Hoodsport' 'West Richland' 'Parkland' 'Chelan' 'Graham'
 'Raymond' 'Brush Prairie' 'Rock Island' 'La Conner' 'St John' 'Mead'
 'Hoquiam' 'Deer Park' 'Electric City' 'Chimacum' 'Burbank' 'Quincy'
 'La Center' 'Ronald' 'Long Beach' 'Valley' 'Beaux Arts' 'Kalama'
 'Indianola' 'Winthrop' 'Buckley' 'Montesano' 'Dayton' 'Vaughn' 'Onalaska'
 'Medical Lake' 'Nooksack' 'Centralia' 'Sultan' 'Trout Lake' 'Seaview'
 'Carson' 'Colbert' 'Lummi Island' 'Newman Lake' 'Cathlamet' 'Woodway'
 'Veradale' 'Valleyford' 'Cashmere' 'Ariel' 'Cosmopolis' 'Bz Corner'
 'Ilwaco' 'Oakville' 'Algona' 'Silverlake' 'Lopez Is' 'Winlock'
 'Greenbank' 'Tenino' 'Royal City' 'Tulalip' 'Custer' 'College Place'
 'Underwood' 'Amboy' 'Bingen' 'Ryderwood' 'Clearlake' 'Naches' 'Surfside'
 'Olga' 'Ocean Park' 'Othello' 'Rosalia' 'Snoqualmie Pass' 'Republic'
 'Grand Coulee' 'Chewelah' 'Packwood' 'Thorp' 'Malaga' 'Lind'
 'Joint Base Lewis Mcchord' 'Granger' 'Wilbur' 'Toledo' 'Pacific'
 'Toppenish' 'Eltopia' 'Sekiu' 'Sedro Woolley' 'Garfield' 'Lincoln'
 'Newport' 'Harrington' 'Ethel' 'Pomeroy' 'Longbranch' 'Connell' 'Brinnon'
 'Skykomish' 'Reardan' 'Maple Falls' 'Coulee City' 'Dallesport' 'Vantage'
 'Oroville' 'Manson' 'Omak' 'Bridgeport Bar' 'Mesa' 'Waterville' 'Chinook'
 'Gold Bar' 'Soap Lake' 'Nahcotta' 'Tieton' 'Mattawa' 'Addy' 'Ruston'
 'Forks' 'Wapato' 'Naselle' 'Quilcene' 'Asotin' 'Easton'
 'Fairchild Air Force Base' 'Skamokawa' 'Lilliwaup' 'Marlin' 'Warden'
 'Seven Bays' 'Kettle Falls' 'South Bend' 'Okanogan' 'Mansfield' 'Pateros'
 'Sumas' 'McCleary' 'Cusick' 'Ashford' 'Elk' 'Carbonado' 'Rockford' 'Lyle'
 'Latah' 'Carlton' 'Darrington' 'Mossyrock' 'Riverside' 'North Cove'
 'Bay Center' 'Brewster' 'Springdale' 'Cougar' 'Endicott' 'Inchelium'
 'Frances' 'Colfax' 'White Swan' 'Grayland' 'Rice' 'Neah Bay' 'Davenport'
 'Coulee Dam' 'Union Gap' 'Shaw Island' 'Marblemount' 'Baring' 'Spangle'
 'Glacier' 'Deer Meadows' 'Silver Creek' 'Menlo' 'Tokeland' 'Roosevelt'
 'Wahkiacus' 'Snowden' 'Walla Walla Co' 'Outlook' 'Vader' 'Kittitas'
 'Mica' 'Mazama' 'Hunters' 'Evans' 'Beaver' 'Loon Lake' 'Grays River'
 'Odessa' 'Usk' 'Oysterville' 'Mineral' 'Amanda Park' 'Toutle' 'Curtis'
 'Cinebar' 'Hartline' 'Waitsburg' 'Husum' 'Klickitat' 'Edwall' 'Sprague'
 'Tekoa' 'Pe Ell' 'Methow' 'Murdock' 'Ford' 'Moclips' 'Holden Village'
 'Lyman' 'Mcchord Afb' 'Glenwood' 'Colton' 'South Prairie' 'Clallam Bay'
 'Malott' 'Rockport' 'Bucoda' 'Smith Creek' 'Lebam' 'Glenoma'
```

```
 'Copalis Beach' 'Satsop' 'Palisades' 'Danville' 'Quinault' 'Maryhill'
 'Waldron' 'Fruitland' 'South Cle Elum' 'Centerville' 'Fairfield' 'Lamont'
 'Copalis Crossing' 'Taholah' 'Port Gamble' 'Rosburg' 'Stratford'
 'Matlock' 'Gifford' 'Prescott' 'Carrolls' 'Mabton' 'Uniontown']
```

In [17]:
```
nunique_val, unique_vals, value_counts = df['Make'].nunique(), df['Make'].unique(), df['Make'].value_counts()

print("Number of unique values:", nunique_val)
print("Value counts:\n", value_counts)
print("Unique values:", unique_vals)
```

```
Number of unique values: 34
Value counts:
 TESLA             51883
NISSAN             12846
CHEVROLET          10140
FORD                5780
BMW                 4660
KIA                 4469
TOYOTA              4368
VOLKSWAGEN          2507
AUDI                2320
VOLVO               2256
CHRYSLER            1780
HYUNDAI             1407
JEEP                1143
RIVIAN               883
FIAT                 820
PORSCHE              817
HONDA                788
MINI                 631
MITSUBISHI           585
POLESTAR             557
MERCEDES-BENZ        503
SMART                271
JAGUAR               218
LINCOLN              167
CADILLAC             108
LUCID MOTORS          65
SUBARU                59
LAND ROVER            38
LEXUS                 33
FISKER                19
GENESIS               18
AZURE DYNAMICS         7
TH!NK                  3
BENTLEY                3
Name: Make, dtype: int64
Unique values: ['NISSAN' 'CHEVROLET' 'FORD' 'TESLA' 'KIA' 'AUDI' 'BMW' 'PORSCHE' 'FIAT'
 'CADILLAC' 'MITSUBISHI' 'CHRYSLER' 'RIVIAN' 'HONDA' 'HYUNDAI' 'VOLVO'
 'VOLKSWAGEN' 'TOYOTA' 'MERCEDES-BENZ' 'JEEP' 'MINI' 'SMART' 'SUBARU'
 'POLESTAR' 'LUCID MOTORS' 'LINCOLN' 'JAGUAR' 'FISKER' 'LAND ROVER'
 'LEXUS' 'TH!NK' 'GENESIS' 'BENTLEY' 'AZURE DYNAMICS']
```

In [18]:
```
nunique_val, unique_vals, value_counts = df['Model'].nunique(), df['Model'].unique(), df['Model'].value_counts(

print("Number of unique values:", nunique_val)
print("Value counts:\n", value_counts)
print("Unique values:", unique_vals)
```

```
Number of unique values: 114
Value counts:
 MODEL 3        23042
MODEL Y        17086
LEAF           12846
MODEL S         7346
BOLT EV         4895
               ...
745LE              2
S-10 PICKUP        1
SOLTERRA           1
918                1
FLYING SPUR        1
Name: Model, Length: 114, dtype: int64
Unique values: ['LEAF' 'BOLT EV' 'FUSION' 'MODEL 3' 'SOUL' 'Q5 E' 'MODEL X' 'VOLT' 'X5'
 '530E' 'TAYCAN' 'X3' 'A3' 'SOUL EV' 'C-MAX' '500' 'MODEL S' 'F-150' 'CT6'
 'I3' 'MODEL Y' 'NIRO' 'OUTLANDER' 'PACIFICA' 'R1T' 'CLARITY'
 'KONA ELECTRIC' 'XC40' 'ID.4' 'PRIUS PLUG-IN' 'MUSTANG MACH-E'
 'EQB-CLASS' 'RAV4 PRIME' 'E-GOLF' 'PRIUS PRIME' 'C40' 'SORENTO' 'XC60'
 'CAYENNE' 'WRANGLER' 'COUNTRYMAN' 'S60' 'EV6' 'FORTWO ELECTRIC DRIVE'
 'GRAND CHEROKEE' '330E' 'CROSSTREK' 'IONIQ 5' 'IONIQ' 'E-TRON' 'ROADSTER'
 'KONA' 'XC90' 'SPARK' 'PS2' 'A7' 'HARDTOP' 'ESCAPE' 'LUCID AIR'
 'E-TRON SPORTBACK' 'Q5' 'RAV4' 'AVIATOR' 'E-TRON GT' 'EDV' 'IX' 'FORTWO'
 'I-PACE' 'SANTA FE' 'B-CLASS' 'KARMA' 'I4' 'OPTIMA' 'GLC-CLASS' 'Q4'
 'SONATA' 'EQ FORTWO' 'FOCUS' 'RANGE ROVER SPORT' 'TRANSIT' 'PANAMERA'
 'I8' 'BOLT EUV' 'CORSAIR' 'ELR' 'GLE-CLASS' 'V60' 'EQS-CLASS SEDAN' 'R1S'
 'I-MIEV' 'NX' '740E' 'SPORTAGE' 'C-CLASS' 'S-CLASS' 'CITY' 'S90' 'TUCSON'
 'GV60' 'EQS-CLASS SUV' 'A8 E' 'RANGE ROVER' 'RS E-TRON GT' 'RANGER'
 'BENTAYGA' '745E' 'TRANSIT CONNECT ELECTRIC' 'ACCORD' 'S-10 PICKUP'
 'SOLTERRA' 'G80' '918' 'FLYING SPUR' '745LE']
```

In [19]:
```
nunique_val, unique_vals, value_counts = df['Electric Vehicle Type'].nunique(), df['Electric Vehicle Type'].uni

print("Number of unique values:", nunique_val)
print("Value counts:\n", value_counts)
print("Unique values:", unique_vals)
```

```
Number of unique values: 2
Value counts:
 Battery Electric Vehicle (BEV)            85732
Plug-in Hybrid Electric Vehicle (PHEV)    26420
Name: Electric Vehicle Type, dtype: int64
Unique values: ['Battery Electric Vehicle (BEV)' 'Plug-in Hybrid Electric Vehicle (PHEV)']
```

observations

Diversity of EVs:

We have electric vehicles (EVs) from 39 different nations. King County has the most EVs (5,890). Garfield has the fewest EVs (4).

Cities Represented:

There are 20,295 EVs from 435 different cities. Seattle has the highest number of EVs.

Car Manufacturers:

The dataset includes 34 distinct EV car manufacturers. Tesla has the most vehicles with 51,883 EVs. Bentley has at least three EVs.

Unique Vehicle Numbers:

The dataset contains 114 distinct numbers from all EV companies models.

Vehicle Ratings:

Compared to Plug-in Hybrid Electric Vehicles (PHEVs), the majority of Battery Electric Vehicles (BEVs) have an 85732 rating.

In [ ]:

In [20]:

```
def numerical_univariate_analysis(numerical_data):
    for col_name in numerical_data:
        print("*"*10, col_name, "*"*10)
        print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))
        print()
```

In [21]:
```
numerical_univariate_analysis(df)
```

```
********** VIN (1-10) **********

C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))

min    1C4JJXP60M
max    YV4H60DZ9N
Name: VIN (1-10), dtype: object

********** County **********

C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))

min     Adams
max    Yakima
Name: County, dtype: object

********** City **********

C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))

min    Aberdeen
max      Zillah
Name: City, dtype: object

********** State **********

C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))

min    WA
max    WA
Name: State, dtype: object

********** Postal Code **********
min       98001.000000
max       99403.000000
mean      98258.856659
median    98121.000000
std         302.889935
Name: Postal Code, dtype: float64

********** Model Year **********
min       1997.000000
max       2023.000000
mean      2019.004494
median    2020.000000
std          2.891859
Name: Model Year, dtype: float64

********** Make **********

C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))

min     AUDI
max    VOLVO
Name: Make, dtype: object

********** Model **********

C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))
```

```
min    330E
max    XC90
Name: Model, dtype: object

********** Electric Vehicle Type **********

C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))

min            Battery Electric Vehicle (BEV)
max    Plug-in Hybrid Electric Vehicle (PHEV)
Name: Electric Vehicle Type, dtype: object

********** Clean Alternative Fuel Vehicle (CAFV) Eligibility **********

C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))

min    Clean Alternative Fuel Vehicle Eligible
max      Not eligible due to low battery range
Name: Clean Alternative Fuel Vehicle (CAFV) Eligibility, dtype: object

********** Electric Range **********
min         0.000000
max       337.000000
mean       87.829651
median     32.000000
std       102.336645
Name: Electric Range, dtype: float64

********** Base MSRP **********
min            0.000000
max       845000.000000
mean        1793.882320
median         0.000000
std        10785.259118
Name: Base MSRP, dtype: float64

********** Legislative District **********
min        1.000000
max       49.000000
mean      29.817703
median    34.000000
std       14.698726
Name: Legislative District, dtype: float64

********** DOL Vehicle ID **********
min       4.777000e+03
max       4.792548e+08
mean      1.994712e+08
median    1.923916e+08
std       9.401842e+07
Name: DOL Vehicle ID, dtype: float64

********** Vehicle Location **********

C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))

min    POINT (-102.69968 22.95716)
max      POINT (27.25316 67.01865)
Name: Vehicle Location, dtype: object

********** Electric Utility **********
min                                        AVISTA CORP
max    PUGET SOUND ENERGY INC||PUD NO 1 OF WHATCOM CO...
Name: Electric Utility, dtype: object

********** 2020 Census Tract **********
min       5.300195e+10
max       5.307794e+10
mean      5.303958e+10
median    5.303303e+10
std       1.617788e+07
Name: 2020 Census Tract, dtype: float64
```

```
C:\Users\VIJAY\AppData\Local\Temp\ipykernel_26668\121005930.py:4: FutureWarning: ['mean', 'median', 'std'] di
d not aggregate successfully. If any error is raised this will raise in a future version of pandas. Drop thes
e columns/ops to avoid this warning.
  print(numerical_data[col_name].agg(['min', 'max', 'mean', 'median', 'std']))
```

Visual Analysis

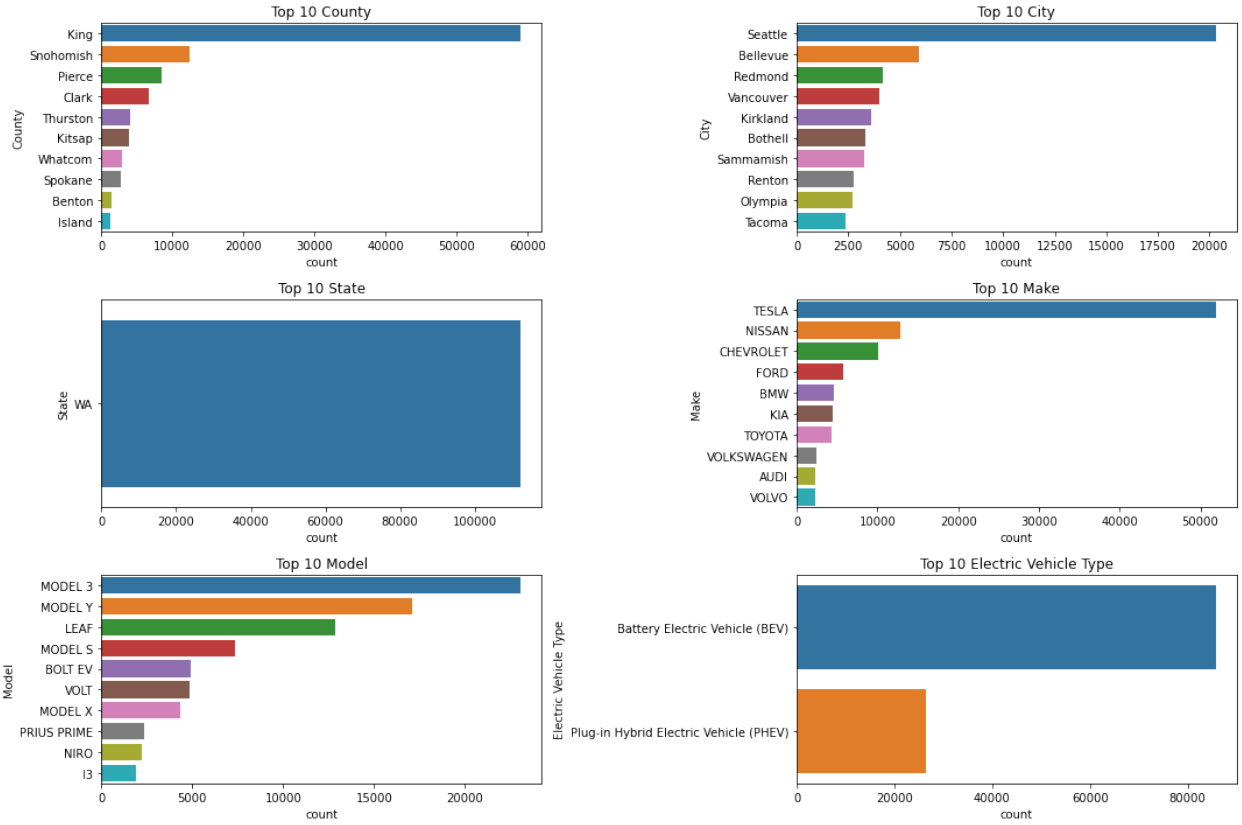
In [22]:
```
discrete_df = df.select_dtypes(include=['object'])
```

In [23]:
```
df.columns
```

Out[23]:
```
Index(['VIN (1-10)', 'County', 'City', 'State', 'Postal Code', 'Model Year',
       'Make', 'Model', 'Electric Vehicle Type',
       'Clean Alternative Fuel Vehicle (CAFV) Eligibility', 'Electric Range',
       'Base MSRP', 'Legislative District', 'DOL Vehicle ID',
       'Vehicle Location', 'Electric Utility', '2020 Census Tract'],
      dtype='object')
```

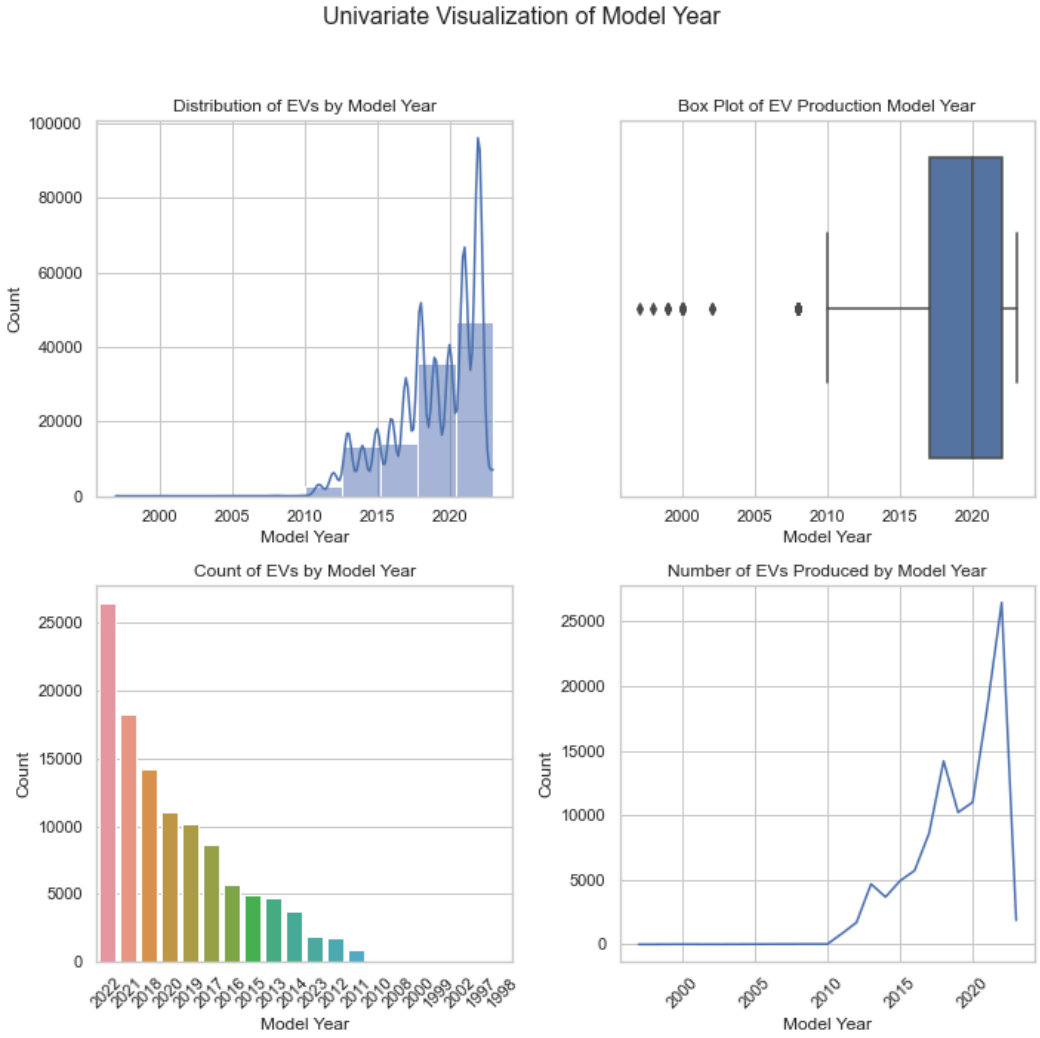
In [24]:
```
categorical_columns = [ 'County', 'City', 'State', 'Make', 'Model', 'Electric Vehicle Type','Clean Alternative
```

In [25]:
```
plt.figure(figsize=(15, 10))
for i, column in enumerate(categorical_columns[:6], 1): # Limiting to first 6 for cla
 plt.subplot(3, 2, i)
 sns.countplot(y=df[column], order=df[column].value_counts().index[:10])
 plt.title(f'Top 10 {column}')
plt.tight_layout()
plt.show()
```



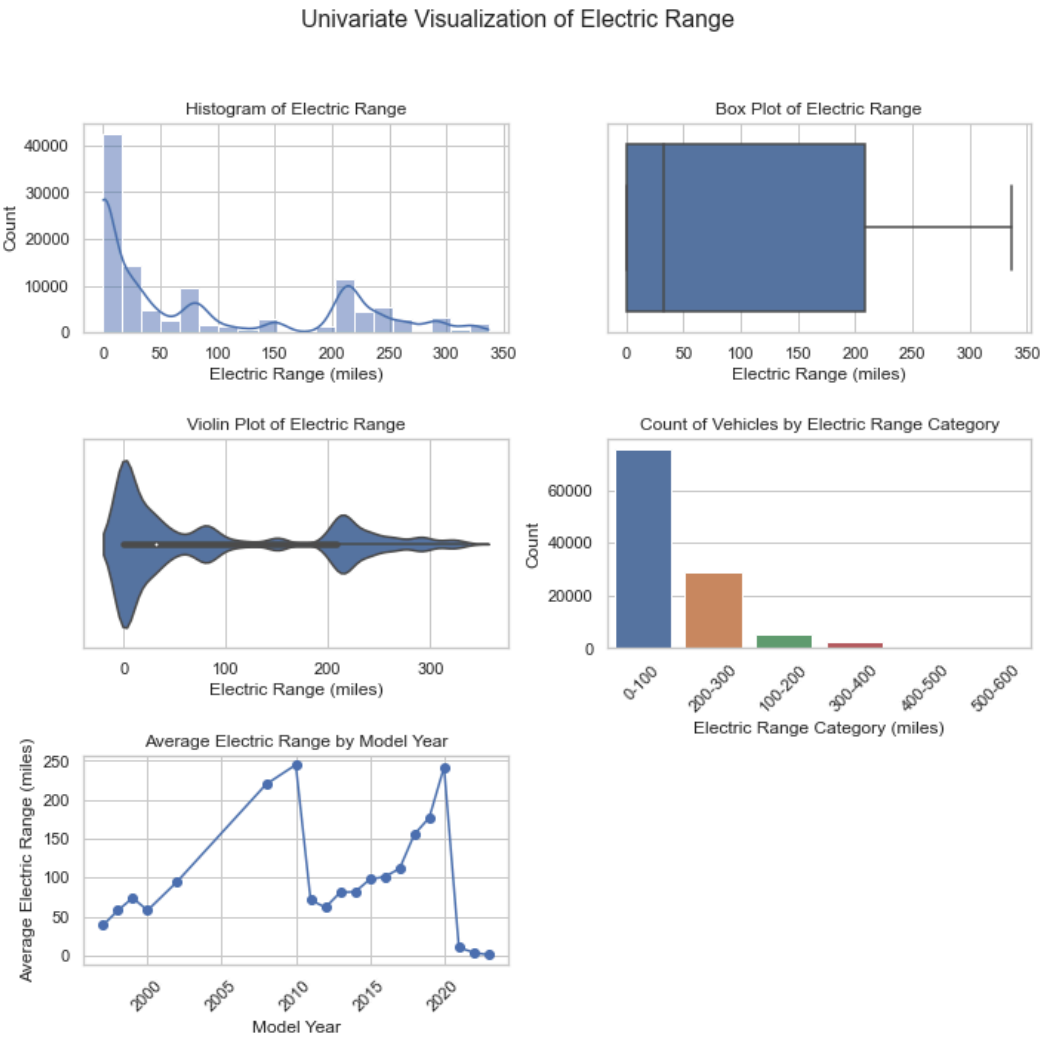
In [26]:

```
sns.set(style="whitegrid")

# Create a figure with 2 rows and 2 columns for subplots
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
fig.suptitle('Univariate Visualization of Model Year', fontsize=16)

# 1. Histogram with KDE
sns.histplot(df['Model Year'], bins=10, kde=True, ax=axs[0, 0])
axs[0, 0].set_title('Distribution of EVs by Model Year')
axs[0, 0].set_xlabel('Model Year')
axs[0, 0].set_ylabel('Count')

# 2. Box Plot
sns.boxplot(x=df['Model Year'], ax=axs[0, 1])
axs[0, 1].set_title('Box Plot of EV Production Model Year')
axs[0, 1].set_xlabel('Model Year')

# 3. Count Plot
sns.countplot(x=df['Model Year'], order=df['Model Year'].value_counts().index, ax=axs[1, 0])
axs[1, 0].set_title('Count of EVs by Model Year')
axs[1, 0].set_xlabel('Model Year')
axs[1, 0].set_ylabel('Count')
axs[1, 0].tick_params(axis='x', rotation=45)

# 4. Line Plot
yearly_counts = df['Model Year'].value_counts().sort_index()
sns.lineplot(x=yearly_counts.index, y=yearly_counts.values, ax=axs[1, 1])
axs[1, 1].set_title('Number of EVs Produced by Model Year')
axs[1, 1].set_xlabel('Model Year')
axs[1, 1].set_ylabel('Count')
axs[1, 1].tick_params(axis='x', rotation=45)

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.95])  # Make room for the main title
plt.show()
```



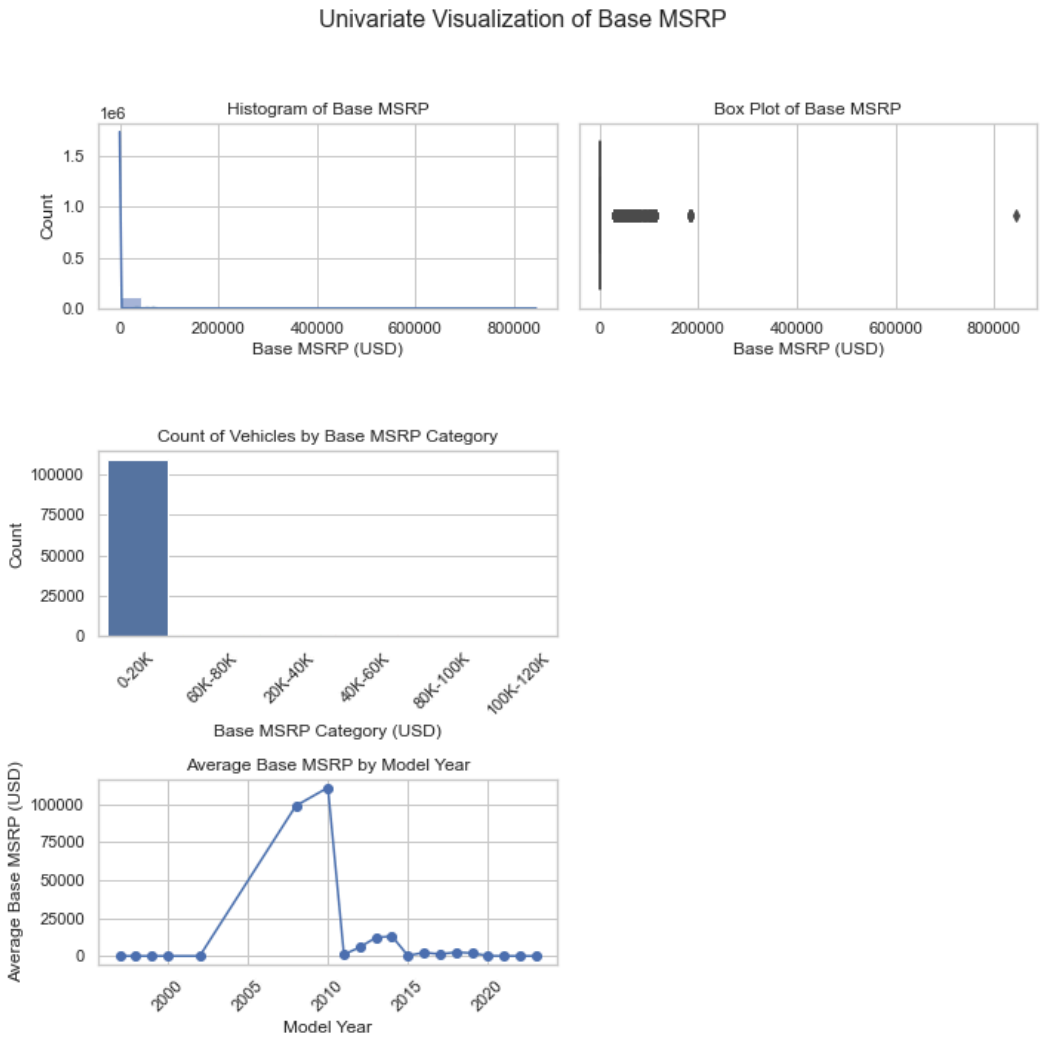
In [27]:

```
# Set the visual style
sns.set(style="whitegrid")

# Create a figure with 3 rows and 2 columns for subplots
fig, axs = plt.subplots(3, 2, figsize=(10, 10))
fig.suptitle('Univariate Visualization of Electric Range', fontsize=16)

# 1. Histogram with KDE
sns.histplot(df['Electric Range'], bins=20, kde=True, ax=axs[0, 0])
axs[0, 0].set_title('Histogram of Electric Range')
axs[0, 0].set_xlabel('Electric Range (miles)')
axs[0, 0].set_ylabel('Count')

# 2. Box Plot
sns.boxplot(x=df['Electric Range'], ax=axs[0, 1])
axs[0, 1].set_title('Box Plot of Electric Range')
axs[0, 1].set_xlabel('Electric Range (miles)')

# 3. Violin Plot
sns.violinplot(x=df['Electric Range'], ax=axs[1, 0])
axs[1, 0].set_title('Violin Plot of Electric Range')
axs[1, 0].set_xlabel('Electric Range (miles)')

# 4. Count Plot (if Electric Range is categorized)
# Define bins for Electric Range (example)
bins = [0, 100, 200, 300, 400, 500, 600]
labels = ['0-100', '100-200', '200-300', '300-400', '400-500', '500-600']
df['Range Category'] = pd.cut(df['Electric Range'], bins=bins, labels=labels, right=False)

sns.countplot(x='Range Category', data=df, order=df['Range Category'].value_counts().index, ax=axs[1, 1])
axs[1, 1].set_title('Count of Vehicles by Electric Range Category')
axs[1, 1].set_xlabel('Electric Range Category (miles)')
axs[1, 1].set_ylabel('Count')
axs[1, 1].tick_params(axis='x', rotation=45)

# 5. Line Plot (Average Electric Range by Model Year)
# Example: Assuming you want to see the average Electric Range per year
avg_range_per_year = df.groupby('Model Year')['Electric Range'].mean()

axs[2, 0].plot(avg_range_per_year.index, avg_range_per_year.values, marker='o')
axs[2, 0].set_title('Average Electric Range by Model Year')
axs[2, 0].set_xlabel('Model Year')
axs[2, 0].set_ylabel('Average Electric Range (miles)')
axs[2, 0].tick_params(axis='x', rotation=45)

# Remove empty subplot
fig.delaxes(axs[2, 1])

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.95])  # Make room for the main title
plt.show()
```


Univariate Visualization of Electric Range
Histogram of Electric Range
Count
Electric Range (miles)
Box Plot of Electric Range
Electric Range (miles)
Violin Plot of Electric Range
Electric Range (miles)
Count of Vehicles by Electric Range Category
Count
0-100
200-300
100-200
300-400
400-500
500-600
Electric Range Category (miles)
Average Electric Range by Model Year
Average Electric Range (miles)
Model Year

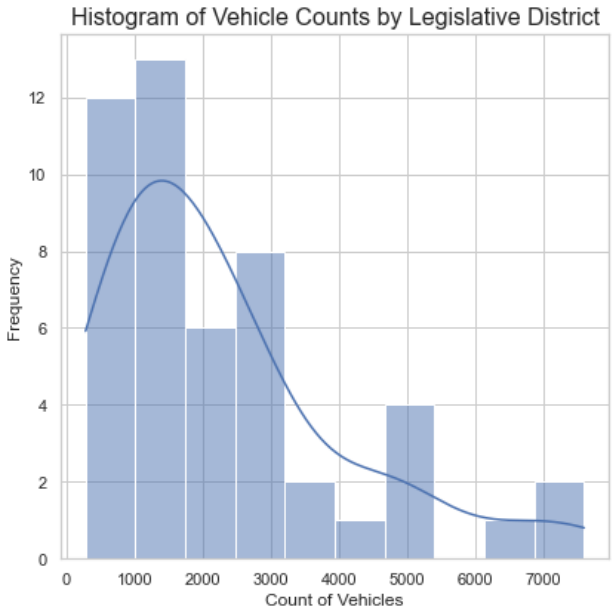
In [28]:

```
sns.set(style="whitegrid")

# Create a figure with 3 rows and 2 columns for subplots
fig, axs = plt.subplots(3, 2, figsize=(10,10))
fig.suptitle('Univariate Visualization of Base MSRP', fontsize=16)

# 1. Histogram with KDE
sns.histplot(df['Base MSRP'], bins=20, kde=True, ax=axs[0, 0])
axs[0, 0].set_title('Histogram of Base MSRP')
axs[0, 0].set_xlabel('Base MSRP (USD)')
axs[0, 0].set_ylabel('Count')

# 2. Box Plot
sns.boxplot(x=df['Base MSRP'], ax=axs[0, 1])
axs[0, 1].set_title('Box Plot of Base MSRP')
axs[0, 1].set_xlabel('Base MSRP (USD)')

# 3. Count Plot (if Base MSRP is categorized)
# Define bins for Base MSRP (example)
msrp_bins = [0, 20000, 40000, 60000, 80000, 100000, 120000]
msrp_labels = ['0-20K', '20K-40K', '40K-60K', '60K-80K', '80K-100K', '100K-120K']
df['MSRP Category'] = pd.cut(df['Base MSRP'], bins=msrp_bins, labels=msrp_labels, right=False)

sns.countplot(x='MSRP Category', data=df, order=df['MSRP Category'].value_counts().index, ax=axs[1, 0])
axs[1, 0].set_title('Count of Vehicles by Base MSRP Category')
axs[1, 0].set_xlabel('Base MSRP Category (USD)')
axs[1, 0].set_ylabel('Count')
axs[1, 0].tick_params(axis='x', rotation=45)

# 4. Line Plot (Average Base MSRP by Model Year)
# Example: Assuming you want to see the average Base MSRP per year
avg_msrp_per_year = df.groupby('Model Year')['Base MSRP'].mean()

axs[2, 0].plot(avg_msrp_per_year.index, avg_msrp_per_year.values, marker='o')
axs[2, 0].set_title('Average Base MSRP by Model Year')
axs[2, 0].set_xlabel('Model Year')
axs[2, 0].set_ylabel('Average Base MSRP (USD)')
axs[2, 0].tick_params(axis='x', rotation=45)

# Remove empty subplots
fig.delaxes(axs[1, 1])
fig.delaxes(axs[2, 1])

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.95])  # Make room for the main title
plt.show()
```

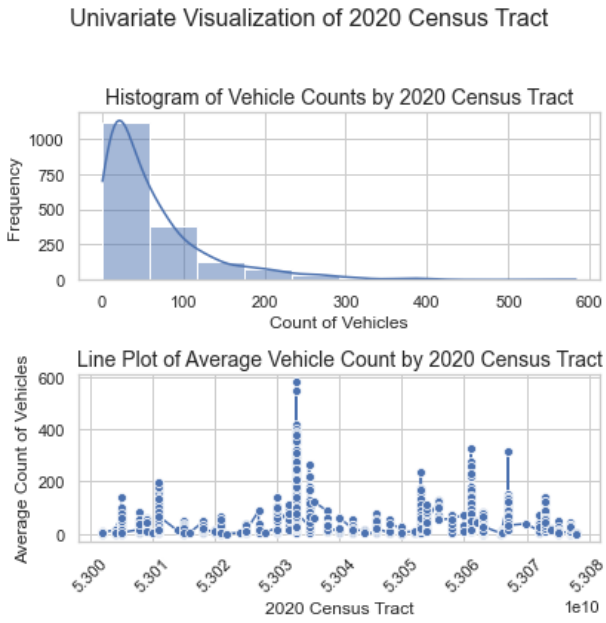

Univariate Visualization of Base MSRP
Histogram of Base MSRP
1e6
Count
Base MSRP (USD)
Box Plot of Base MSRP
Base MSRP (USD)
Count of Vehicles by Base MSRP Category
Count
0-20K
60K-80K
20K-40K
40K-60K
80K-100K
100K-120K
Base MSRP Category (USD)
Average Base MSRP by Model Year
Average Base MSRP (USD)
Model Year

In [29]:

```
# Set the visual style
sns.set(style="whitegrid")

# Create a figure for the histogram
plt.figure(figsize=(6, 6))
plt.title('Histogram of Vehicle Counts by Legislative District', fontsize=16)

# Plot the histogram of vehicle counts
sns.histplot(df['Legislative District'].value_counts(), bins=10, kde=True)

# Set labels
plt.xlabel('Count of Vehicles')
plt.ylabel('Frequency')

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



In [30]:
```
# Set the visual style
sns.set(style="whitegrid")

# Create a figure with 2 rows for subplots and size 6x6
fig, axs = plt.subplots(2, 1, figsize=(6, 6))
fig.suptitle('Univariate Visualization of 2020 Census Tract', fontsize=16)

# 1. Histogram for 2020 Census Tract
sns.histplot(df['2020 Census Tract'].value_counts(), bins=10, kde=True, ax=axs[0])
axs[0].set_title('Histogram of Vehicle Counts by 2020 Census Tract', fontsize=14)
axs[0].set_xlabel('Count of Vehicles')
axs[0].set_ylabel('Frequency')

# 2. Line Plot for 2020 Census Tract
avg_count_per_tract = df['2020 Census Tract'].value_counts().sort_index()
sns.lineplot(x=avg_count_per_tract.index, y=avg_count_per_tract.values, marker='o', ax=axs[1])
axs[1].set_title('Line Plot of Average Vehicle Count by 2020 Census Tract', fontsize=14)
axs[1].set_xlabel('2020 Census Tract')
axs[1].set_ylabel('Average Count of Vehicles')

# Rotate x-axis labels for better readability
axs[1].tick_params(axis='x', rotation=45)

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.95])  # Make room for the main title
plt.show()
```



In [31]:
```
df.columns
```

Out[31]:
```
Index(['VIN (1-10)', 'County', 'City', 'State', 'Postal Code', 'Model Year',
       'Make', 'Model', 'Electric Vehicle Type',
       'Clean Alternative Fuel Vehicle (CAFV) Eligibility', 'Electric Range',
       'Base MSRP', 'Legislative District', 'DOL Vehicle ID',
       'Vehicle Location', 'Electric Utility', '2020 Census Tract',
       'Range Category', 'MSRP Category'],
      dtype='object')
```

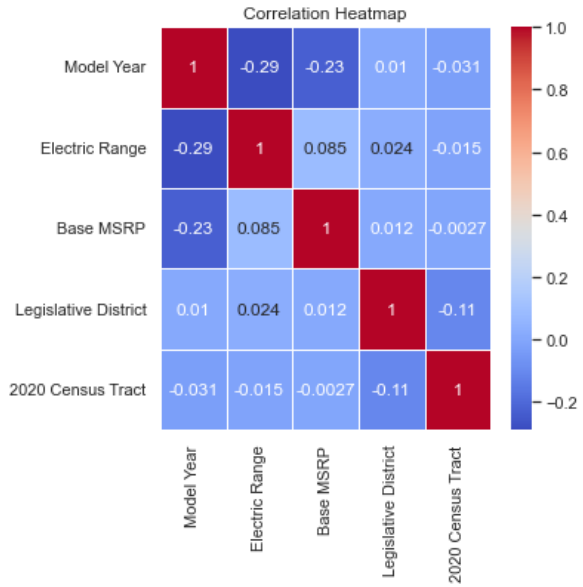
Type *Markdown* and LaTeX: α^2

In [32]:
```
num_vs_num = df[['Model Year', 'Electric Range', 'Base MSRP','Legislative District','Electric Utility','2020 Ce
```

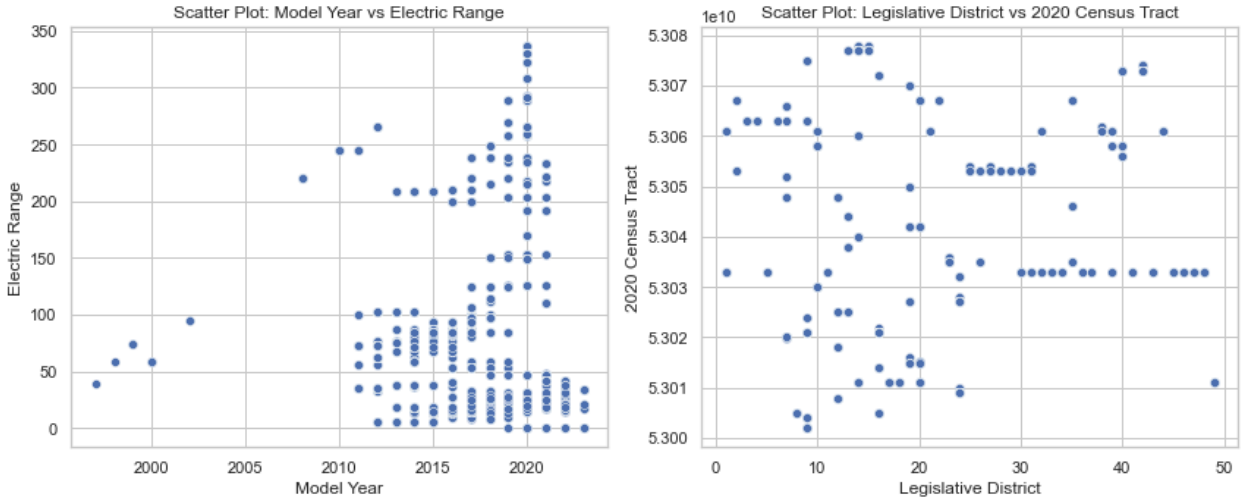
In [33]:
```
# Calculate the correlation matrix, but only for numeric columns
correlation_matrix = num_vs_num.select_dtypes(include=np.number).corr()

# Print the correlation matrix
print(correlation_matrix)
```

```
                    Model Year  Electric Range  Base MSRP  \
Model Year            1.000000       -0.288952  -0.229369
Electric Range       -0.288952        1.000000   0.085310
Base MSRP            -0.229369        0.085310   1.000000
Legislative District  0.010423        0.024383   0.012474
2020 Census Tract    -0.030528       -0.015352  -0.002684

                    Legislative District  2020 Census Tract
Model Year                      0.010423          -0.030528
Electric Range                  0.024383          -0.015352
Base MSRP                       0.012474          -0.002684
Legislative District            1.000000          -0.111296
2020 Census Tract              -0.111296           1.000000
```

In [34]:
```
# Calculate the correlation matrix, but only for numeric columns
correlation_matrix = num_vs_num.select_dtypes(include=np.number).corr()

# Create the heatmap
plt.figure(figsize=(5, 5))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

# Add a title for clarity
plt.title('Correlation Heatmap')

# Display the heatmap
plt.show()
```

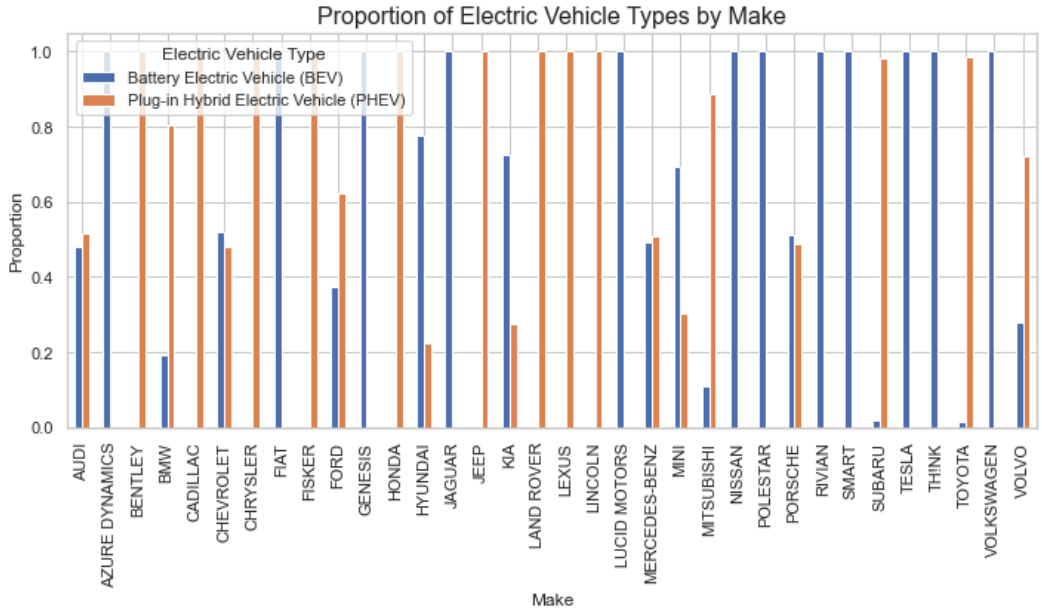


In [35]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# First scatter plot
sns.scatterplot(data=num_vs_num, x='Model Year', y='Electric Range', ax=axes[0])
axes[0].set_title('Scatter Plot: Model Year vs Electric Range')

# Second scatter plot
sns.scatterplot(data=num_vs_num, x='Legislative District', y='2020 Census Tract', ax=axes[1])
axes[1].set_title('Scatter Plot: Legislative District vs 2020 Census Tract')

# Display the plots
plt.tight_layout()
plt.show()
```



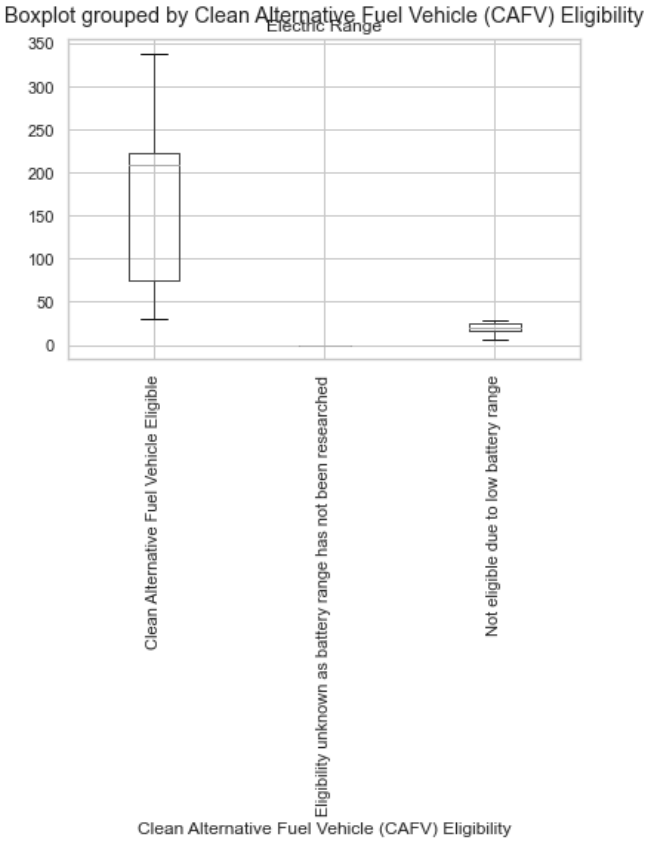
In [36]:
```
cat_cat=df[['County', 'City', 'State','Make', 'Model','Electric Vehicle Type','Clean Alternative Fuel Vehicle (
```

In [37]:
```
# Create a frequency table with normalization
tab = pd.crosstab(df['Make'], df['Electric Vehicle Type'], normalize='index')

# Create a vertical bar plot
tab.plot(kind='bar', figsize=(10, 6))

# Add titles and labels
plt.title('Proportion of Electric Vehicle Types by Make', fontsize=16)
plt.xlabel('Make', fontsize=12)
plt.ylabel('Proportion', fontsize=12)

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



In [38]:
```
df.boxplot(by="Clean Alternative Fuel Vehicle (CAFV) Eligibility", column=['Electric Range'])
# Rotate x-axis labels by 90 degrees
plt.xticks(rotation=90)
# Show the p
```

Out[38]:
```
(array([1, 2, 3]),
 [Text(1, 0, 'Clean Alternative Fuel Vehicle Eligible'),
  Text(2, 0, 'Eligibility unknown as battery range has not been researched'),
  Text(3, 0, 'Not eligible due to low battery range')])
```



In [ ]:

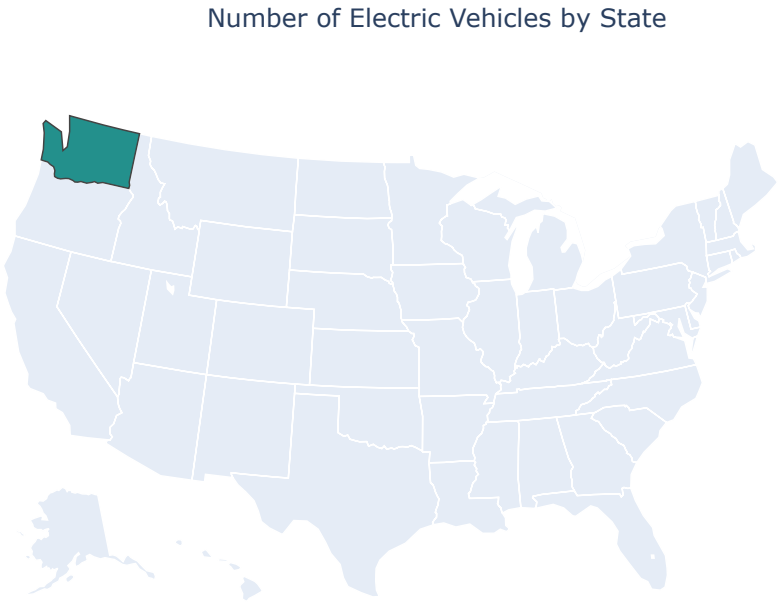
TASK_2

plotly.express to display the number of EV vehicles based on location

In [39]:
```
import plotly.express as px
```

In [40]:
```
ev_count_by_state = df.groupby('State').size().reset_index(name='Number_of_EV_Vehicles')
```

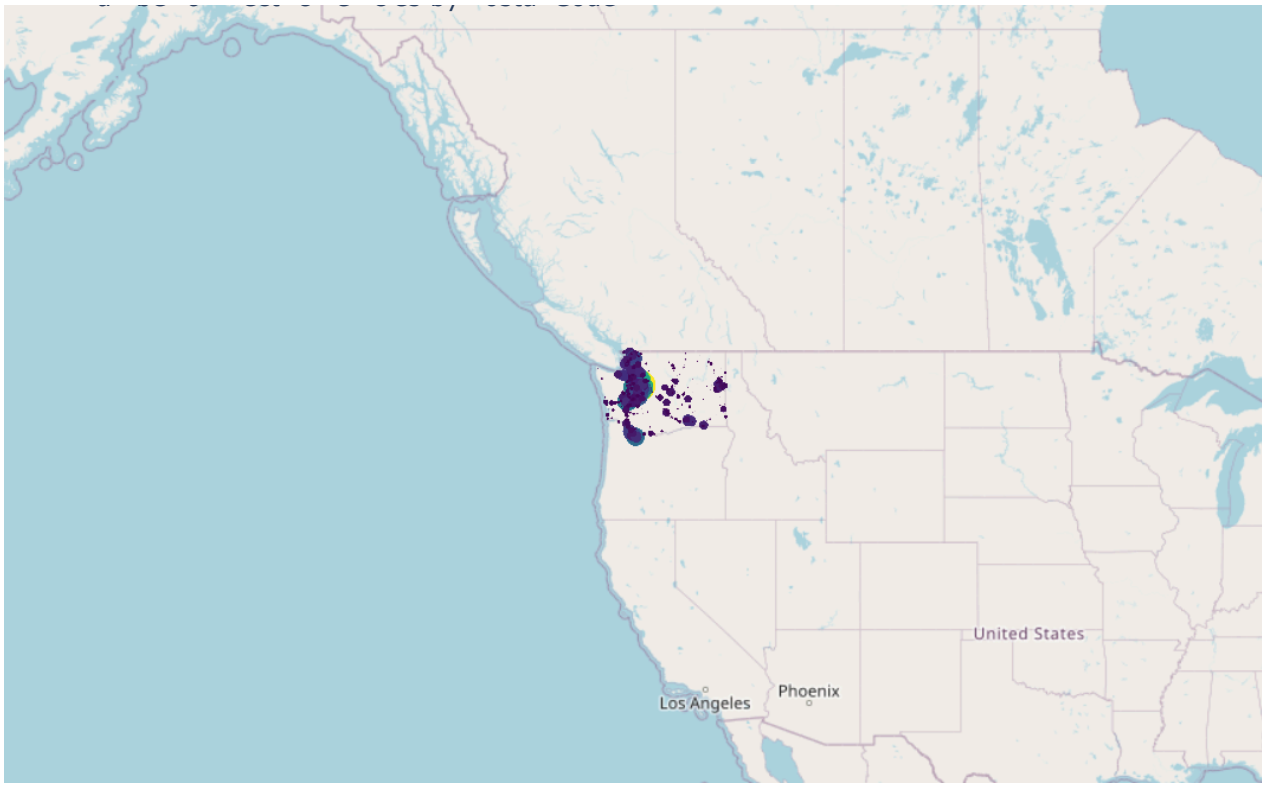
In [41]:
```
#count of EVs per state
ev_count_by_state = df['State'].value_counts().reset_index()
ev_count_by_state.columns = ['State', 'EV_Count']
# Create the Choropleth map
fig = px.choropleth(ev_count_by_state,
 locations='State',
 locationmode="USA-states",
 color='EV_Count',
 scope="usa",
 color_continuous_scale="Viridis",
 title="Number of Electric Vehicles by State")
# Update the layout
fig.update_layout(
 title_x=0.5,
 geo_scope='usa',
)
fig.show()
# Save the plot as an HTML file
fig.write_html("ev_choropleth_map.html")
print("Choropleth map has been created and saved as 'ev_choropleth_map.html'.")
print("\
Top 5 states by EV count:")
print(ev_count_by_state.head().to_string(index=False))
```




```
Choropleth map has been created and saved as 'ev_choropleth_map.html'.
Top 5 states by EV count:
State  EV_Count
   WA    112152
```

In [42]:
```
import pandas as pd
import plotly.express as px

# Count the number of EVs per postal code
ev_count_by_postal = df['Postal Code'].value_counts().reset_index()
ev_count_by_postal.columns = ['Postal Code', 'EV_Count']
# Merge the count with the original dataframe to get location data
df_merged = df.merge(ev_count_by_postal, on='Postal Code')
# Extract latitude and longitude from the 'Vehicle Location' column
df_merged['Longitude'] = df_merged['Vehicle Location'].str.extract('POINT \(([-\d.]+) ')
df_merged['Latitude'] = df_merged['Vehicle Location'].str.extract('([-\d.]+)\)')
# Convert to numeric
df_merged['Longitude'] = pd.to_numeric(df_merged['Longitude'])
df_merged['Latitude'] = pd.to_numeric(df_merged['Latitude'])
df_merged['Longitude'] = pd.to_numeric(df_merged['Longitude'])
df_merged['Latitude'] = pd.to_numeric(df_merged['Latitude'])
```

In [43]:
```
# Create the scatter plot on a map
fig = px.scatter_mapbox(df_merged,
 lat='Latitude',
 lon='Longitude',
 color='EV_Count',
 size='EV_Count',
 hover_name='Postal Code',
 hover_data=['City', 'State', 'EV_Count'],
 color_continuous_scale="Viridis",
 size_max=15,
 zoom=3,
 title="Number of Electric Vehicles by Postal Code")
fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
# Save the plot as an HTML file
fig.write_html("ev_postal_code_map.html")
fig.show()
print("Scatter map based on postal codes has been created and saved as 'ev_postal_code_map.html'")
print("\
Top 10 postal codes by EV count:")
print(ev_count_by_postal.head(10).to_string(index=False))
# Display some statistics
print("\
Total number of unique postal codes:", len(ev_count_by_postal))
print("Average number of EVs per postal code:", round(ev_count_by_postal['EV_Count'].mean()))
print("Median number of EVs per postal code:", ev_count_by_postal['EV_Count'].median())
print("Maximum number of EVs in a single postal code:", ev_count_by_postal['EV_Count'].max())
```




```
Scatter map based on postal codes has been created and saved as 'ev_postal_code_map.html'
Top 10 postal codes by EV count:
 Postal Code  EV_Count
       98052      2914
       98033      2059
       98004      2001
       98115      1878
       98006      1851
       98012      1850
       98072      1661
       98040      1639
       98074      1594
       98034      1578
Total number of unique postal codes: 516
Average number of EVs per postal code: 217
Median number of EVs per postal code: 49.0
Maximum number of EVs in a single postal code: 2914
```

TASK_3

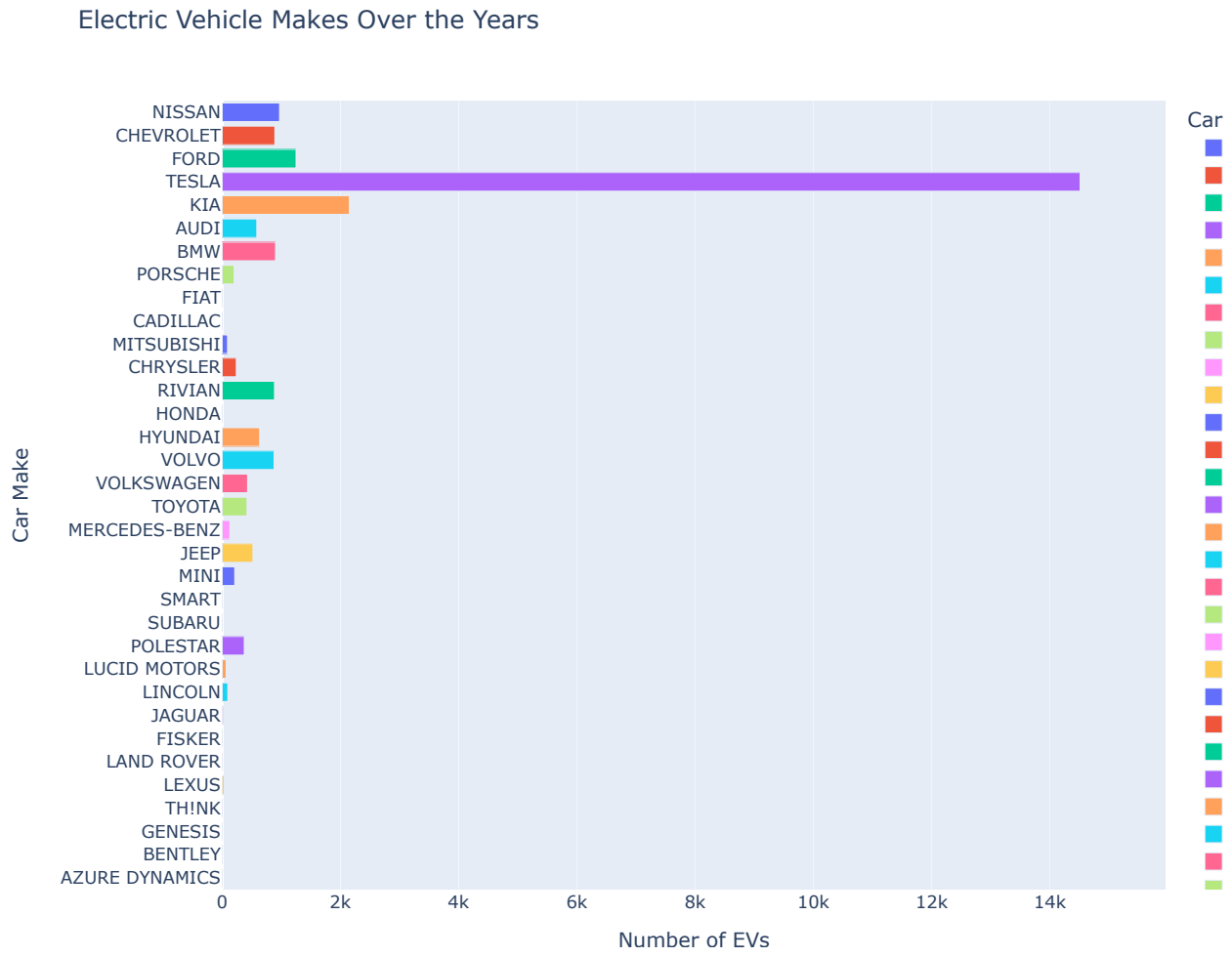
#Racing Bar plot

In [44]:
```
!pip install bar-chart-race
```

```
Collecting bar-chart-race
  Downloading bar_chart_race-0.1.0-py3-none-any.whl (156 kB)
Requirement already satisfied: matplotlib>=3.1 in c:\users\vijay\anaconda3\lib\site-packages (from bar-chart-
race) (3.5.1)
Requirement already satisfied: pandas>=0.24 in c:\users\vijay\anaconda3\lib\site-packages (from bar-chart-rac
e) (1.4.2)
Requirement already satisfied: packaging>=20.0 in c:\users\vijay\anaconda3\lib\site-packages (from matplotlib
>=3.1->bar-chart-race) (21.3)
Requirement already satisfied: cycler>=0.10 in c:\users\vijay\anaconda3\lib\site-packages (from matplotlib>=
3.1->bar-chart-race) (0.11.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\vijay\anaconda3\lib\site-packages (from matplotli
b>=3.1->bar-chart-race) (3.0.4)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\vijay\anaconda3\lib\site-packages (from matplotl
ib>=3.1->bar-chart-race) (1.3.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\vijay\anaconda3\lib\site-packages (from matpl
otlib>=3.1->bar-chart-race) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in c:\users\vijay\anaconda3\lib\site-packages (from matplotlib>=
3.1->bar-chart-race) (9.0.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\vijay\anaconda3\lib\site-packages (from matplotl
ib>=3.1->bar-chart-race) (4.25.0)
Requirement already satisfied: numpy>=1.17 in c:\users\vijay\anaconda3\lib\site-packages (from matplotlib>=3.
1->bar-chart-race) (1.21.5)
Requirement already satisfied: pytz>=2020.1 in c:\users\vijay\anaconda3\lib\site-packages (from pandas>=0.24-
>bar-chart-race) (2021.3)
Requirement already satisfied: six>=1.5 in c:\users\vijay\anaconda3\lib\site-packages (from python-dateutil>=
2.7->matplotlib>=3.1->bar-chart-race) (1.16.0)
Installing collected packages: bar-chart-race
Successfully installed bar-chart-race-0.1.0
```

In [45]:
```
import bar_chart_race as bcr
import warnings
```

In [46]:
```
ev_make_by_year = df.groupby(['Model Year', 'Make']).size().reset_index(name='EV Count')

# Step 2: Create a list of all unique makes
unique_makes = df['Make'].unique()

# Step 3: Ensure all makes appear in every year by filling missing combinations
all_years = pd.DataFrame({'Model Year': sorted(df['Model Year'].unique())})
all_combinations = all_years.assign(key=1).merge(pd.DataFrame({'Make': unique_makes, 'key':1}), on='key').drop(
ev_make_by_year_full = all_combinations.merge(ev_make_by_year, on=['Model Year', 'Make'], how='left').fillna(0)

# Step 4: Convert EV Count to integer (since it was NaN before)
ev_make_by_year_full['EV Count'] = ev_make_by_year_full['EV Count'].astype(int)

# Step 5: Create the animated racing bar plot with increased height
fig = px.bar(
    ev_make_by_year_full,  # Data
    x='EV Count',  # X-axis shows the count of EVs
    y='Make',  # Y-axis shows the car Make
    color='Make',  # Color by car Make
    animation_frame='Model Year',  # Animation by year
    orientation='h',  # Horizontal bar chart
    title='Electric Vehicle Makes Over the Years',
    labels={'EV Count':'Number of EVs', 'Make':'Car Make'},  # Axis labels
    range_x=[0, ev_make_by_year_full['EV Count'].max() * 1.1],  # Dynamically set x-axis range
    height=800  # Increased height for better visibility
)

# Step 6: Show the plot
fig.show()
```



In [ ]: