FDS_PROJ

December 1, 2024

[2]:
```
import pandas as pd
import numpy as np
from prophet import Prophet
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import FuncFormatter
from prophet.diagnostics import cross_validation, performance_metrics
```

Loading data and Initial Exploration

[3]:
```
df= pd.read_csv(r"C:\Users\Abhilove Goyal\Downloads\IEA Global EV Data 2024.
 ↪csv")
df.head()
```

[3]:

	region	category	parameter	mode	powertrain	year	unit	value
0	Australia	Historical	EV stock share	Cars	EV	2011	percent	0.00039
1	Australia	Historical	EV sales share	Cars	EV	2011	percent	0.00650
2	Australia	Historical	EV sales	Cars	BEV	2011	Vehicles	49.00000
3	Australia	Historical	EV stock	Cars	BEV	2011	Vehicles	49.00000
4	Australia	Historical	EV stock	Cars	BEV	2012	Vehicles	220.00000

[4]:
```
df_shape=df.shape
print(df_shape)
```

```
(12654, 8)
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12654 entries, 0 to 12653
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
```

```
---  ------      --------------  -----
 0   region      12654 non-null  object
 1   category    12654 non-null  object
 2   parameter   12654 non-null  object
 3   mode        12654 non-null  object
 4   powertrain  12654 non-null  object
 5   year        12654 non-null  int64
 6   unit        12654 non-null  object
 7   value       12654 non-null  float64
dtypes: float64(1), int64(1), object(6)
memory usage: 791.0+ KB
```

[8]:
```
# display(df.head)
display(df.region.unique())
```

```
array(['Australia', 'Austria', 'Belgium', 'Brazil', 'Bulgaria', 'Canada',
       'Chile', 'China', 'Colombia', 'Costa Rica', 'Croatia', 'Cyprus',
       'Czech Republic', 'Denmark', 'Estonia', 'EU27', 'Europe',
       'Finland', 'France', 'Germany', 'Greece', 'Hungary', 'Iceland',
       'India', 'Indonesia', 'Ireland', 'Israel', 'Italy', 'Japan',
       'Korea', 'Latvia', 'Lithuania', 'Luxembourg', 'Mexico',
       'Netherlands', 'New Zealand', 'Norway', 'Poland', 'Portugal',
       'Rest of the world', 'Romania', 'Seychelles', 'Slovakia',
       'Slovenia', 'South Africa', 'Spain', 'Sweden', 'Switzerland',
       'Thailand', 'Turkiye', 'United Arab Emirates', 'United Kingdom',
       'USA', 'World'], dtype=object)
```

[6]:
```
region_counts= df['region'].value_counts().head()
print(region_counts) # it is not the sales of each country it is the number of␣
  ↪columns they have in 12K set
```

```
region
World                1250
Europe               1234
China                1138
Rest of the world     954
USA                   737
Name: count, dtype: int64
```

[7]:
```
display(df.year.unique())
```

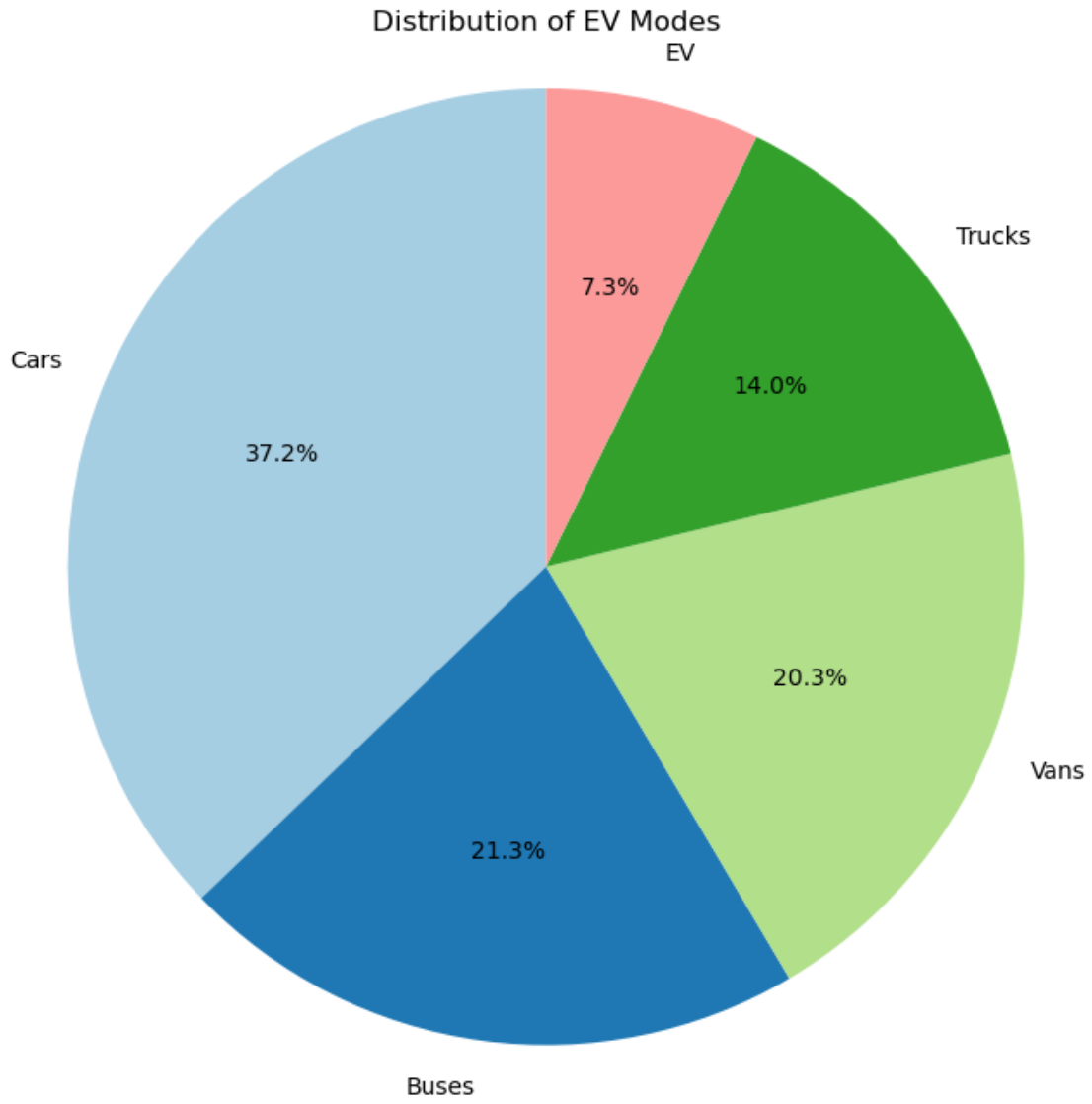
```
array([2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021,
       2022, 2023, 2010, 2025, 2030, 2035], dtype=int64)
```

Analyzing EV data

[10]:
```
mode=df['mode'].value_counts()
display(mode)
# Data for the pie chart
mode_counts = df['mode'].value_counts()
```

```
# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(mode_counts, labels=mode_counts.index, autopct='%1.1f%%',␣
 ↪startangle=90, colors=plt.cm.Paired.colors)
plt.title('Distribution of EV Modes')
plt.axis('equal')  # Equal aspect ratio ensures the pie is drawn as a circle.
plt.show()
```

```
mode
Cars      4706
Buses     2696
Vans      2568
Trucks    1766
EV         918
Name: count, dtype: int64
```

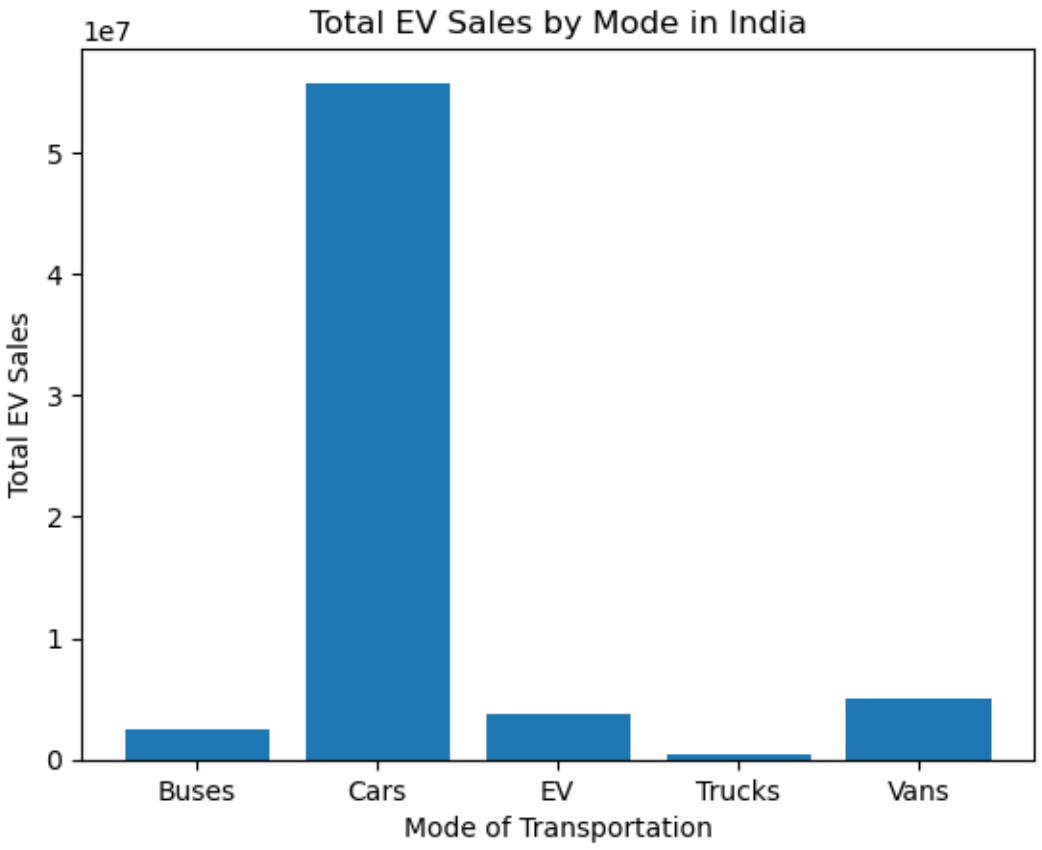


[10]:
```
ev_sales_data = df[(df['parameter'] == 'EV sales') & (df['year']<2025)]

ev_sales_by_country = ev_sales_data.groupby('region')['value'].sum().
 ↪reset_index()

ev_sales_by_country.columns = ['Country', 'Total EV Sales']
sorted_ev_sales = ev_sales_by_country.sort_values(by='Total EV Sales',␣
 ↪ascending=False)

print(sorted_ev_sales.head())
# create matplot for this
```

```
  Country  Total EV Sales
```

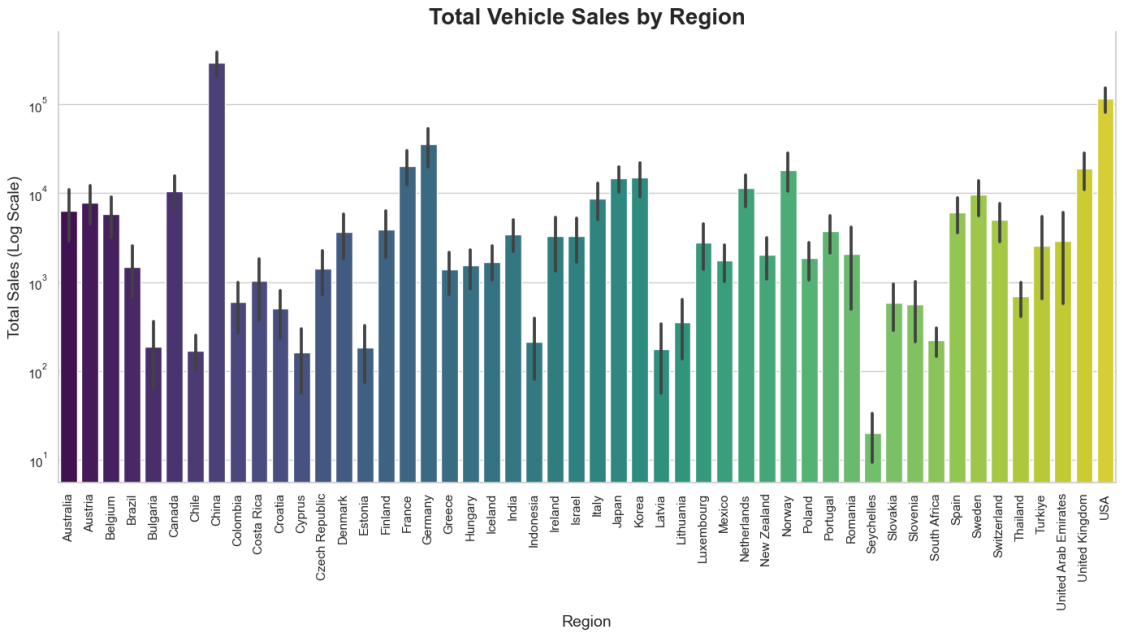
```
51    World     113426649.0
7     China      61755388.0
16   Europe      32277364.0
48      USA      11408828.2
14     EU27       8944714.0
```

[11]:
```
# Group by mode and sum the values
mode_data = df[df['region'] == "India"]
mode_grouped = mode_data.groupby('mode')['value'].sum()

# Plotting
plt.bar(mode_grouped.index, mode_grouped.values)  # Use mode for X and total EV␣
 ↪sales for Y
plt.xlabel('Mode of Transportation')
plt.ylabel('Total EV Sales')
plt.title('Total EV Sales by Mode in India')
plt.show()
```

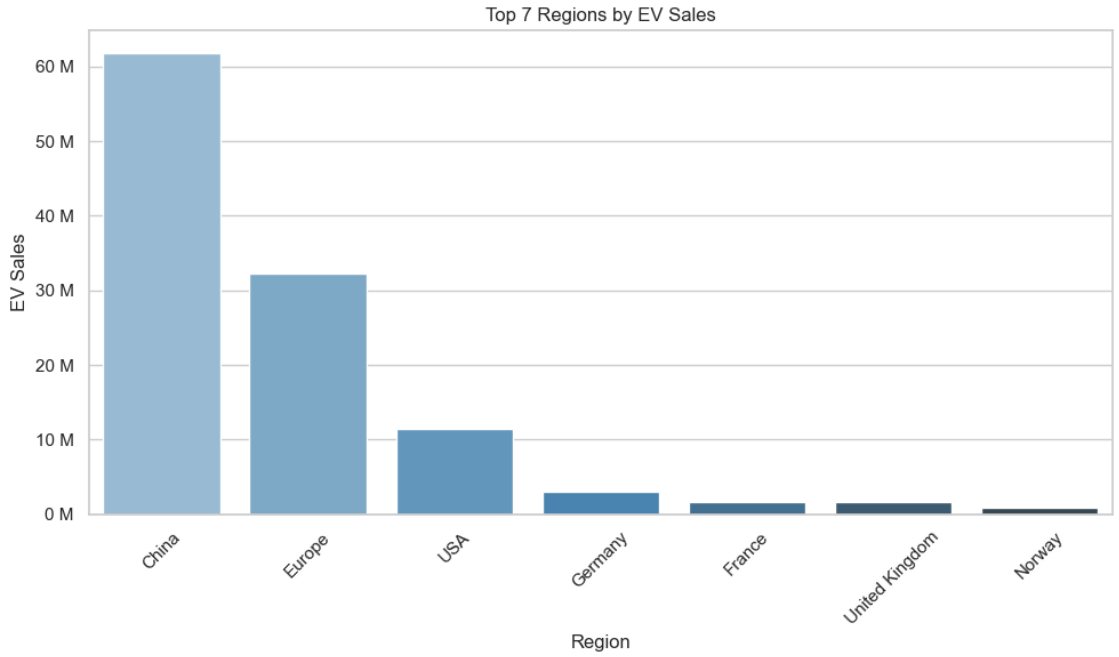


[12]:
```
# Set the style to 'whitegrid'
sns.set(style="whitegrid")
```

```
# Create the figure and axis
plt.figure(figsize=(14, 8))

# Filter the data
filter_data = df[(~df['region'].isin(['World', 'EU27', 'Europe','Rest of the␣
 ↪world'])) & (df['year']<2025)]

# Plot the bar chart
sns.barplot(data=filter_data, x='region', y='value', palette="viridis")

# Customize the plot
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.title('Total Vehicle Sales by Region', fontsize=20, fontweight='bold')
plt.xlabel('Region', fontsize=14)
plt.ylabel('Total Sales (Log Scale)', fontsize=14)

# Use logarithmic scale for the y-axis if values span a large range
plt.yscale('log')

# Remove the top border line (spine)
plt.gca().spines['top'].set_visible(False)

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

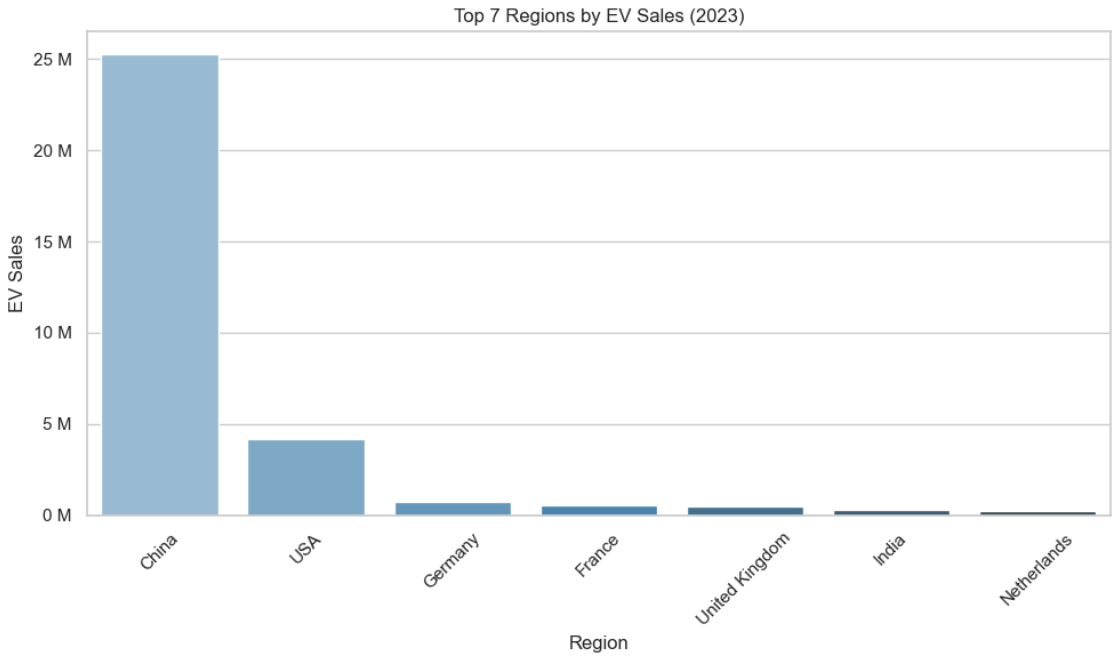


[13]:
```
# Filter the data
ev_sales_data = df[(df['parameter'] == 'EV sales') &
                   (~df['region'].isin(['World', 'EU27','Rest of the world'])) &
                   (df['year'] < 2025)]

# Group by region and sum the sales for each region
ev_sales_by_region = ev_sales_data.groupby('region')['value'].sum()

# Sort by sales in descending order
ev_sales_by_region_sorted = ev_sales_by_region.sort_values(ascending=False)

# Select the top n regions (for example, top 7)
top_n = 7
top_countries = ev_sales_by_region_sorted.head(top_n).reset_index()

# Create the plot using Seaborn
plt.figure(figsize=(10, 6))
sns.barplot(x='region', y='value', data=top_countries, palette='Blues_d')

# Format the y-axis to remove scientific notation
formatter = FuncFormatter(lambda x, _: f'{int(x / 1e6):,} M')  # Format as␣
 ↪millions with "M"
plt.gca().yaxis.set_major_formatter(formatter)

# Customize the plot
plt.xlabel('Region')
plt.ylabel('EV Sales')
plt.title(f'Top {top_n} Regions by EV Sales')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels

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



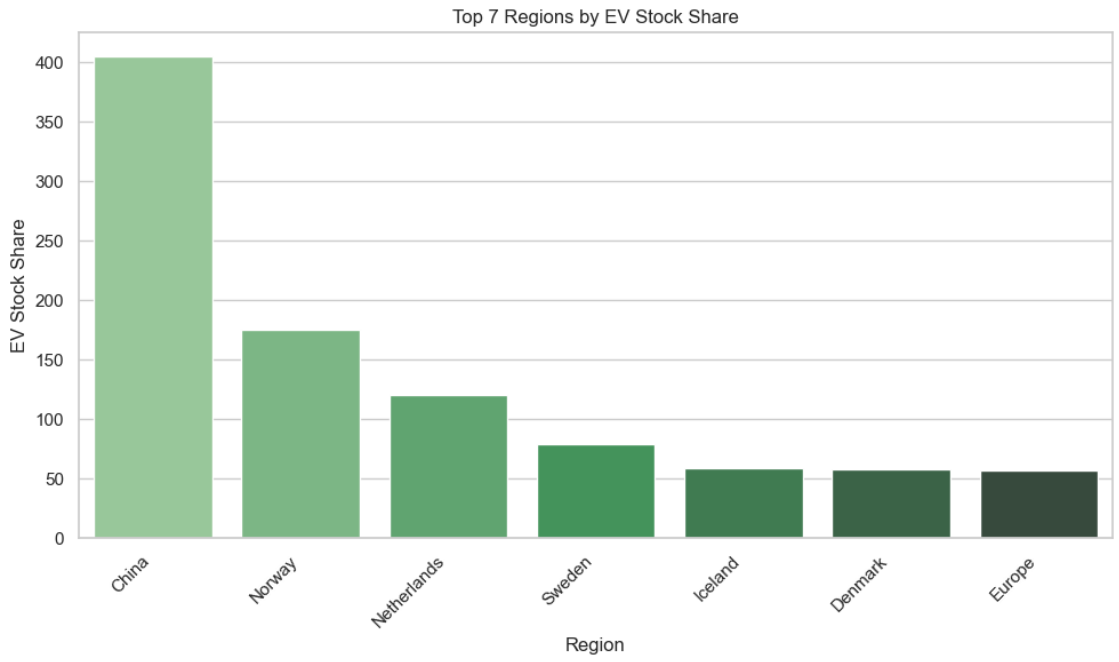
[14]:
```
# Filter the data
ev_sales_data = df[(df['parameter'] == 'EV sales') &
                   (~df['region'].isin(['World', 'EU27', 'Europe','Rest of the␣
 ↪world'])) &
                   (df['year'] == 2023)]

# Group by region and sum the sales for each region
ev_sales_by_region = ev_sales_data.groupby('region')['value'].sum()

# Sort by sales in descending order
ev_sales_by_region_sorted = ev_sales_by_region.sort_values(ascending=False)

# Select the top n regions (for example, top 7)
top_n = 7
top_countries = ev_sales_by_region_sorted.head(top_n).reset_index()

# Create the plot using Seaborn
plt.figure(figsize=(10, 6))
sns.barplot(x='region', y='value', data=top_countries, palette='Blues_d')

# Format the y-axis to remove scientific notation
formatter = FuncFormatter(lambda x, _: f'{int(x / 1e6):,} M')  # Format as␣
 ↪millions with "M"
plt.gca().yaxis.set_major_formatter(formatter)
```

```
# Customize the plot
plt.xlabel('Region')
plt.ylabel('EV Sales')
plt.title(f'Top {top_n} Regions by EV Sales (2023)')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels

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



[15]:
```
ev_stock = df[(df['parameter'] == 'EV stock share') & (df['region'] != 'World')␣
↪& (df['year']<2024)]
ev_stock_by_country = ev_stock.groupby('region')['value'].sum()
ev_stock_sorted = ev_stock_by_country.sort_values(ascending=False)
top_n = 7
ev_stock_top = ev_stock_sorted.head(top_n).reset_index()

# Plot using Seaborn
plt.figure(figsize=(10, 6))  # Set figure size
sns.barplot(x='region', y='value', data=ev_stock_top, palette='Greens_d')

# Customize the plot
plt.title("Top 7 Regions by EV Stock Share")
plt.xlabel("Region")
plt.ylabel("EV Stock Share")
```

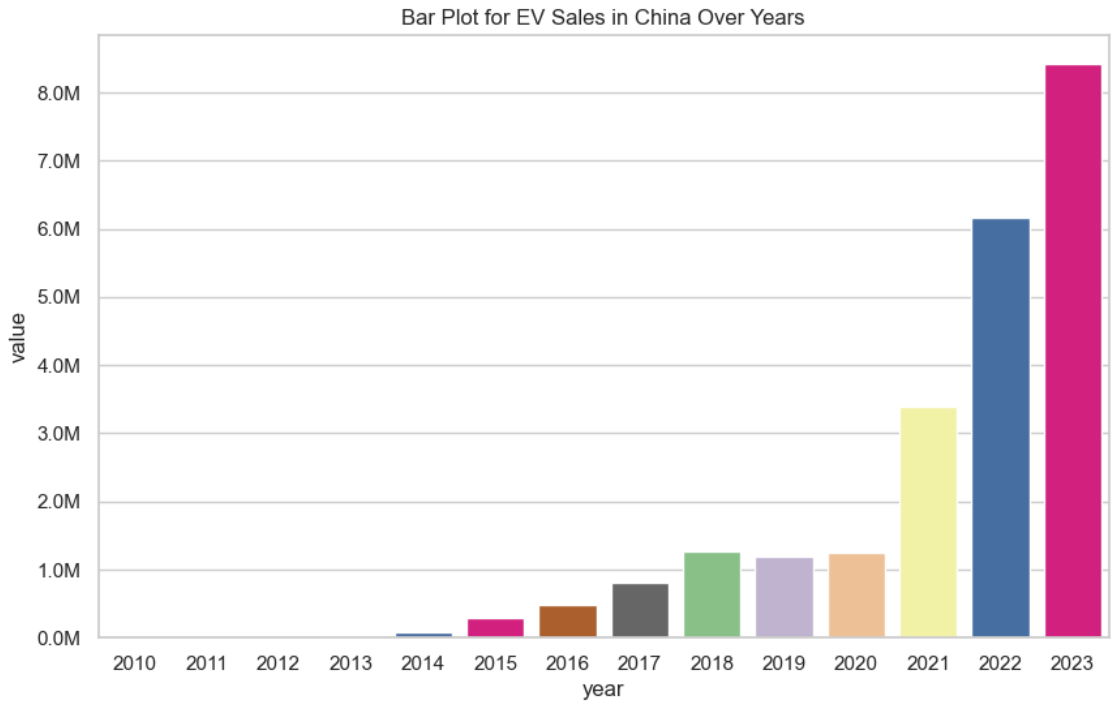
```
plt.xticks(rotation=45, ha='right')

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



[ ]:

China Sales distribution

[16]:
```
china_sales=df[(df['region']=='China') & (df['parameter']=='EV sales') &␣
 ↪(df['category']=='Historical')]
china_filter= china_sales[china_sales['year']<2024];
china_sale_grp=china_filter.groupby('year')['value'].sum().reset_index()
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'
plt.figure(figsize=(10,6))
sns.barplot(x='year',y='value',data=china_sale_grp,palette='Accent')
plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))
plt.xlabel=('Year')
plt.ylabel=('Values')
plt.title('Bar Plot for EV Sales in China Over Years')
plt.show()
```

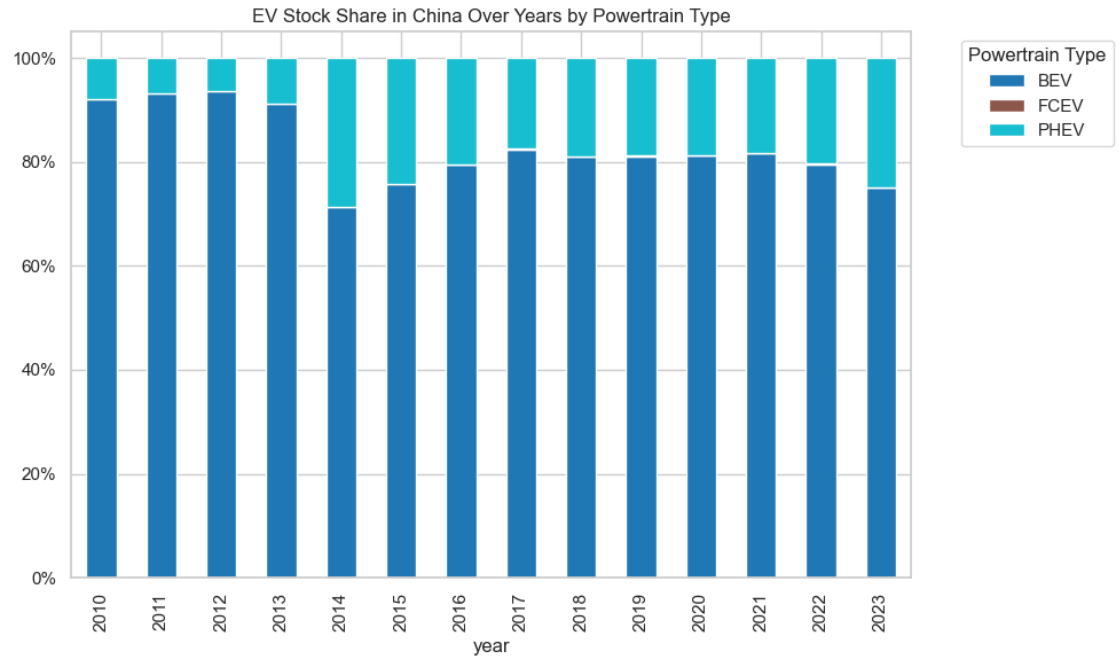


Having a better look->

[19]:
```
# Filter for China, EV stock, and historical data
china_stock = df[(df['region'] == 'China') & (df['parameter'] == 'EV stock') &␣
 ↪(df['category'] == 'Historical')]
china_stock_filter = china_stock[china_stock['year'] < 2024]

# Group by year and powertrain, then sum the stock values
china_stock_grp = china_stock_filter.groupby(['year', 'powertrain'])['value'].
 ↪sum().reset_index()

# Pivot to get total stock values by year and powertrain
china_stock_pivot = china_stock_grp.pivot(index='year', columns='powertrain',␣
 ↪values='value').fillna(0)

# Calculate the share of each powertrain by dividing by the total stock for␣
 ↪each year
china_stock_share = china_stock_pivot.div(china_stock_pivot.sum(axis=1),␣
 ↪axis=0) * 100

# Function to format y-axis as percentage
def percentage(x, pos):
    return f'{x:.0f}%'
```

```
# Plotting
plt.figure(figsize=(10, 6))

# Stacked barplot for EV stock share by year and powertrain type
china_stock_share.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in percentage
plt.gca().yaxis.set_major_formatter(FuncFormatter(percentage))

# Setting labels and title
plt.xlabel='Year'
plt.ylabel='Share (%)'
plt.title('EV Stock Share in China Over Years by Powertrain Type')

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

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

```
<Figure size 1000x600 with 0 Axes>
```



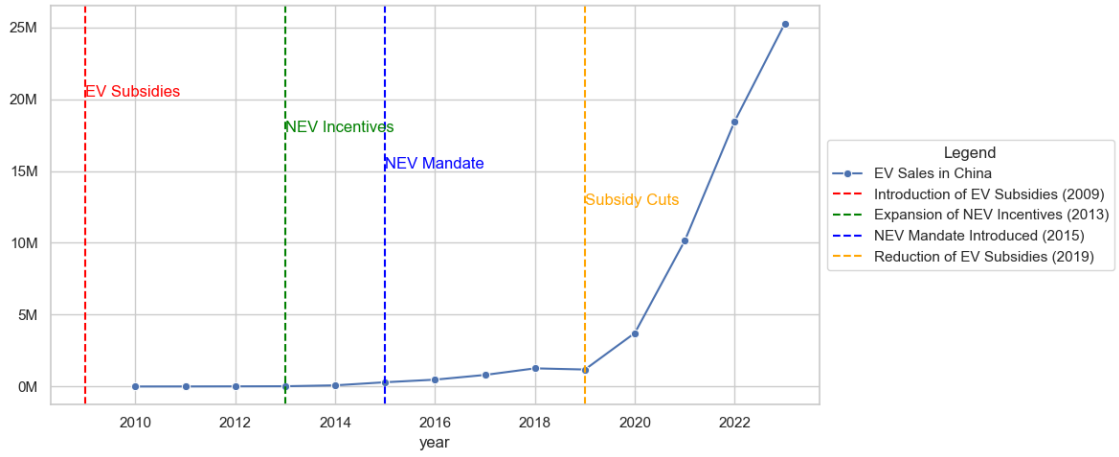
Now in here i feteched data about various govt policies regarding EV incentives and subsidies So i have made a plot depicting sales growth along with the scheme launch year

CHINA SALES v/s GOVT POLICY

[163]:
```
# Set the Seaborn style
sns.set(style="whitegrid")

# Filter data for China with EV sales and year < 2024
china_sales = df[(df['region'] == 'China') & (df['parameter'] == 'EV sales') &␣
 ↪(df['year'] < 2024)]

# Group by year and sum the sales values
china_sales_group = china_sales.groupby('year')['value'].sum()

# Plot the EV sales time-series
plt.figure(figsize=(14, 5))
sns.lineplot(x=china_sales_group.index, y=china_sales_group.values, marker='o',␣
 ↪label='EV Sales in China')

# Adding vertical lines to indicate policy events
plt.axvline(x=2009, color='red', linestyle='--', label='Introduction of EV␣
 ↪Subsidies (2009)')
plt.axvline(x=2013, color='green', linestyle='--', label='Expansion of NEV␣
 ↪Incentives (2013)')
plt.axvline(x=2015, color='blue', linestyle='--', label='NEV Mandate Introduced␣
 ↪(2015)')
plt.axvline(x=2019, color='orange', linestyle='--', label='Reduction of EV␣
 ↪Subsidies (2019)')

def millions(x, pos):
    return f'{int(x / 1e6)}M'  # Convert to millions

formatter = FuncFormatter(millions)
plt.gca().yaxis.set_major_formatter(formatter)


# Adding annotations for policies
plt.text(2009, max(china_sales_group)*0.8, 'EV Subsidies', color='red')
plt.text(2013, max(china_sales_group)*0.7, 'NEV Incentives', color='green')
plt.text(2015, max(china_sales_group)*0.6, 'NEV Mandate', color='blue')
plt.text(2019, max(china_sales_group)*0.5, 'Subsidy Cuts', color='orange')

# Setting labels and title
plt.xlabel=('Year')
plt.ylabel=('EV Sales (Vehicles)')
plt.title=('Impact of Government Policies on EV Sales in China')

# Show legend
plt.legend(loc='center left',bbox_to_anchor=(1,0.5),title="Legend")
```

```
# Show plot
plt.tight_layout(rect=[0,0,0.85,1])
plt.show()
```

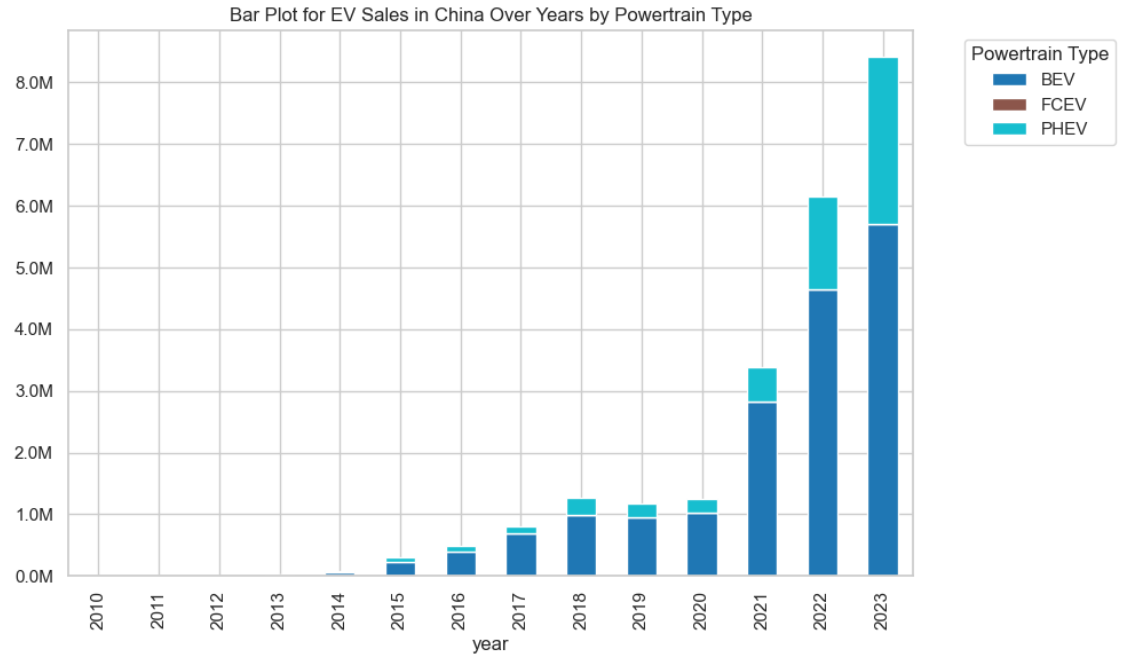
```
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[17]:
```
#IMPROVED GRAPH
# Assuming 'df' is already defined in your code
china_sales = df[(df['region'] == 'China') & (df['parameter'] == 'EV sales') &␣
 ↪(df['category'] == 'Historical')]
china_filter = china_sales[china_sales['year'] < 2024]

# Group by year and powertrain, then sum the sales values
china_sale_grp = china_filter.groupby(['year', 'powertrain'])['value'].sum().
 ↪reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))
```

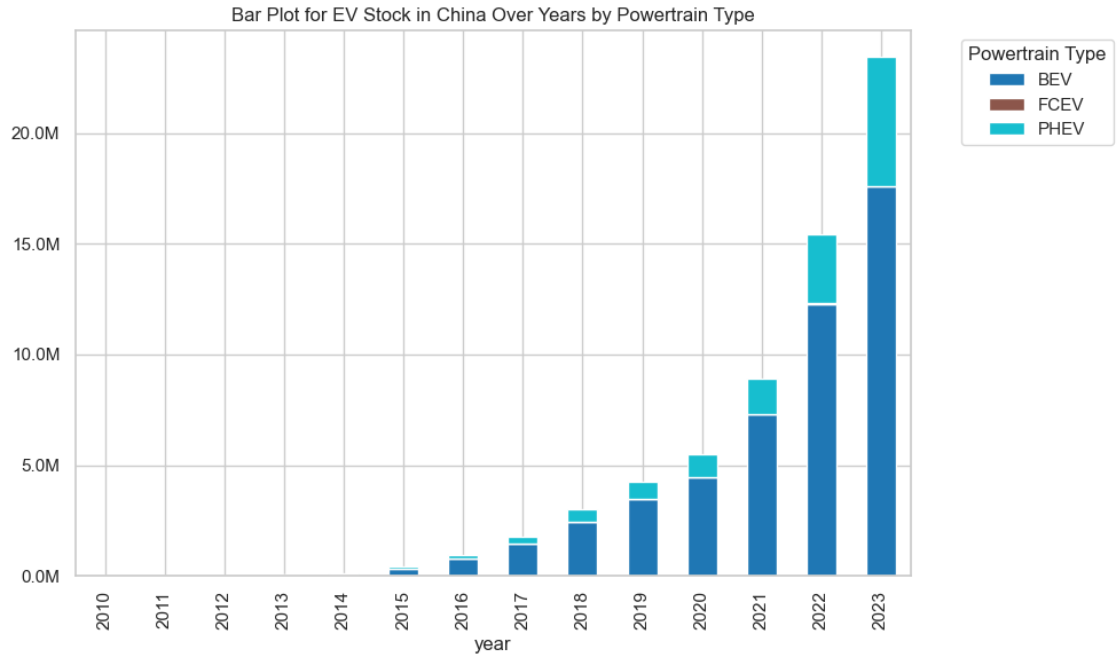
```
# Stacked barplot by year and powertrain type
china_sale_pivot = china_sale_grp.pivot(index='year', columns='powertrain',␣
 ↪values='value').fillna(0)

# Plot each powertrain type as a stacked bar
china_sale_pivot.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in millions
plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))

# Setting labels and title
plt.xlabel=('Year')
plt.ylabel=('Values')
plt.title('Bar Plot for EV Sales in China Over Years by Powertrain Type')

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



[18]:
```
# Filter for China, EV stock, and historical data
china_stock = df[(df['region'] == 'China') & (df['parameter'] == 'EV stock') &␣
 ↪(df['category'] == 'Historical')]
china_stock_filter = china_stock[china_stock['year'] < 2024]

# Group by year and powertrain, then sum the stock values
china_stock_grp = china_stock_filter.groupby(['year', 'powertrain'])['value'].
 ↪sum().reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))

# Stacked barplot by year and powertrain type for EV stock
china_stock_pivot = china_stock_grp.pivot(index='year', columns='powertrain',␣
 ↪values='value').fillna(0)

# Plot each powertrain type as a stacked bar
china_stock_pivot.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in millions
plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))

# Setting labels and title
plt.xlabel='Year'
plt.ylabel='Values (Millions)'
plt.title('Bar Plot for EV Stock in China Over Years by Powertrain Type')

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

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

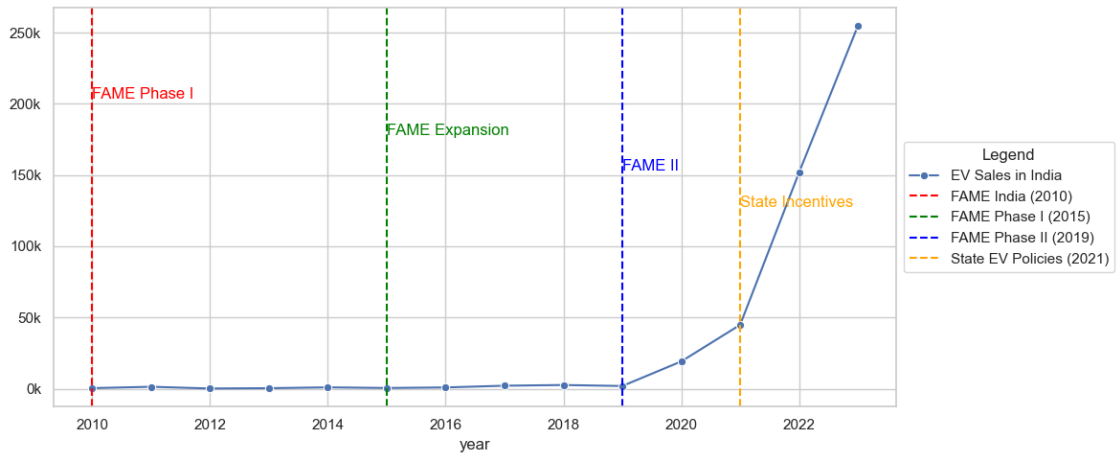
```
<Figure size 1000x600 with 0 Axes>
```



[170]:
```
# Set the Seaborn style
sns.set(style="whitegrid")

# Filter data for India with EV sales and year < 2024
india_sales = df[(df['region'] == 'India') & (df['parameter'] == 'EV sales') &␣
 ↪(df['year'] < 2024)]

# Group by year and sum the sales values
india_sales_group = india_sales.groupby('year')['value'].sum()

# Plot the EV sales time-series
plt.figure(figsize=(14, 5))
sns.lineplot(x=india_sales_group.index, y=india_sales_group.values, marker='o',␣
 ↪label='EV Sales in India')

# Adding vertical lines to indicate policy events
plt.axvline(x=2010, color='red', linestyle='--', label='FAME India (2010)')
plt.axvline(x=2015, color='green', linestyle='--', label='FAME Phase I (2015)')
plt.axvline(x=2019, color='blue', linestyle='--', label='FAME Phase II (2019)')
plt.axvline(x=2021, color='orange', linestyle='--', label='State EV Policies␣
 ↪(2021)')

def thousand(x, pos):
    return f'{int(x / 1e3)}k'  # Convert to millions
```

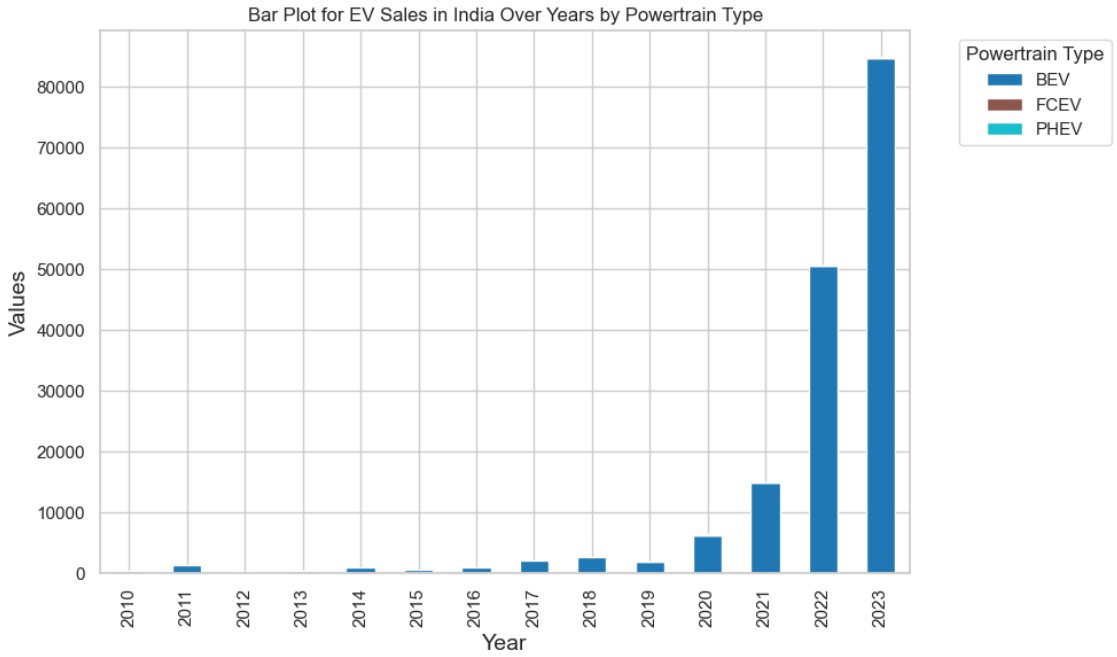
```
formatter = FuncFormatter(thousand)
plt.gca().yaxis.set_major_formatter(formatter)


# Adding annotations for policies
plt.text(2010, max(india_sales_group)*0.8, 'FAME Phase I', color='red')
plt.text(2015, max(india_sales_group)*0.7, 'FAME Expansion', color='green')
plt.text(2019, max(india_sales_group)*0.6, 'FAME II', color='blue')
plt.text(2021, max(india_sales_group)*0.5, 'State Incentives', color='orange')

# Setting labels and title
plt.xlabel=('Year')
plt.ylabel=('EV Sales (Vehicles)')
plt.title=('Impact of Government Policies on EV Sales in India')

# Show legend
plt.legend(loc='center left',bbox_to_anchor=(1,0.5),title="Legend")

# Show plot
plt.tight_layout(rect=[0,0,0.85,1])
plt.show()
```

```
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[13]:
```
from matplotlib.ticker import FuncFormatter
sns.set(style="whitegrid")
# Assuming 'df' is already defined in your code

# Filter data for India
india_sales = df[(df['region'] == 'India') & (df['parameter'] == 'EV sales') &␣
 ↪(df['category'] == 'Historical')]
india_filter = india_sales[india_sales['year'] < 2024]

# Group by year and powertrain, then sum the sales values
india_sale_grp = india_filter.groupby(['year', 'powertrain'])['value'].sum().
 ↪reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))

# Stacked barplot by year and powertrain type for India
india_sale_pivot = india_sale_grp.pivot(index='year', columns='powertrain',␣
 ↪values='value').fillna(0)

# Plot each powertrain type as a stacked bar
india_sale_pivot.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in millions
# plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))

# Setting labels and title
plt.title('Bar Plot for EV Sales in India Over Years by Powertrain Type')
plt.xlabel('Year', fontsize=14)
plt.ylabel('Values', fontsize=14)

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.show()
```

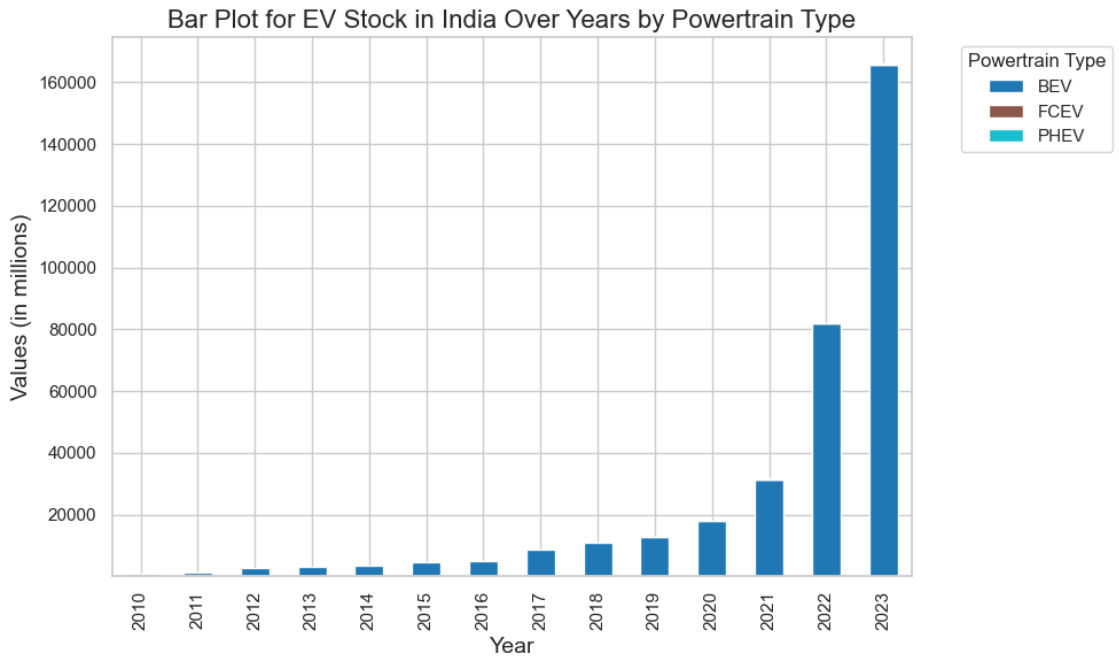
```
<Figure size 1000x600 with 0 Axes>
```



[12]:
```
from matplotlib.ticker import FuncFormatter, MaxNLocator

# Assuming 'df' is already defined in your code
sns.set(style="whitegrid")
# Filter data for India (EV stock)
india_stock = df[(df['region'] == 'India') & (df['parameter'] == 'EV stock') &␣
 ↪(df['category'] == 'Historical')]
india_stock_filter = india_stock[india_stock['year'] < 2024]

# Group by year and powertrain, then sum the stock values
india_stock_grp = india_stock_filter.groupby(['year', 'powertrain'])['value'].
 ↪sum().reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))

# Stacked barplot by year and powertrain type for India (EV stock)
india_stock_pivot = india_stock_grp.pivot(index='year', columns='powertrain',␣
 ↪values='value').fillna(0)

# Plot each powertrain type as a stacked bar
```

```
india_stock_pivot.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in millions
# plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))

# Adjusting the y-axis tick interval and formatting
plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True, prune='lower'))  #␣
 ↪Ensure integer ticks and no lower bound

# Setting labels and title
plt.title('Bar Plot for EV Stock in India Over Years by Powertrain Type',␣
 ↪fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Values (in millions)', fontsize=14)

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

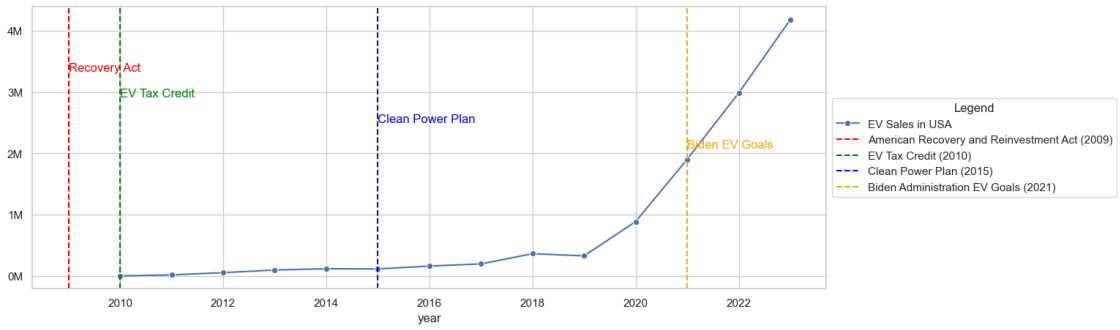


[ ]:
```
USA
```

[171]:
```
# Set the Seaborn style
sns.set(style="whitegrid")

# Filter data for USA with EV sales and year < 2024
usa_sales = df[(df['region'] == 'USA') & (df['parameter'] == 'EV sales') &␣
 ↪(df['year'] < 2024)]

# Group by year and sum the sales values
usa_sales_group = usa_sales.groupby('year')['value'].sum()

# Plot the EV sales time-series
plt.figure(figsize=(14, 5))
sns.lineplot(x=usa_sales_group.index, y=usa_sales_group.values, marker='o',␣
 ↪label='EV Sales in USA')

# Adding vertical lines to indicate policy events
plt.axvline(x=2009, color='red', linestyle='--', label='American Recovery and␣
 ↪Reinvestment Act (2009)')
plt.axvline(x=2010, color='green', linestyle='--', label='EV Tax Credit (2010)')
plt.axvline(x=2015, color='blue', linestyle='--', label='Clean Power Plan␣
 ↪(2015)')
plt.axvline(x=2021, color='orange', linestyle='--', label='Biden Administration␣
 ↪EV Goals (2021)')

# Adding annotations for policies
plt.text(2009, max(usa_sales_group)*0.8, 'Recovery Act', color='red')
plt.text(2010, max(usa_sales_group)*0.7, 'EV Tax Credit', color='green')
plt.text(2015, max(usa_sales_group)*0.6, 'Clean Power Plan', color='blue')
plt.text(2021, max(usa_sales_group)*0.5, 'Biden EV Goals', color='orange')

def millions(x, pos):
    return f'{int(x / 1e6)}M'  # Convert to millions

formatter = FuncFormatter(millions)
plt.gca().yaxis.set_major_formatter(formatter)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Legend")

# Setting labels and title
plt.xlabel('Year')
plt.ylabel('EV Sales (Vehicles)')
plt.title('Impact of Government Policies on EV Sales in the USA')

# Show legend

# Show plot
plt.tight_layout(rect=[0, 0, 0.85, 1])
plt.show()
```

```
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

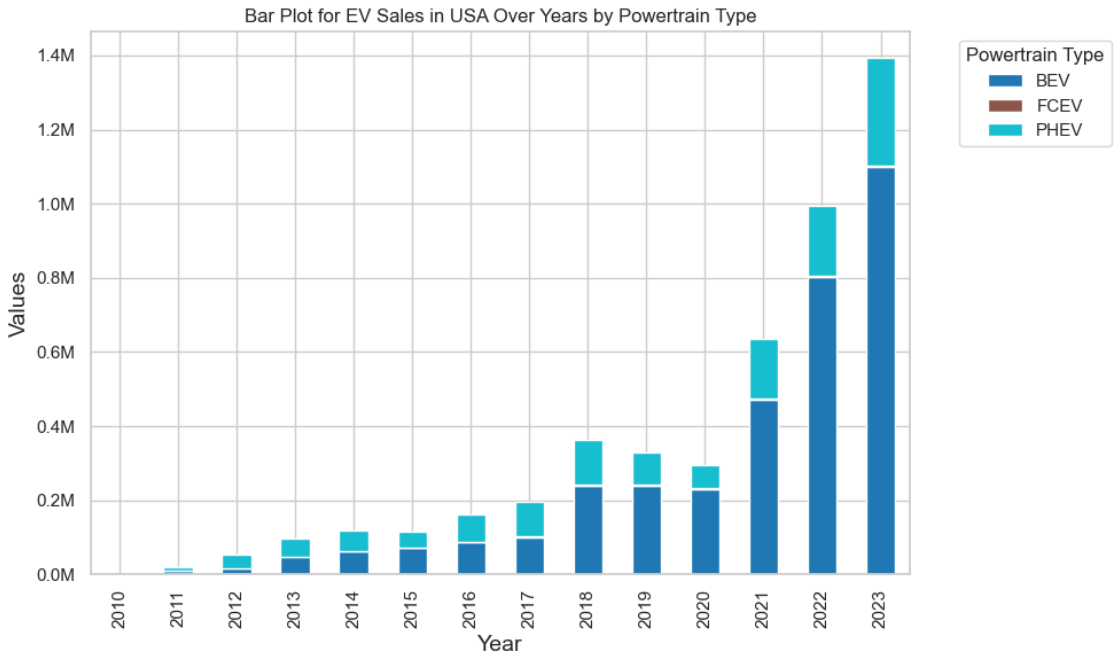
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[171], line 38
     35 plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Legend")
     37 # Setting labels and title
---> 38 plt.xlabel('Year')
     39 plt.ylabel('EV Sales (Vehicles)')
     40 plt.title('Impact of Government Policies on EV Sales in the USA')

TypeError: 'str' object is not callable
```



[11]:
```
usa_sales = df[(df['region'] == 'USA') & (df['parameter'] == 'EV sales') &␣
 ↪(df['category'] == 'Historical')]
usa_filter = usa_sales[usa_sales['year'] < 2024]
sns.set(style="whitegrid")
# Group by year and powertrain, then sum the sales values
usa_sale_grp = usa_filter.groupby(['year', 'powertrain'])['value'].sum().
 ↪reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))
```

```
# Stacked barplot by year and powertrain type
usa_sale_pivot = usa_sale_grp.pivot(index='year', columns='powertrain',␣
 ↪values='value').fillna(0)

# Plot each powertrain type as a stacked bar
usa_sale_pivot.plot(kind='bar', stacked=True, figsize=(10, 6), colormap='tab10')

# Formatting the y-axis to display values in millions
plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))
# Setting labels and title
plt.title('Bar Plot for EV Sales in USA Over Years by Powertrain Type')
plt.xlabel('Year',fontsize=14)
plt.ylabel('Values',fontsize=14)
# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.show()
```

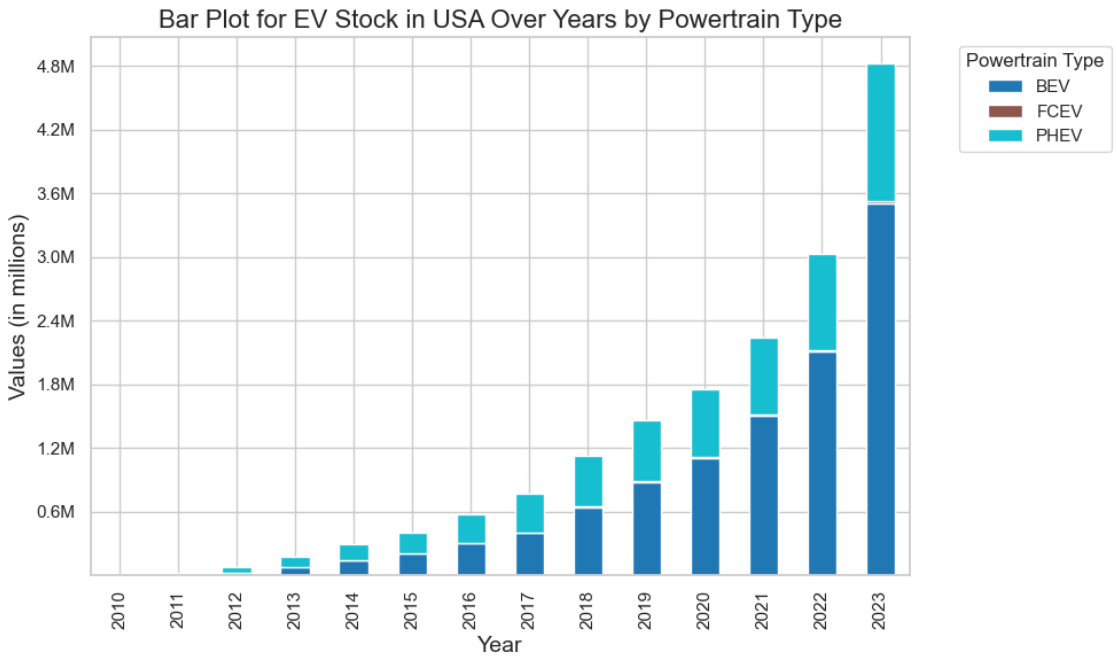
```
<Figure size 1000x600 with 0 Axes>
```



[14]:
```
sns.set(style="whitegrid")
# Filter data for USA (EV stock)
```

```
usa_stock = df[(df['region'] == 'USA') & (df['parameter'] == 'EV stock') &␣
 ↪(df['category'] == 'Historical')]
usa_stock_filter = usa_stock[usa_stock['year'] < 2024]

# Group by year and powertrain, then sum the stock values
usa_stock_grp = usa_stock_filter.groupby(['year', 'powertrain'])['value'].sum().
 ↪reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))

# Stacked barplot by year and powertrain type for USA (EV stock)
usa_stock_pivot = usa_stock_grp.pivot(index='year', columns='powertrain',␣
 ↪values='value').fillna(0)

# Plot each powertrain type as a stacked bar
usa_stock_pivot.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in millions
plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))

# Adjusting the y-axis tick interval and formatting
plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True, prune='lower'))  #␣
 ↪Ensure integer ticks and no lower bound

# Setting labels and title
plt.title('Bar Plot for EV Stock in USA Over Years by Powertrain Type',␣
 ↪fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Values (in millions)', fontsize=14)

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.show()
```

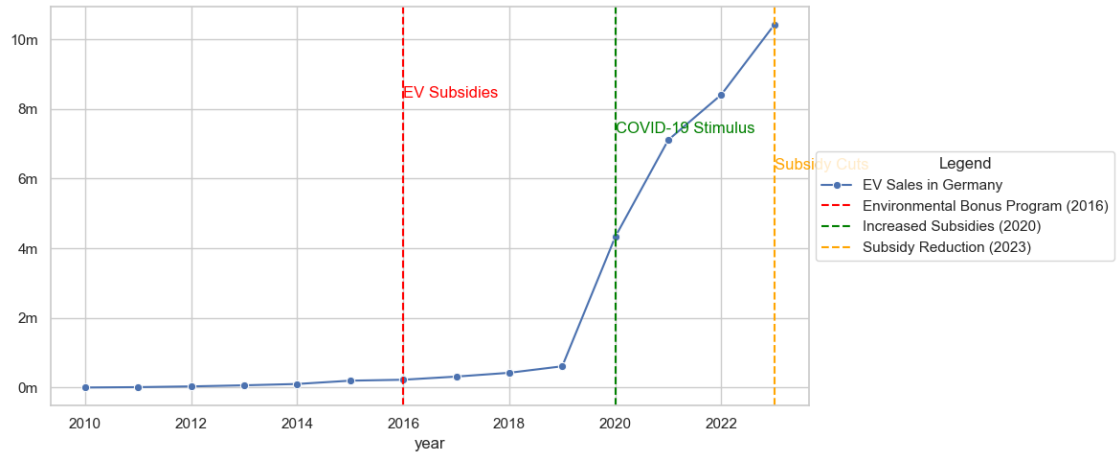
```
<Figure size 1000x600 with 0 Axes>
```



[40]:
```
sns.set(style="whitegrid")
# Filtering data for Germany with EV sales and year < 2024
germany_sales = df[(df['region'] == 'Europe') & (df['parameter'] == 'EV sales')␣
 ↪& (df['year'] < 2024)]

# Group by year and sum the sales values
germany_sales_group = germany_sales.groupby('year')['value'].sum()

# Plot the EV sales time-series
plt.figure(figsize=(14, 5))
sns.lineplot(x=germany_sales_group.index, y=germany_sales_group.values,␣
 ↪marker='o', label='EV Sales in Germany')

# Adding vertical lines to indicate policy events
plt.axvline(x=2016, color='red', linestyle='--', label='Environmental Bonus␣
 ↪Program (2016)')
plt.axvline(x=2020, color='green', linestyle='--', label='Increased Subsidies␣
 ↪(2020)')
plt.axvline(x=2023, color='orange', linestyle='--', label='Subsidy Reduction␣
 ↪(2023)')

def million(x, pos):
    return f'{int(x / 1e6)}m'  # Convert to millions

formatter = FuncFormatter(million)
```

```
plt.gca().yaxis.set_major_formatter(formatter)


# Adding annotations for policies
plt.text(2016, max(germany_sales_group)*0.8, 'EV Subsidies', color='red')
plt.text(2020, max(germany_sales_group)*0.7, 'COVID-19 Stimulus', color='green')
plt.text(2023, max(germany_sales_group)*0.6, 'Subsidy Cuts', color='orange')

# Setting labels and title
plt.xlabel=('Year')
plt.ylabel=('EV Sales (Vehicles)')
plt.title=('Impact of Government Policies on EV Sales in Germany')

# Show legend
plt.legend(loc='center left',bbox_to_anchor=(1,0.5),title="Legend")

# Show plot
plt.tight_layout(rect=[0,0,0.85,1])
plt.show()
```

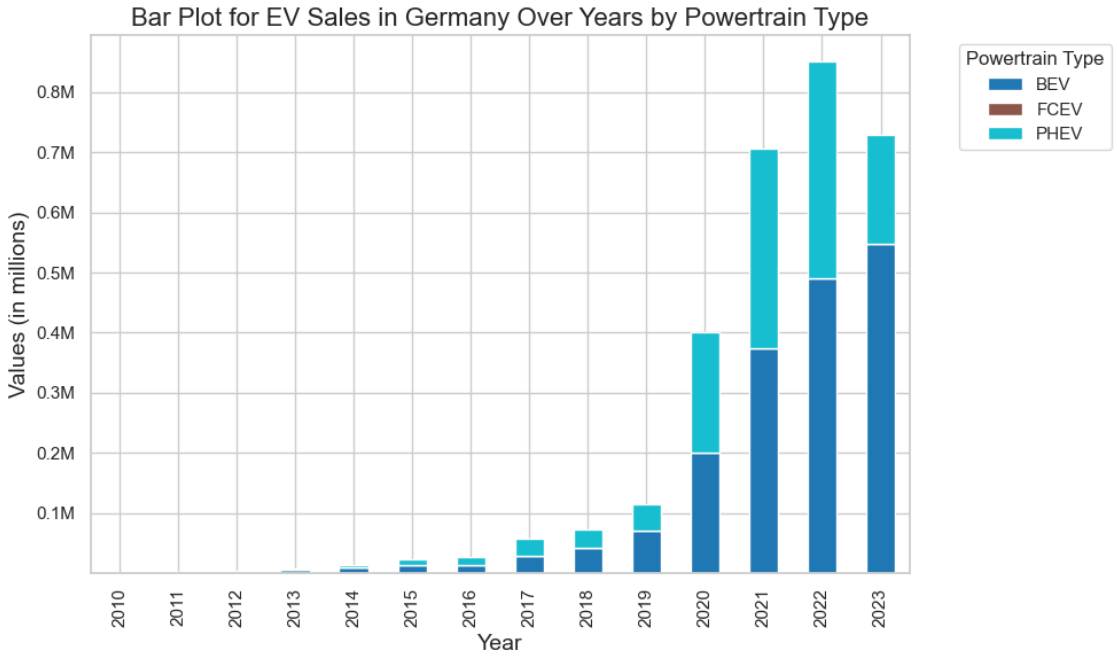
```
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[17]:
```
# Filter data for Germany (EV sales)
```

```
germany_sales = df[(df['region'] == 'Germany') & (df['parameter'] == 'EV␣
 ↪sales') & (df['category'] == 'Historical')]
germany_sales_filter = germany_sales[germany_sales['year'] < 2024]

# Group by year and powertrain, then sum the sales values
germany_sales_grp = germany_sales_filter.groupby(['year',␣
 ↪'powertrain'])['value'].sum().reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))

# Stacked barplot by year and powertrain type for Germany (EV sales)
germany_sales_pivot = germany_sales_grp.pivot(index='year',␣
 ↪columns='powertrain', values='value').fillna(0)

# Plot each powertrain type as a stacked bar
germany_sales_pivot.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in millions
plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))

# Adjusting the y-axis tick interval and formatting
plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True, prune='lower'))  #␣
 ↪Ensure integer ticks and no lower bound

# Setting labels and title
plt.title('Bar Plot for EV Sales in Germany Over Years by Powertrain Type',␣
 ↪fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Values (in millions)', fontsize=14)

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.show()
```

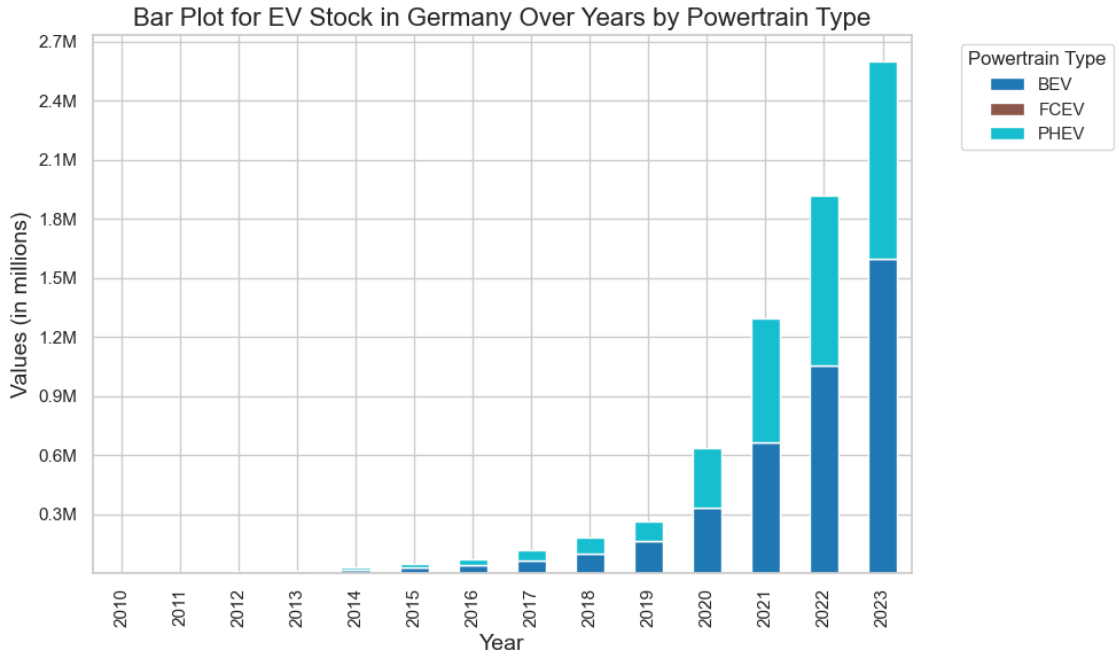
```
<Figure size 1000x600 with 0 Axes>
```



[16]:
```
# Filter data for Germany (EV stock)
germany_stock = df[(df['region'] == 'Germany') & (df['parameter'] == 'EV␣
 ↪stock') & (df['category'] == 'Historical')]
germany_stock_filter = germany_stock[germany_stock['year'] < 2024]

# Group by year and powertrain, then sum the stock values
germany_stock_grp = germany_stock_filter.groupby(['year',␣
 ↪'powertrain'])['value'].sum().reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))

# Stacked barplot by year and powertrain type for Germany (EV stock)
germany_stock_pivot = germany_stock_grp.pivot(index='year',␣
 ↪columns='powertrain', values='value').fillna(0)

# Plot each powertrain type as a stacked bar
germany_stock_pivot.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in millions
```

```
plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))

# Adjusting the y-axis tick interval and formatting
plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True, prune='lower'))  #␣
 ↪Ensure integer ticks and no lower bound

# Setting labels and title
plt.title('Bar Plot for EV Stock in Germany Over Years by Powertrain Type',␣
 ↪fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Values (in millions)', fontsize=14)

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



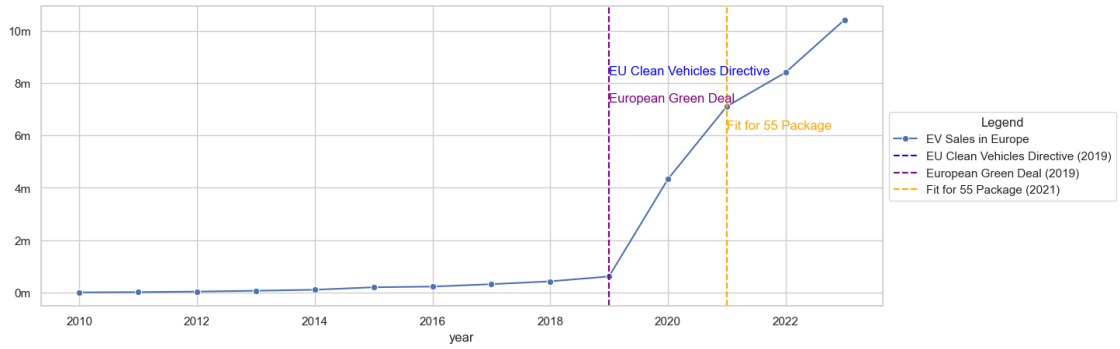
[47]:
```
sns.set(style="whitegrid")
# Filter data for Europe with EV sales and year < 2024
europe_sales = df[(df['region'] == 'Europe') & (df['parameter'] == 'EV sales')␣
 ↪& (df['year'] < 2024)]
```

```
# Group by year and sum the sales values
europe_sales_group = europe_sales.groupby('year')['value'].sum()

# Plot the EV sales time-series
plt.figure(figsize=(14, 5))
sns.lineplot(x=europe_sales_group.index, y=europe_sales_group.values,␣
 ↪marker='o', label='EV Sales in Europe')

# Adding vertical lines to indicate policy events
plt.axvline(x=2019, color='blue', linestyle='--', label='EU Clean Vehicles␣
 ↪Directive (2019)')
plt.axvline(x=2019, color='purple', linestyle='--', label='European Green Deal␣
 ↪(2019)')
plt.axvline(x=2021, color='orange', linestyle='--', label='Fit for 55 Package␣
 ↪(2021)')

# Create a function to format y-axis labels
def million(x, pos):
    return f'{int(x / 1e6)}m'  # Convert to millions

formatter = FuncFormatter(million)
plt.gca().yaxis.set_major_formatter(formatter)

# Adding annotations for policies
plt.text(2019, max(europe_sales_group) * 0.8, 'EU Clean Vehicles Directive',␣
 ↪color='blue')
plt.text(2019, max(europe_sales_group) * 0.7, 'European Green Deal',␣
 ↪color='purple')
plt.text(2021, max(europe_sales_group) * 0.6, 'Fit for 55 Package',␣
 ↪color='orange')

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Legend")
# Setting labels and title
plt.xlabel('Year')
plt.ylabel('EV Sales (Vehicles)')
plt.title('Impact of Government Policies on EV Sales in Europe')

# Show legend

# Show plot
plt.tight_layout(rect=[0, 0, 0.85, 1])
plt.show()
```

```
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

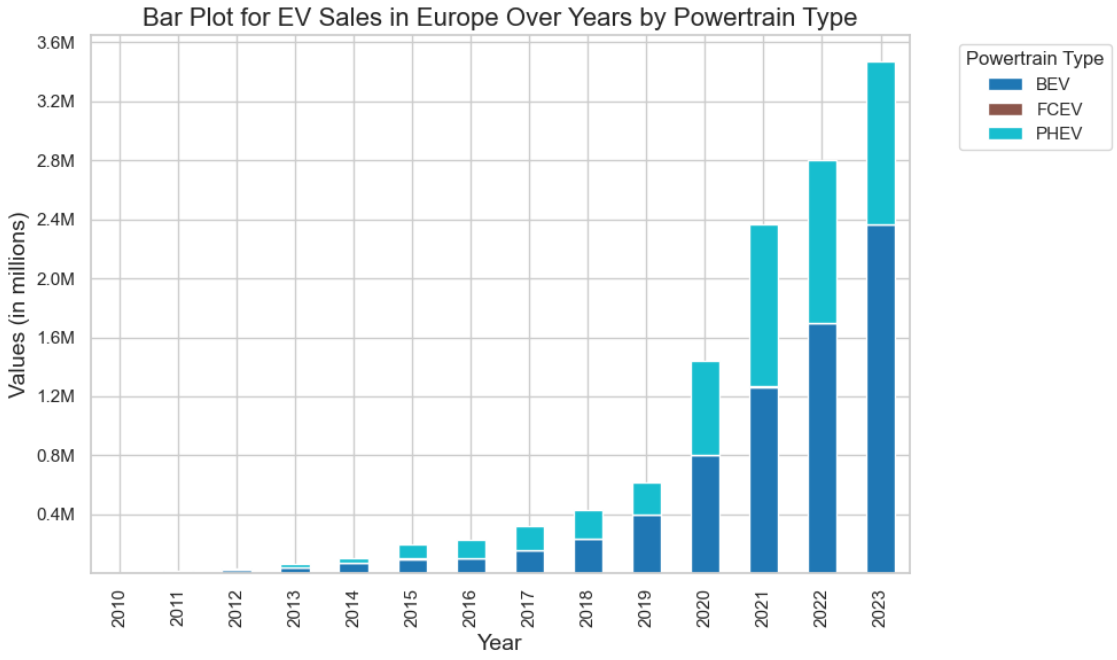
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[47], line 37
     35 plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Legend")
     36 # Setting labels and title
---> 37 plt.xlabel('Year')
     38 plt.ylabel('EV Sales (Vehicles)')
     39 plt.title('Impact of Government Policies on EV Sales in Europe')

TypeError: 'str' object is not callable
```



[18]:
```
import pandas as pd
# Filter data for Europe (EV sales)
europe_sales = df[(df['region'] == 'Europe') & (df['parameter'] == 'EV sales')␣
 ↪& (df['category'] == 'Historical')]
europe_sales_filter = europe_sales[europe_sales['year'] < 2024]

# Group by year and powertrain, then sum the sales values
europe_sales_grp = europe_sales_filter.groupby(['year', 'powertrain'])['value'].
 ↪sum().reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))
```

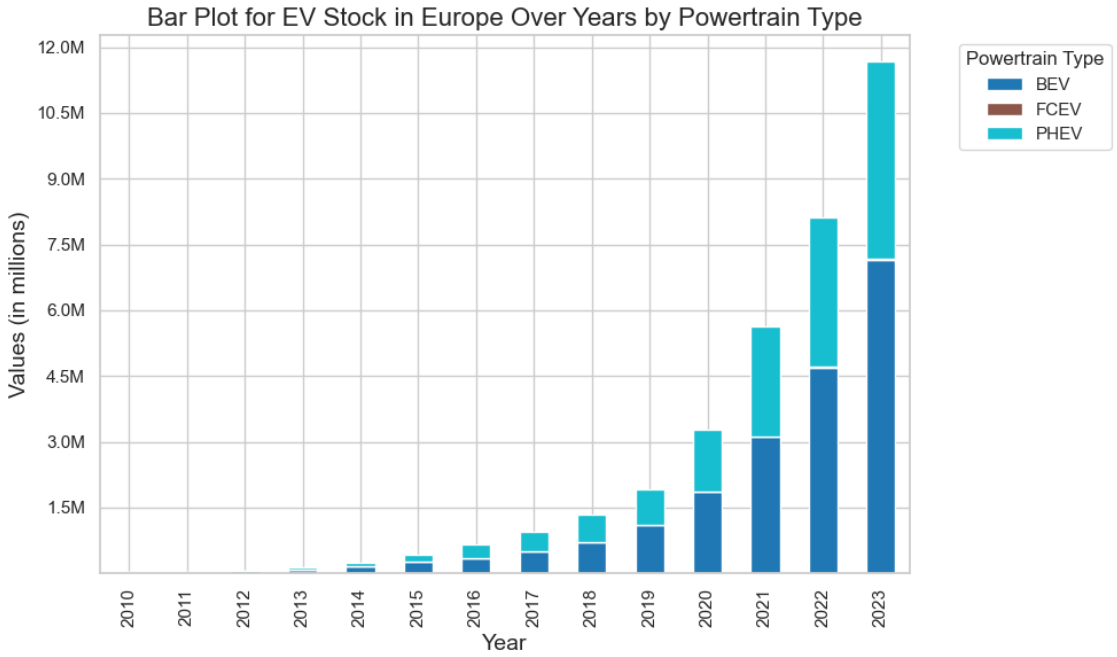
```
# Stacked barplot by year and powertrain type for Europe (EV sales)
europe_sales_pivot = europe_sales_grp.pivot(index='year', columns='powertrain',␣
 ↪values='value').fillna(0)

# Plot each powertrain type as a stacked bar
europe_sales_pivot.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in millions
plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))

# Adjusting the y-axis tick interval and formatting
plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True, prune='lower'))  #␣
 ↪Ensure integer ticks and no lower bound

# Setting labels and title
plt.title('Bar Plot for EV Sales in Europe Over Years by Powertrain Type',␣
 ↪fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Values (in millions)', fontsize=14)

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



[19]:
```
# Filter data for Europe (EV stock)
europe_stock = df[(df['region'] == 'Europe') & (df['parameter'] == 'EV stock')␣
 ↪& (df['category'] == 'Historical')]
europe_stock_filter = europe_stock[europe_stock['year'] < 2024]

# Group by year and powertrain, then sum the stock values
europe_stock_grp = europe_stock_filter.groupby(['year', 'powertrain'])['value'].
 ↪sum().reset_index()

# Function to format y-axis in millions
def millions(x, pos):
    return f'{x * 1e-6:.1f}M'

# Plotting
plt.figure(figsize=(10, 6))

# Stacked barplot by year and powertrain type for Europe (EV stock)
europe_stock_pivot = europe_stock_grp.pivot(index='year', columns='powertrain',␣
 ↪values='value').fillna(0)

# Plot each powertrain type as a stacked bar
europe_stock_pivot.plot(kind='bar', stacked=True, figsize=(10, 6),␣
 ↪colormap='tab10')

# Formatting the y-axis to display values in millions
plt.gca().yaxis.set_major_formatter(FuncFormatter(millions))

# Adjusting the y-axis tick interval and formatting
plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True, prune='lower'))  #␣
 ↪Ensure integer ticks and no lower bound

# Setting labels and title
plt.title('Bar Plot for EV Stock in Europe Over Years by Powertrain Type',␣
 ↪fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Values (in millions)', fontsize=14)

# Show legend and plot
plt.legend(title='Powertrain Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()

plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



China EV Market Predictions -> Sales share, sales, stock share

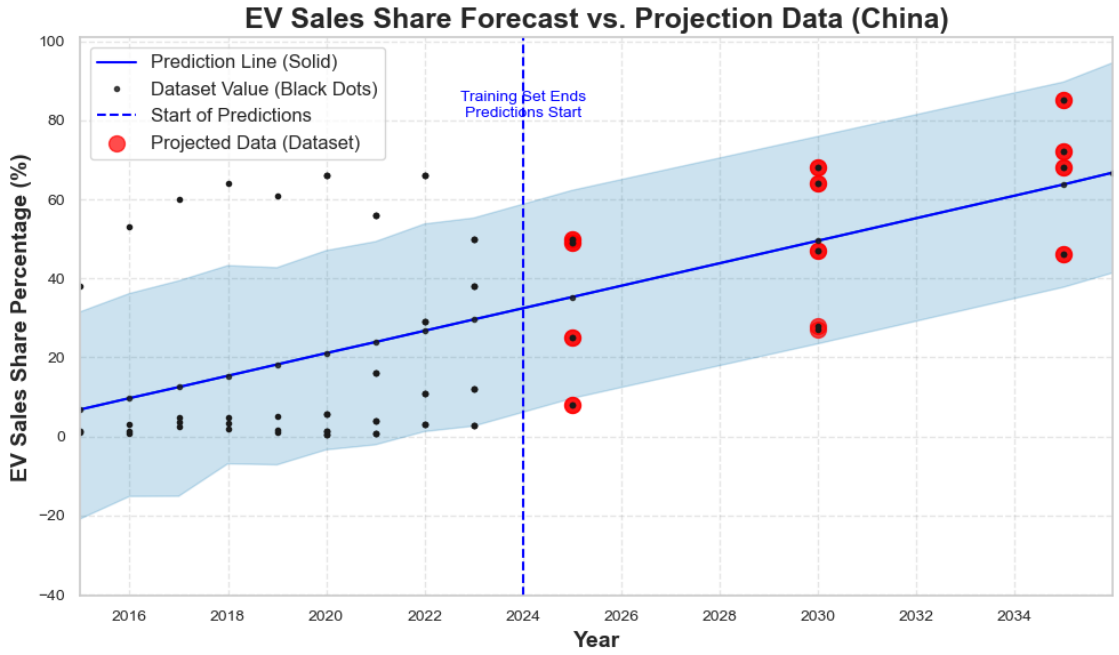
[99]:
```
#china sales share
from prophet.diagnostics import cross_validation, performance_metrics

# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV SALES SHARE Data Preparation for China (Updated variable names)
df_china_sales_share_hist = df[(df['region'] == 'China') &
                               (df['parameter'] == 'EV sales share') &
                               (df['year'] < 2024)]  # Historical data (before␣
 ↪2024)

df_china_sales_share_proj = df[(df['region'] == 'China') &
                               (df['parameter'] == 'EV sales share') &
                               (df['year'] >= 2024) &
                               (df['year'] <= 2035)]  # Projection data (2024␣
 ↪to 2035)

# Prepare the projection data for plotting
df_china_sales_share_proj['time'] = pd.
 ↪to_datetime(df_china_sales_share_proj['year'], format='%Y')
df_china_sales_share_proj['percentage (%)'] = df_china_sales_share_proj['value']
```

```
# Combine historical and projection data for training
df_sales_share_combined_china = pd.concat([df_china_sales_share_hist,␣
 ↪df_china_sales_share_proj])

df_sales_share_combined_china['time'] = pd.
 ↪to_datetime(df_sales_share_combined_china['year'], format='%Y')
df_sales_share_combined_china['percentage (%)'] =␣
 ↪df_sales_share_combined_china['value']

# Step 2: Initialize and fit the Prophet model with yearly seasonality and␣
 ↪changepoint sensitivity
sales_share_model_china = Prophet(yearly_seasonality=True,  # Enable yearly␣
 ↪seasonality
                                  changepoint_prior_scale=0.05)  # Adjust␣
 ↪changepoint sensitivity
sales_share_model_china.fit(df_sales_share_combined_china.
 ↪rename(columns={'time': 'ds', 'percentage (%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_sales_share_years_china = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_sales_share_china = sales_share_model_china.
 ↪make_future_dataframe(periods=future_sales_share_years_china, freq='Y')

# Restrict future data to end at 2035
future_sales_share_china =␣
 ↪future_sales_share_china[future_sales_share_china['ds'] <= pd.
 ↪to_datetime('2035-12-31')]  # Future data only until 2035

# Make predictions
forecast_sales_share_china = sales_share_model_china.
 ↪predict(future_sales_share_china)

# Set a lower bound for predictions
forecast_sales_share_china['yhat'] = forecast_sales_share_china['yhat'].
 ↪clip(lower=0)  # Clip negative predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot_share_china = sales_share_model_china.
 ↪plot(forecast_sales_share_china, xlabel='Year', ylabel='Percentage (%)',␣
 ↪ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
```

```
prediction_line_share_china, = ax.plot(forecast_sales_share_china['ds'],␣
 ↪forecast_sales_share_china['yhat'], color='blue', label='Prediction Line␣
 ↪(Solid)')

# Step 5: Overlay the projection data on the same plot
projected_data_share_china = ax.scatter(df_china_sales_share_proj['time'],␣
 ↪df_china_sales_share_proj['percentage (%)'],
                                        color='red', marker='o', s=100,␣
 ↪label='Projected Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points_share_china = ax.plot(forecast_sales_share_china['ds'],␣
 ↪forecast_sales_share_china['yhat'], 'k.', label='Forecast Data (Black␣
 ↪Dots)', alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
prediction_start_share_china = ax.axvline(x=pd.to_datetime('2024-01-01'),␣
 ↪color='blue', linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Sales Share Forecast vs. Projection Data (China)',␣
 ↪fontsize=18, fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Sales Share Percentage (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')

# Step 8: Add the legend explicitly
legend_labels_share_china = ['Prediction Line (Solid)', 'Dataset Value (Black␣
 ↪Dots)', 'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line_share_china, forecast_points_share_china[0],␣
 ↪prediction_start_share_china, projected_data_share_china],
          legend_labels_share_china, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-12-31'))

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

```
# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
df_sales_share_cv_china = cross_validation(sales_share_model_china,␣
 ↪initial='730 days', period='365 days', horizon='365 days')

# Calculate performance metrics
df_sales_share_p_china = performance_metrics(df_sales_share_cv_china)

# Output performance metrics
print(df_sales_share_p_china[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\4017358167.py:18:
SettingWithCopyWarning:
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
  df_china_sales_share_proj['time'] =
pd.to_datetime(df_china_sales_share_proj['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\4017358167.py:19:
SettingWithCopyWarning:
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
  df_china_sales_share_proj['percentage (%)'] =
df_china_sales_share_proj['value']
23:26:37 - cmdstanpy - INFO - Chain [1] start processing
23:26:37 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
23:26:39 - cmdstanpy - INFO - Chain [1] start processing
23:26:39 - cmdstanpy - INFO - Chain [1] done processing
23:26:39 - cmdstanpy - INFO - Chain [1] start processing
23:26:39 - cmdstanpy - INFO - Chain [1] done processing
23:26:39 - cmdstanpy - INFO - Chain [1] start processing
23:26:39 - cmdstanpy - INFO - Chain [1] done processing
23:26:39 - cmdstanpy - INFO - Chain [1] start processing
23:26:39 - cmdstanpy - INFO - Chain [1] done processing
23:26:39 - cmdstanpy - INFO - Chain [1] start processing
23:26:40 - cmdstanpy - INFO - Chain [1] done processing
23:26:40 - cmdstanpy - INFO - Chain [1] start processing
23:26:40 - cmdstanpy - INFO - Chain [1] done processing
23:26:40 - cmdstanpy - INFO - Chain [1] start processing
23:26:40 - cmdstanpy - INFO - Chain [1] done processing
23:26:40 - cmdstanpy - INFO - Chain [1] start processing
23:26:41 - cmdstanpy - INFO - Chain [1] done processing
23:26:41 - cmdstanpy - INFO - Chain [1] start processing
23:26:41 - cmdstanpy - INFO - Chain [1] done processing
23:26:41 - cmdstanpy - INFO - Chain [1] start processing
23:26:41 - cmdstanpy - INFO - Chain [1] done processing
23:26:41 - cmdstanpy - INFO - Chain [1] start processing
23:26:41 - cmdstanpy - INFO - Chain [1] done processing
23:26:42 - cmdstanpy - INFO - Chain [1] start processing
23:26:42 - cmdstanpy - INFO - Chain [1] done processing
```

```
23:26:42 - cmdstanpy - INFO - Chain [1] start processing
23:26:42 - cmdstanpy - INFO - Chain [1] done processing
23:26:42 - cmdstanpy - INFO - Chain [1] start processing
23:26:42 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	8.528754	4.526921	15.451424
1	364 days	23.728098	9.270835	26.823162
2	365 days	18.289252	2.963868	20.474916

[121]:
```python
# SALES SHARES Accuracy

from prophet.diagnostics import cross_validation, performance_metrics
df_China_sasrH = df[(df['region'] == 'China') &
                    (df['parameter'] == 'EV sales share') &
                    (df['year'] < 2024)]  # Historical data

df_China_sasrPro = df[(df['region'] == 'China') &
                      (df['parameter'] == 'EV sales share') &
                      (df['year'] >= 2024) &
                      (df['year'] <= 2035)]  # Projection data

df_China_sasrPro['ds'] = pd.to_datetime(df_China_sasrPro['year'], format='%Y')
df_China_sasrPro['y'] = df_China_sasrPro['value']

df_combined2 = pd.concat([df_China_sasrH, df_china_sasrPro])

df_combined2['ds'] = pd.to_datetime(df_combined2['year'], format='%Y')
df_combined2['y'] = df_combined2['value']

# Step 2: Initialize and fit the Prophet model
model = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
model.fit(df_combined2)

horizon_days = (pd.to_datetime('2035-01-01') - pd.to_datetime('2024-01-01')).
 ↪days

df_cv = cross_validation(model, initial='730 days', period='365 days',␣
 ↪horizon=f'{horizon_days} days')

df_p = performance_metrics(df_cv)

start_year = 2024  # Starting year for predictions

years = [start_year + (horizon.days // 365) for horizon in df_p['horizon']]

print(df_p[['horizon', 'mae', 'mape', 'rmse']])
```

```
plt.figure(figsize=(12, 6))
plt.plot(years, df_p['rmse'], label='RMSE', marker='o')
plt.plot(years, df_p['mae'], label='MAE', marker='x')
plt.plot(years, df_p['mape'], label='MAPE (%)', marker='s')
plt.legend()
plt.title('Performance Metrics by Year', fontsize=18)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Error', fontsize=14)
plt.xticks(years, fontsize=12)  # Set x-ticks to show years
plt.yticks(fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```

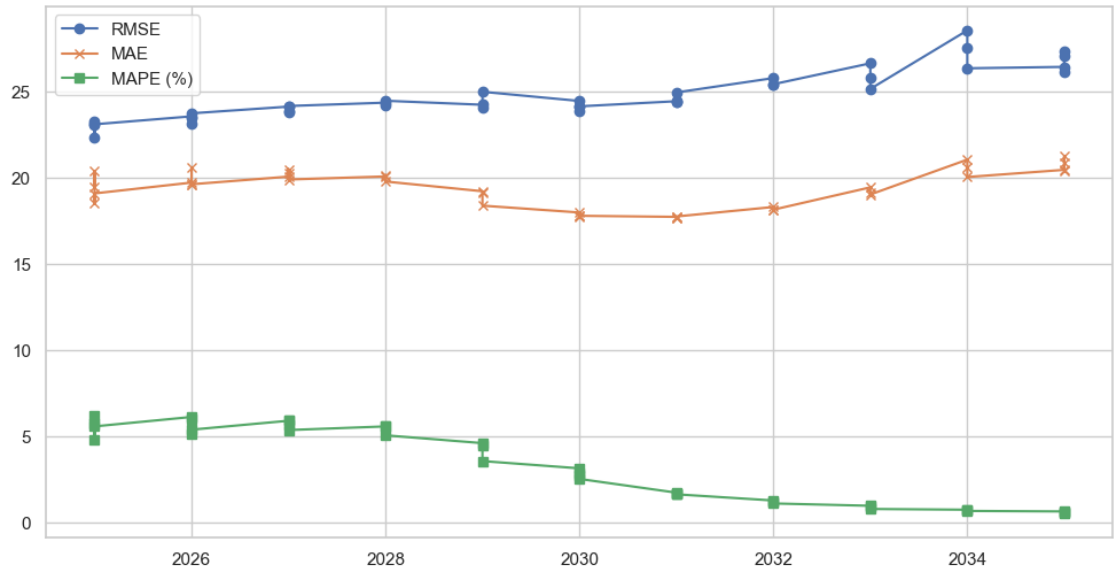
```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\671209777.py:21:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_China_sasrPro['ds'] = pd.to_datetime(df_China_sasrPro['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\671209777.py:22:
SettingWithCopyWarning:
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
  df_China_sasrPro['y'] = df_China_sasrPro['value']
23:55:49 - cmdstanpy - INFO - Chain [1] start processing
23:55:49 - cmdstanpy - INFO - Chain [1] done processing

  0%|          | 0/13 [00:00<?, ?it/s]

23:55:49 - cmdstanpy - INFO - Chain [1] start processing
23:55:49 - cmdstanpy - INFO - Chain [1] done processing
23:55:49 - cmdstanpy - INFO - Chain [1] start processing
23:55:49 - cmdstanpy - INFO - Chain [1] done processing
23:55:50 - cmdstanpy - INFO - Chain [1] start processing
23:55:50 - cmdstanpy - INFO - Chain [1] done processing
23:55:50 - cmdstanpy - INFO - Chain [1] start processing
23:55:50 - cmdstanpy - INFO - Chain [1] done processing
23:55:50 - cmdstanpy - INFO - Chain [1] start processing
23:55:50 - cmdstanpy - INFO - Chain [1] done processing
23:55:51 - cmdstanpy - INFO - Chain [1] start processing
23:55:51 - cmdstanpy - INFO - Chain [1] done processing
23:55:51 - cmdstanpy - INFO - Chain [1] start processing
23:55:51 - cmdstanpy - INFO - Chain [1] done processing
```

```
23:55:51 - cmdstanpy - INFO - Chain [1] start processing
23:55:51 - cmdstanpy - INFO - Chain [1] done processing
23:55:51 - cmdstanpy - INFO - Chain [1] start processing
23:55:52 - cmdstanpy - INFO - Chain [1] done processing
23:55:52 - cmdstanpy - INFO - Chain [1] start processing
23:55:52 - cmdstanpy - INFO - Chain [1] done processing
23:55:52 - cmdstanpy - INFO - Chain [1] start processing
23:55:52 - cmdstanpy - INFO - Chain [1] done processing
23:55:52 - cmdstanpy - INFO - Chain [1] start processing
23:55:53 - cmdstanpy - INFO - Chain [1] done processing
23:55:53 - cmdstanpy - INFO - Chain [1] start processing
23:55:53 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	365 days	19.490479	6.192271	23.102537
1	366 days	20.401479	5.783858	23.288498
2	728 days	18.527348	4.812250	22.327526
3	729 days	19.100673	5.594064	23.107132
4	730 days	19.727807	6.141318	23.568353
5	731 days	20.580573	5.995943	23.473875
6	1093 days	19.585203	5.147805	23.133895
7	1094 days	19.631066	5.404212	23.743276
8	1095 days	20.074303	5.921278	24.132394
9	1096 days	20.505419	5.962474	23.792020
10	1458 days	20.297149	5.713946	23.841384
11	1459 days	19.906980	5.381776	24.168196
12	1460 days	20.084294	5.592692	24.362072
13	1461 days	20.166940	5.596393	24.186668
14	1824 days	19.790047	5.076786	24.471736
15	1825 days	19.225788	4.623467	24.239949
16	1826 days	19.115778	4.511125	24.089064
17	2189 days	18.390677	3.583354	24.992643
18	2190 days	17.994124	3.162417	24.460183
19	2191 days	17.751898	2.864017	24.134597
20	2192 days	17.801802	2.755722	23.883954
21	2554 days	17.802571	2.548924	24.147518
22	2555 days	17.740545	1.752647	24.445443
23	2556 days	17.679146	1.680070	24.382510
24	2557 days	17.695290	1.658619	24.408016
25	2919 days	17.763450	1.657115	24.954124
26	2920 days	18.313344	1.298652	25.786338
27	2921 days	18.147272	1.199702	25.419567
28	2922 days	18.158642	1.127447	25.431138
29	3285 days	19.451978	0.991866	26.642286
30	3286 days	19.135411	0.932357	25.801783
31	3287 days	19.038281	0.807013	25.159072
32	3650 days	21.058503	0.763975	28.550231
33	3651 days	20.579893	0.753640	27.528299

```
34 3652 days  20.050833  0.692852  26.353572
35 4015 days  20.466900  0.664160  26.436048
36 4016 days  21.265090  0.628323  27.353312
37 4017 days  20.857655  0.608996  27.096111
38 4018 days  20.406897  0.566571  26.127394
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[121], line 59
     57 plt.plot(years, df_p['mape'], label='MAPE (%)', marker='s')
     58 plt.legend()
---> 59 plt.title('Performance Metrics by Year', fontsize=18)
     60 plt.xlabel('Year', fontsize=14)
     61 plt.ylabel('Error', fontsize=14)

TypeError: 'str' object is not callable
```



[127]:
```
#China Stock share
# STOCK SHARE Accuracy for China
from prophet.diagnostics import cross_validation, performance_metrics

# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV STOCK SHARE Data Preparation for China (Updated variable names)
df_china_stock_share_hist = df[(df['region'] == 'China') &
                               (df['parameter'] == 'EV stock share') &
```

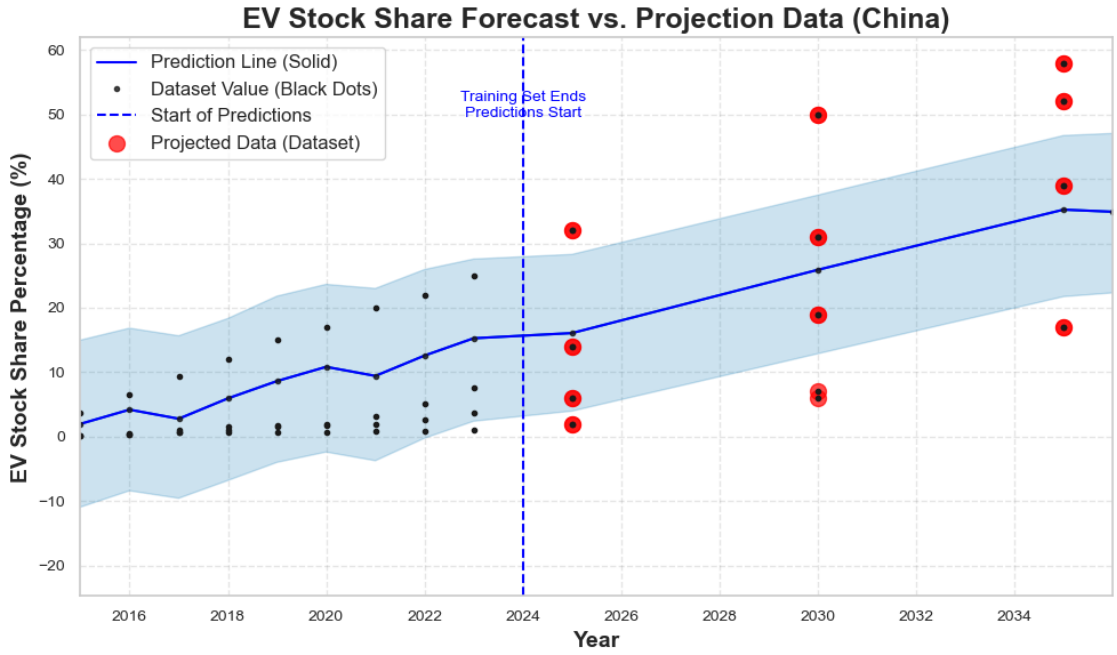
```
                                (df['category']=='Historical') &
                                (df['year'] < 2024)]  # Historical data (before␣
 ↪2024)

df_china_stock_share_proj = df[(df['region'] == 'China') &
                                (df['parameter'] == 'EV stock share') &
                                (df['year'] >= 2024) &
                                (df['year'] <= 2035)]  # Projection data (2024␣
 ↪to 2035)

# Prepare the projection data for plotting
df_china_stock_share_proj['time'] = pd.
 ↪to_datetime(df_china_stock_share_proj['year'], format='%Y')
df_china_stock_share_proj['percentage (%)'] = df_china_stock_share_proj['value']

# Combine historical and projection data for training
df_stock_share_combined_china = pd.concat([df_china_stock_share_hist,␣
 ↪df_china_stock_share_proj])

df_stock_share_combined_china['time'] = pd.
 ↪to_datetime(df_stock_share_combined_china['year'], format='%Y')
df_stock_share_combined_china['percentage (%)'] =␣
 ↪df_stock_share_combined_china['value']

# Step 2: Initialize and fit the Prophet model with yearly seasonality and␣
 ↪changepoint sensitivity
stock_share_model_china = Prophet(yearly_seasonality=True,  # Enable yearly␣
 ↪seasonality
                                  changepoint_prior_scale=0.05)  # Adjust␣
 ↪changepoint sensitivity
stock_share_model_china.fit(df_stock_share_combined_china.
 ↪rename(columns={'time': 'ds', 'percentage (%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_stock_share_years_china = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_stock_share_china = stock_share_model_china.
 ↪make_future_dataframe(periods=future_stock_share_years_china, freq='Y')

# Restrict future data to end at 2035
future_stock_share_china =␣
 ↪future_stock_share_china[future_stock_share_china['ds'] <= pd.
 ↪to_datetime('2035-12-31')]  # Future data only until 2035

# Make predictions
```

```
forecast_stock_share_china = stock_share_model_china.
 ↪predict(future_stock_share_china)

# Set a lower bound for predictions
forecast_stock_share_china['yhat'] = forecast_stock_share_china['yhat'].
 ↪clip(lower=0)  # Clip negative predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot_stock_share_china = stock_share_model_china.
 ↪plot(forecast_stock_share_china, xlabel='Year', ylabel='Percentage (%)',␣
 ↪ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
prediction_line_stock_share_china, = ax.plot(forecast_stock_share_china['ds'],␣
 ↪forecast_stock_share_china['yhat'], color='blue', label='Prediction Line␣
 ↪(Solid)')

# Step 5: Overlay the projection data on the same plot
projected_data_stock_share_china = ax.
 ↪scatter(df_china_stock_share_proj['time'],␣
 ↪df_china_stock_share_proj['percentage (%)'],
                                                color='red', marker='o', s=100,␣
 ↪label='Projected Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points_stock_share_china = ax.plot(forecast_stock_share_china['ds'],␣
 ↪forecast_stock_share_china['yhat'], 'k.', label='Forecast Data (Black␣
 ↪Dots)', alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
prediction_start_stock_share_china = ax.axvline(x=pd.to_datetime('2024-01-01'),␣
 ↪color='blue', linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Stock Share Forecast vs. Projection Data (China)',␣
 ↪fontsize=18, fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Stock Share Percentage (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
```

```
ax.grid(True, linestyle='--')

# Step 8: Add the legend explicitly
legend_labels_stock_share_china = ['Prediction Line (Solid)', 'Dataset Value␣
 ↪(Black Dots)', 'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line_stock_share_china,␣
 ↪forecast_points_stock_share_china[0], prediction_start_stock_share_china,␣
 ↪projected_data_stock_share_china],
          legend_labels_stock_share_china, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-12-31'))

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

# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
df_stock_share_cv_china = cross_validation(stock_share_model_china,␣
 ↪initial='730 days', period='365 days', horizon='365 days')

# Calculate performance metrics
df_stock_share_p_china = performance_metrics(df_stock_share_cv_china)

# Output performance metrics
print(df_stock_share_p_china[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\2926731562.py:25:
SettingWithCopyWarning:
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
  df_china_stock_share_proj['time'] =
pd.to_datetime(df_china_stock_share_proj['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\2926731562.py:26:
SettingWithCopyWarning:
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
  df_china_stock_share_proj['percentage (%)'] =
df_china_stock_share_proj['value']
```

```
00:04:07 - cmdstanpy - INFO - Chain [1] start processing
00:04:07 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
00:04:08 - cmdstanpy - INFO - Chain [1] start processing
00:04:09 - cmdstanpy - INFO - Chain [1] done processing
00:04:09 - cmdstanpy - INFO - Chain [1] start processing
00:04:09 - cmdstanpy - INFO - Chain [1] done processing
00:04:09 - cmdstanpy - INFO - Chain [1] start processing
00:04:09 - cmdstanpy - INFO - Chain [1] done processing
00:04:09 - cmdstanpy - INFO - Chain [1] start processing
00:04:09 - cmdstanpy - INFO - Chain [1] done processing
00:04:09 - cmdstanpy - INFO - Chain [1] start processing
00:04:10 - cmdstanpy - INFO - Chain [1] done processing
00:04:10 - cmdstanpy - INFO - Chain [1] start processing
00:04:10 - cmdstanpy - INFO - Chain [1] done processing
```

```
00:04:10 - cmdstanpy - INFO - Chain [1] start processing
00:04:10 - cmdstanpy - INFO - Chain [1] done processing
00:04:10 - cmdstanpy - INFO - Chain [1] start processing
00:04:10 - cmdstanpy - INFO - Chain [1] done processing
00:04:11 - cmdstanpy - INFO - Chain [1] start processing
00:04:11 - cmdstanpy - INFO - Chain [1] done processing
00:04:11 - cmdstanpy - INFO - Chain [1] start processing
00:04:11 - cmdstanpy - INFO - Chain [1] done processing
00:04:11 - cmdstanpy - INFO - Chain [1] start processing
00:04:11 - cmdstanpy - INFO - Chain [1] done processing
00:04:11 - cmdstanpy - INFO - Chain [1] start processing
00:04:11 - cmdstanpy - INFO - Chain [1] done processing
00:04:12 - cmdstanpy - INFO - Chain [1] start processing
00:04:12 - cmdstanpy - INFO - Chain [1] done processing
00:04:12 - cmdstanpy - INFO - Chain [1] start processing
00:04:12 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	0.768409	2.323645	1.679664
1	364 days	3.935165	1.474875	5.570351
2	365 days	12.023757	1.246547	15.528842

[145]:
```python
#china stock share accuracy
# STOCK SHARES Accuracy for China

from prophet.diagnostics import cross_validation, performance_metrics

# Sample Data Preparation (replace with your actual DataFrame)
# Assuming 'df' is your original DataFrame with columns: region, parameter,␣
 ↪year, and value
# Filter for EV stock share
df_China_stockH = df[(df['region'] == 'China') &
                     (df['parameter'] == 'EV stock share') &
                     (df['year'] < 2024)]

df_China_stockPro = df[(df['region'] == 'China') &
                       (df['parameter'] == 'EV stock share') &
                       (df['year'] >= 2024) &
                       (df['year'] <= 2035)]  # Projection data

# Prepare the projection data for plotting
df_China_stockPro['ds'] = pd.to_datetime(df_China_stockPro['year'], format='%Y')
df_China_stockPro['y'] = df_China_stockPro['value']

# Combine historical and projection data for training
df_combined_stock = pd.concat([df_China_stockH, df_China_stockPro])

df_combined_stock['ds'] = pd.to_datetime(df_combined_stock['year'], format='%Y')
```

```
df_combined_stock['y'] = df_combined_stock['value']

# Step 2: Initialize and fit the Prophet model
model_stock = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
model_stock.fit(df_combined_stock)

# Step 3: Cross-Validation to assess model performance with different horizons
# Calculate the total number of days from 2024 to 2035
horizon_days_stock = (pd.to_datetime('2035-01-01') - pd.
 ↪to_datetime('2024-01-01')).days

# Perform cross-validation with horizon set to the calculated days
df_cv_stock = cross_validation(model_stock, initial='730 days', period='365␣
 ↪days', horizon=f'{horizon_days_stock} days')

# Step 4: Calculate performance metrics
df_p_stock = performance_metrics(df_cv_stock)

# Generate year labels based on the prediction horizons
start_year_stock = 2024  # Starting year for predictions

# Convert horizons from days to years
years_stock = [start_year_stock + (horizon.days // 365) for horizon in␣
 ↪df_p_stock['horizon']]

# Output performance metrics
print(df_p_stock[['horizon', 'mae', 'mape', 'rmse']])

# Optional: Plot the performance metrics with years on the x-axis
plt.figure(figsize=(12, 6))
plt.plot(years_stock, df_p_stock['rmse'], label='RMSE', marker='o')
plt.plot(years_stock, df_p_stock['mae'], label='MAE', marker='x')
plt.plot(years_stock, df_p_stock['mape'], label='MAPE (%)', marker='s')
plt.legend()
plt.title('Performance Metrics by Year (EV Stock Share - China)', fontsize=18)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Error', fontsize=14)
plt.xticks(years_stock, fontsize=12)  # Set x-ticks to show years
plt.yticks(fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\1220157845.py:22:
SettingWithCopyWarning:
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
```

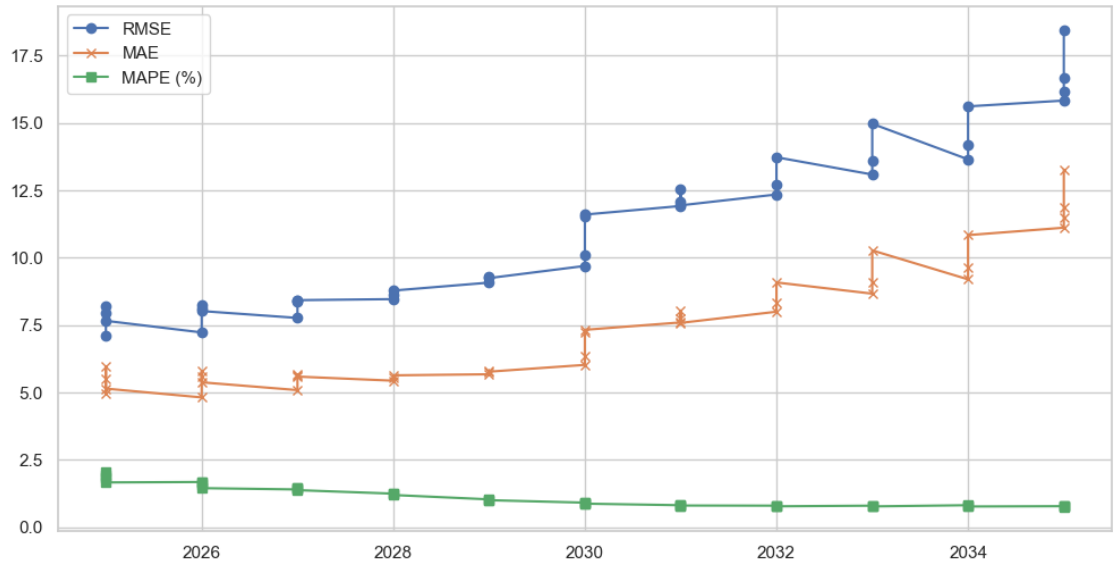
```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_China_stockPro['ds'] = pd.to_datetime(df_China_stockPro['year'],
format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\1220157845.py:23:
SettingWithCopyWarning:
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
  df_China_stockPro['y'] = df_China_stockPro['value']
00:19:34 - cmdstanpy - INFO - Chain [1] start processing
00:19:34 - cmdstanpy - INFO - Chain [1] done processing

  0%|          | 0/13 [00:00<?, ?it/s]

00:19:34 - cmdstanpy - INFO - Chain [1] start processing
00:19:34 - cmdstanpy - INFO - Chain [1] done processing
00:19:34 - cmdstanpy - INFO - Chain [1] start processing
00:19:34 - cmdstanpy - INFO - Chain [1] done processing
00:19:34 - cmdstanpy - INFO - Chain [1] start processing
00:19:34 - cmdstanpy - INFO - Chain [1] done processing
00:19:35 - cmdstanpy - INFO - Chain [1] start processing
00:19:35 - cmdstanpy - INFO - Chain [1] done processing
00:19:35 - cmdstanpy - INFO - Chain [1] start processing
00:19:35 - cmdstanpy - INFO - Chain [1] done processing
00:19:35 - cmdstanpy - INFO - Chain [1] start processing
00:19:35 - cmdstanpy - INFO - Chain [1] done processing
00:19:36 - cmdstanpy - INFO - Chain [1] start processing
00:19:36 - cmdstanpy - INFO - Chain [1] done processing
00:19:36 - cmdstanpy - INFO - Chain [1] start processing
00:19:36 - cmdstanpy - INFO - Chain [1] done processing
00:19:36 - cmdstanpy - INFO - Chain [1] start processing
00:19:36 - cmdstanpy - INFO - Chain [1] done processing
00:19:37 - cmdstanpy - INFO - Chain [1] start processing
00:19:37 - cmdstanpy - INFO - Chain [1] done processing
00:19:37 - cmdstanpy - INFO - Chain [1] start processing
00:19:37 - cmdstanpy - INFO - Chain [1] done processing
00:19:37 - cmdstanpy - INFO - Chain [1] start processing
00:19:37 - cmdstanpy - INFO - Chain [1] done processing
00:19:37 - cmdstanpy - INFO - Chain [1] start processing
00:19:38 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	365 days	4.973621	2.050863	7.122875
1	366 days	5.957468	1.943044	8.182666
2	728 days	5.521485	1.859908	7.937426

```
3   729 days   5.143463  1.663635   7.657597
4   730 days   4.816660  1.679297   7.226543
5   731 days   5.782174  1.679921   8.262437
6  1093 days   5.604219  1.596719   8.135102
7  1094 days   5.376372  1.451391   8.019623
8  1095 days   5.087776  1.403161   7.764684
9  1096 days   5.661588  1.450079   8.403341
10 1458 days   5.650967  1.452051   8.387810
11 1459 days   5.588951  1.377033   8.423466
12 1460 days   5.436685  1.247411   8.462921
13 1461 days   5.544981  1.247396   8.608701
14 1824 days   5.635129  1.197179   8.779603
15 1825 days   5.675294  1.035575   9.074546
16 1826 days   5.818873  1.004329   9.314265
17 2189 days   5.770886  1.004395   9.242148
18 2190 days   6.023098  0.913546   9.693663
19 2191 days   6.368744  0.895371  10.115377
20 2192 days   7.228061  0.853834  11.520831
21 2554 days   7.323007  0.875744  11.598523
22 2555 days   7.598295  0.818200  11.916748
23 2556 days   7.742873  0.810719  12.095456
24 2557 days   8.038936  0.796323  12.525566
25 2919 days   7.580581  0.806984  11.940749
26 2920 days   7.995495  0.801891  12.343488
27 2921 days   8.331599  0.802707  12.709991
28 2922 days   9.079821  0.778061  13.723866
29 3285 days   8.665776  0.801338  13.084358
30 3286 days   9.101834  0.789970  13.581395
31 3287 days  10.270123  0.774444  14.966820
32 3650 days   9.195950  0.818699  13.646424
33 3651 days   9.647734  0.794367  14.182037
34 3652 days  10.836673  0.771286  15.611351
35 4015 days  11.111711  0.782800  15.826931
36 4016 days  11.466890  0.795124  16.170525
37 4017 days  11.855739  0.790716  16.688393
38 4018 days  13.268244  0.755146  18.444979
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[145], line 60
     58 plt.plot(years_stock, df_p_stock['mape'], label='MAPE (%)', marker='s')
     59 plt.legend()
---> 60 plt.title('Performance Metrics by Year (EV Stock Share - China)',␣
 ↪fontsize=18)
     61 plt.xlabel('Year', fontsize=14)
     62 plt.ylabel('Error', fontsize=14)
```

```
TypeError: 'str' object is not callable
```



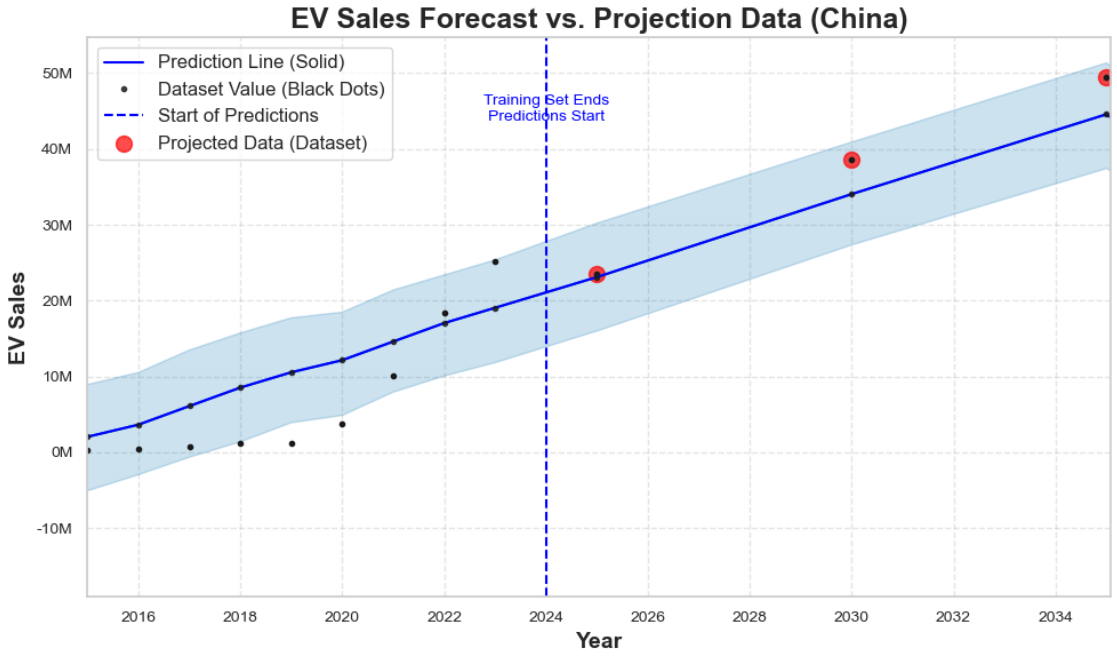
[27]:
```
#china sales
# Import libraries
from prophet.diagnostics import cross_validation, performance_metrics
from matplotlib.ticker import FuncFormatter

# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# Step 1: Aggregate Sales Data by Year for Historical and Projection Data
# Filter data for China EV sales and aggregate by year
df_china_sales_hist = df[(df['region'] == 'China') &
                         (df['parameter'] == 'EV sales') &
                         (df['year'] < 2024)]
df_china_sales_hist = df_china_sales_hist.groupby('year')['value'].sum().
 ↪reset_index()  # Sum sales per year
df_china_sales_hist['time'] = pd.to_datetime(df_china_sales_hist['year'],␣
 ↪format='%Y')
df_china_sales_hist.rename(columns={'value': 'sales'}, inplace=True)

df_china_sales_proj = df[(df['region'] == 'China') &
                         (df['parameter'] == 'EV sales') &
                         (df['year'] >= 2024) &
                         (df['year'] <= 2035)]
df_china_sales_proj = df_china_sales_proj.groupby('year')['value'].sum().
 ↪reset_index()  # Sum sales per year
```

```
df_china_sales_proj['time'] = pd.to_datetime(df_china_sales_proj['year'],␣
 ↪format='%Y')
df_china_sales_proj.rename(columns={'value': 'sales'}, inplace=True)

# Combine historical and projection data for training
df_sales_combined_china = pd.concat([df_china_sales_hist, df_china_sales_proj])

# Step 2: Initialize and fit the Prophet model with the aggregated data
sales_model_china = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.
 ↪05)
sales_model_china.fit(df_sales_combined_china.rename(columns={'time': 'ds',␣
 ↪'sales': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_sales_years_china = 2035 - 2023  # Years until 2035
future_sales_china = sales_model_china.
 ↪make_future_dataframe(periods=future_sales_years_china, freq='Y')
future_sales_china = future_sales_china[future_sales_china['ds'] <= pd.
 ↪to_datetime('2035-12-31')]  # Future data only until 2035

# Make predictions
forecast_sales_china = sales_model_china.predict(future_sales_china)
forecast_sales_china['yhat'] = forecast_sales_china['yhat'].clip(lower=0)  #␣
 ↪Clip negative predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot_sales_china = sales_model_china.plot(forecast_sales_china,␣
 ↪xlabel='Year', ylabel='EV Sales', ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
prediction_line_sales_china, = ax.plot(forecast_sales_china['ds'],␣
 ↪forecast_sales_china['yhat'], color='blue', label='Prediction Line (Solid)')

# Step 5: Overlay the projection data on the same plot
projected_data_sales_china = ax.scatter(df_china_sales_proj['time'],␣
 ↪df_china_sales_proj['sales'],
                                        color='red', marker='o', s=100,␣
 ↪label='Projected Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points_sales_china = ax.plot(forecast_sales_china['ds'],␣
 ↪forecast_sales_china['yhat'], 'k.', label='Forecast Data (Black Dots)',␣
 ↪alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
```

```
prediction_start_sales_china = ax.axvline(x=pd.to_datetime('2024-01-01'),␣
 ↪color='blue', linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Sales Forecast vs. Projection Data (China)', fontsize=18,␣
 ↪fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Sales', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')

# Format y-axis to display values in millions
def millions(x, pos):
    return f'{int(x / 1e6)}M'  # Convert to millions

formatter = FuncFormatter(millions)
plt.gca().yaxis.set_major_formatter(formatter)

# Step 8: Add the legend explicitly
legend_labels_sales_china = ['Prediction Line (Solid)', 'Dataset Value (Black␣
 ↪Dots)', 'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line_sales_china, forecast_points_sales_china[0],␣
 ↪prediction_start_sales_china, projected_data_sales_china],
          legend_labels_sales_china, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-01-31'))

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

# Step 9: Cross-Validation to assess model performance
# Perform cross-validation on the model
df_sales_cv_china = cross_validation(sales_model_china, initial='730 days',␣
 ↪period='365 days', horizon='365 days')

# Calculate performance metrics
df_sales_p_china = performance_metrics(df_sales_cv_china)
```

```
# Output performance metrics
print(df_sales_p_china[['horizon', 'mae', 'mape', 'rmse']])
```

```
19:52:33 - cmdstanpy - INFO - Chain [1] start processing
19:52:33 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
19:52:34 - cmdstanpy - INFO - Chain [1] start processing
19:52:34 - cmdstanpy - INFO - Chain [1] done processing
19:52:34 - cmdstanpy - INFO - Chain [1] start processing
19:52:34 - cmdstanpy - INFO - Chain [1] done processing
19:52:34 - cmdstanpy - INFO - Chain [1] start processing
19:52:34 - cmdstanpy - INFO - Chain [1] done processing
19:52:35 - cmdstanpy - INFO - Chain [1] start processing
19:52:35 - cmdstanpy - INFO - Chain [1] done processing
19:52:35 - cmdstanpy - INFO - Chain [1] start processing
```

```
19:52:35 - cmdstanpy - INFO - Chain [1] done processing
19:52:35 - cmdstanpy - INFO - Chain [1] start processing
19:52:35 - cmdstanpy - INFO - Chain [1] done processing
19:52:35 - cmdstanpy - INFO - Chain [1] start processing
19:52:35 - cmdstanpy - INFO - Chain [1] done processing
19:52:35 - cmdstanpy - INFO - Chain [1] start processing
19:52:35 - cmdstanpy - INFO - Chain [1] done processing
19:52:35 - cmdstanpy - INFO - Chain [1] start processing
19:52:35 - cmdstanpy - INFO - Chain [1] done processing
19:52:35 - cmdstanpy - INFO - Chain [1] start processing
19:52:35 - cmdstanpy - INFO - Chain [1] done processing
19:52:36 - cmdstanpy - INFO - Chain [1] start processing
19:52:36 - cmdstanpy - INFO - Chain [1] done processing
19:52:36 - cmdstanpy - INFO - Chain [1] start processing
19:52:36 - cmdstanpy - INFO - Chain [1] done processing
19:52:36 - cmdstanpy - INFO - Chain [1] start processing
19:52:36 - cmdstanpy - INFO - Chain [1] done processing
19:52:36 - cmdstanpy - INFO - Chain [1] start processing
19:52:36 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	1.221410e+05	0.529476	1.532320e+05
1	364 days	9.298367e+05	0.463864	1.312621e+06
2	365 days	1.083125e+07	0.488238	1.128797e+07

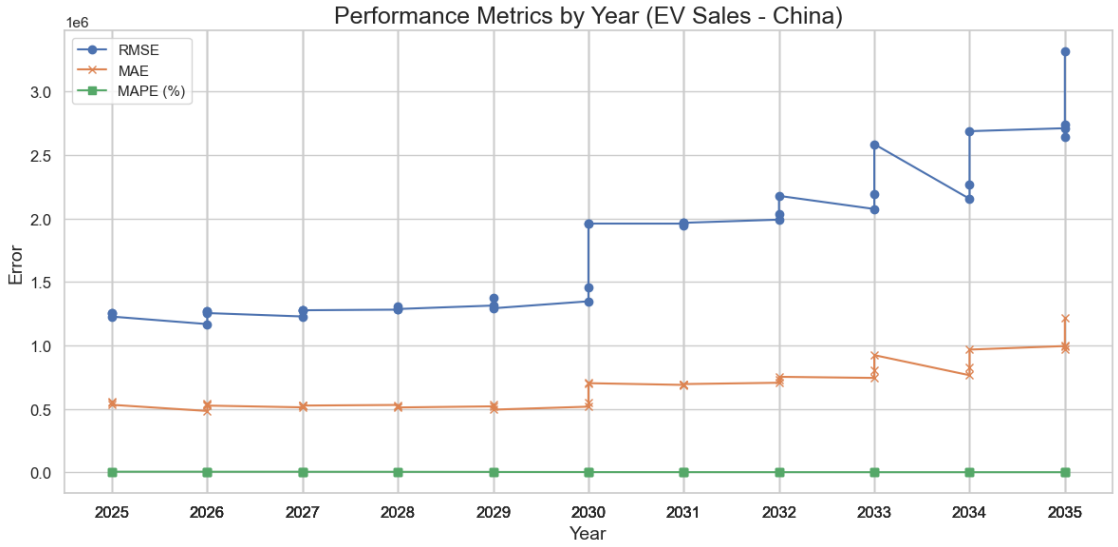
[ ]:

[6]:
```python
#china sales accuracy
# EV SALES Accuracy for China
from prophet.diagnostics import cross_validation, performance_metrics

# Sample Data Preparation (replace with your actual DataFrame)
# Assuming 'df' is your original DataFrame with columns: region, parameter, year, and value

# Filter for EV sales data
df_China_salesH = df[(df['region'] == 'China') &
                     (df['parameter'] == 'EV sales') &
                     (df['year'] < 2024)]  # Historical data

df_China_salesPro = df[(df['region'] == 'China') &
                       (df['parameter'] == 'EV sales') &
                       (df['year'] >= 2024) &
                       (df['year'] <= 2035)]  # Projection data

# Prepare the projection data for Prophet
df_China_salesPro['ds'] = pd.to_datetime(df_China_salesPro['year'], format='%Y')
df_China_salesPro['y'] = df_China_salesPro['value']
```

```
# Combine historical and projection data for training
df_combined_sales = pd.concat([df_China_salesH, df_China_salesPro])

df_combined_sales['ds'] = pd.to_datetime(df_combined_sales['year'], format='%Y')
df_combined_sales['y'] = df_combined_sales['value']

# Step 2: Initialize and fit the Prophet model
model_sales = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
model_sales.fit(df_combined_sales)

# Step 3: Cross-Validation to assess model performance with different horizons
# Calculate the total number of days from 2024 to 2035
horizon_days_sales = (pd.to_datetime('2035-01-01') - pd.
 ↪to_datetime('2024-01-01')).days

# Perform cross-validation with horizon set to the calculated days
df_cv_sales = cross_validation(model_sales, initial='730 days', period='365␣
 ↪days', horizon=f'{horizon_days_sales} days')

# Step 4: Calculate performance metrics
df_p_sales = performance_metrics(df_cv_sales)

# Generate year labels based on the prediction horizons
start_year_sales = 2024  # Starting year for predictions

# Convert horizons from days to years
years_sales = [start_year_sales + (horizon.days // 365) for horizon in␣
 ↪df_p_sales['horizon']]

# Output performance metrics
print(df_p_sales[['horizon', 'mae', 'mape', 'rmse']])

# Optional: Plot the performance metrics with years on the x-axis
plt.figure(figsize=(12, 6))
plt.plot(years_sales, df_p_sales['rmse'], label='RMSE', marker='o')
plt.plot(years_sales, df_p_sales['mae'], label='MAE', marker='x')
plt.plot(years_sales, df_p_sales['mape'], label='MAPE (%)', marker='s')
plt.legend()
plt.title('Performance Metrics by Year (EV Sales - China)', fontsize=18)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Error', fontsize=14)
plt.xticks(years_sales, fontsize=12)  # Set x-ticks to show years
plt.yticks(fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_9000\1837122651.py:23:
SettingWithCopyWarning:
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
  df_China_salesPro['ds'] = pd.to_datetime(df_China_salesPro['year'],
format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_9000\1837122651.py:24:
SettingWithCopyWarning:
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
  df_China_salesPro['y'] = df_China_salesPro['value']
18:50:01 - cmdstanpy - INFO - Chain [1] start processing
18:50:01 - cmdstanpy - INFO - Chain [1] done processing
  0%|          | 0/13 [00:00<?, ?it/s]
18:50:01 - cmdstanpy - INFO - Chain [1] start processing
18:50:01 - cmdstanpy - INFO - Chain [1] done processing
18:50:02 - cmdstanpy - INFO - Chain [1] start processing
18:50:02 - cmdstanpy - INFO - Chain [1] done processing
18:50:02 - cmdstanpy - INFO - Chain [1] start processing
18:50:02 - cmdstanpy - INFO - Chain [1] done processing
18:50:02 - cmdstanpy - INFO - Chain [1] start processing
18:50:02 - cmdstanpy - INFO - Chain [1] done processing
18:50:03 - cmdstanpy - INFO - Chain [1] start processing
18:50:03 - cmdstanpy - INFO - Chain [1] done processing
18:50:03 - cmdstanpy - INFO - Chain [1] start processing
18:50:03 - cmdstanpy - INFO - Chain [1] done processing
18:50:03 - cmdstanpy - INFO - Chain [1] start processing
18:50:04 - cmdstanpy - INFO - Chain [1] done processing
18:50:04 - cmdstanpy - INFO - Chain [1] start processing
18:50:04 - cmdstanpy - INFO - Chain [1] done processing
18:50:04 - cmdstanpy - INFO - Chain [1] start processing
18:50:04 - cmdstanpy - INFO - Chain [1] done processing
18:50:04 - cmdstanpy - INFO - Chain [1] start processing
18:50:05 - cmdstanpy - INFO - Chain [1] done processing
18:50:05 - cmdstanpy - INFO - Chain [1] start processing
18:50:05 - cmdstanpy - INFO - Chain [1] done processing
18:50:05 - cmdstanpy - INFO - Chain [1] start processing
18:50:05 - cmdstanpy - INFO - Chain [1] done processing
18:50:05 - cmdstanpy - INFO - Chain [1] start processing
18:50:05 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	366 days	5.575007e+05	3666.955087	1.256923e+06
1	728 days	5.519103e+05	3263.407505	1.255834e+06
2	729 days	5.308326e+05	3912.387446	1.225979e+06
3	730 days	4.823086e+05	4017.683342	1.166972e+06
4	731 days	5.386150e+05	4027.969343	1.267842e+06
5	1093 days	5.367940e+05	3823.440026	1.267407e+06
6	1094 days	5.247952e+05	3555.101344	1.253882e+06
7	1095 days	5.110373e+05	3686.100576	1.227047e+06
8	1096 days	5.263998e+05	3685.495772	1.275532e+06
9	1458 days	5.268758e+05	3685.495724	1.275585e+06
10	1459 days	5.249919e+05	3321.834139	1.276005e+06
11	1460 days	5.298582e+05	3074.843268	1.280825e+06
12	1461 days	5.217658e+05	3043.630278	1.306138e+06
13	1824 days	5.101631e+05	2819.021315	1.285903e+06
14	1825 days	5.188921e+05	2242.574328	1.313123e+06
15	1826 days	5.343617e+05	2161.235857	1.376501e+06
16	2189 days	4.929536e+05	1663.510225	1.291717e+06
17	2190 days	5.163243e+05	1234.057030	1.346514e+06
18	2191 days	5.518952e+05	1139.415953	1.455228e+06
19	2192 days	7.055482e+05	1048.865145	1.961339e+06
20	2554 days	7.014845e+05	786.805887	1.959210e+06
21	2555 days	6.877903e+05	485.029049	1.958734e+06
22	2556 days	6.847605e+05	485.896112	1.941869e+06
23	2557 days	6.941369e+05	481.989680	1.964942e+06
24	2919 days	6.943678e+05	485.294787	1.965106e+06
25	2920 days	7.048322e+05	300.796700	1.990066e+06
26	2921 days	7.256619e+05	280.642720	2.036467e+06
27	2922 days	7.515090e+05	278.707984	2.177295e+06
28	3285 days	7.428221e+05	129.613460	2.073909e+06
29	3286 days	8.011047e+05	113.378958	2.190227e+06
30	3287 days	9.223374e+05	103.248631	2.584892e+06
31	3650 days	7.640211e+05	66.448211	2.153951e+06
32	3651 days	8.230831e+05	56.901307	2.266563e+06
33	3652 days	9.661479e+05	48.631005	2.687016e+06
34	4015 days	9.939587e+05	31.119337	2.710470e+06
35	4016 days	9.702001e+05	26.067573	2.641376e+06
36	4017 days	1.001052e+06	26.827920	2.736056e+06
37	4018 days	1.217510e+06	36.587948	3.313556e+06



Europe future market

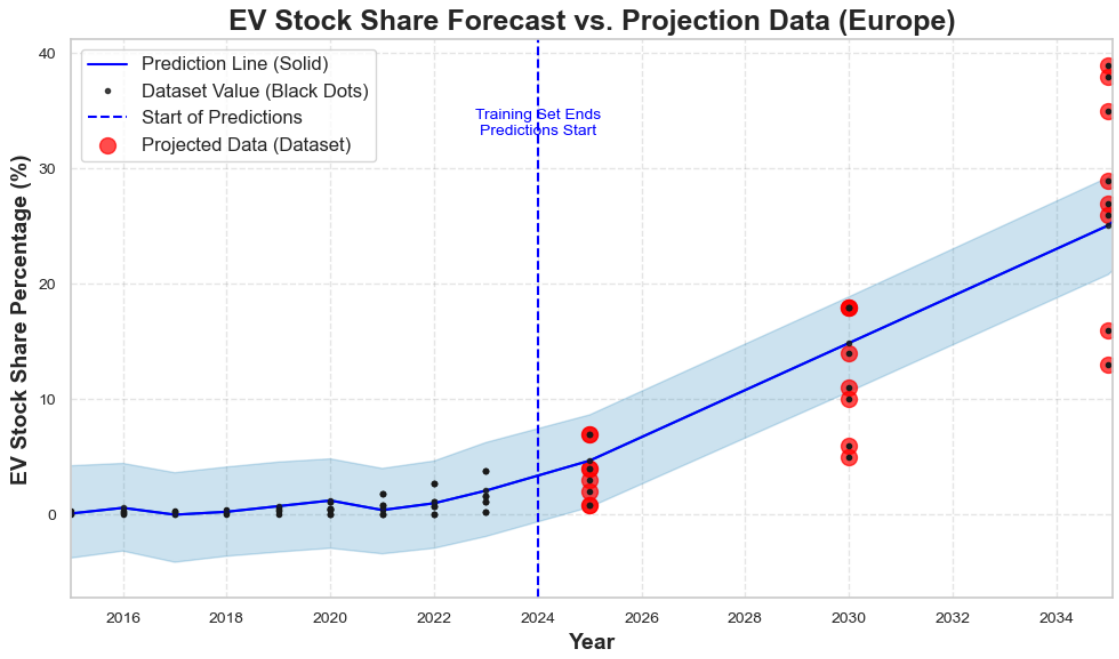
[29]:
```
from prophet.diagnostics import cross_validation, performance_metrics
from prophet import Prophet

# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV STOCK SHARE Data Preparation for Europe
df_europe_stock_share_hist = df[(df['region'] == 'Europe') &
                                (df['parameter'] == 'EV stock share') &
                                (df['year'] < 2024)]  # Historical data␣
 ↪(before 2024)

df_europe_stock_share_proj = df[(df['region'] == 'Europe') &
                                (df['parameter'] == 'EV stock share') &
                                (df['year'] >= 2024) &
                                (df['year'] <= 2035)]  # Projection data (2024␣
 ↪to 2035)

# Prepare the projection data for plotting
df_europe_stock_share_proj['time'] = pd.
 ↪to_datetime(df_europe_stock_share_proj['year'], format='%Y')
df_europe_stock_share_proj['percentage (%)'] =␣
 ↪df_europe_stock_share_proj['value']

# Combine historical and projection data for training
df_stock_share_combined_europe = pd.concat([df_europe_stock_share_hist,␣
 ↪df_europe_stock_share_proj])
```

```
df_stock_share_combined_europe['time'] = pd.
 ↪to_datetime(df_stock_share_combined_europe['year'], format='%Y')
df_stock_share_combined_europe['percentage (%)'] =␣
 ↪df_stock_share_combined_europe['value']

# Step 2: Initialize and fit the Prophet model with yearly seasonality and␣
 ↪changepoint sensitivity
stock_share_model_europe = Prophet(yearly_seasonality=True,  # Enable yearly␣
 ↪seasonality
                                   changepoint_prior_scale=0.05)  # Adjust␣
 ↪changepoint sensitivity
stock_share_model_europe.fit(df_stock_share_combined_europe.
 ↪rename(columns={'time': 'ds', 'percentage (%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_stock_share_years_europe = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_stock_share_europe = stock_share_model_europe.
 ↪make_future_dataframe(periods=future_stock_share_years_europe, freq='Y')

# Restrict future data to end at 2035
future_stock_share_europe =␣
 ↪future_stock_share_europe[future_stock_share_europe['ds'] <= pd.
 ↪to_datetime('2035-12-31')]  # Future data only until 2035

# Make predictions
forecast_stock_share_europe = stock_share_model_europe.
 ↪predict(future_stock_share_europe)

# Set a lower bound for predictions
forecast_stock_share_europe['yhat'] = forecast_stock_share_europe['yhat'].
 ↪clip(lower=0)  # Clip negative predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot_share_europe = stock_share_model_europe.
 ↪plot(forecast_stock_share_europe, xlabel='Year', ylabel='Percentage (%)',␣
 ↪ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
prediction_line_share_europe, = ax.plot(forecast_stock_share_europe['ds'],␣
 ↪forecast_stock_share_europe['yhat'], color='blue', label='Prediction Line␣
 ↪(Solid)')
```

```
# Step 5: Overlay the projection data on the same plot
projected_data_share_europe = ax.scatter(df_europe_stock_share_proj['time'],␣
 ↪df_europe_stock_share_proj['percentage (%)'],
                                          color='red', marker='o', s=100,␣
 ↪label='Projected Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points_share_europe = ax.plot(forecast_stock_share_europe['ds'],␣
 ↪forecast_stock_share_europe['yhat'], 'k.', label='Forecast Data (Black␣
 ↪Dots)', alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
prediction_start_share_europe = ax.axvline(x=pd.to_datetime('2024-01-01'),␣
 ↪color='blue', linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Stock Share Forecast vs. Projection Data (Europe)',␣
 ↪fontsize=18, fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Stock Share Percentage (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')

# Step 8: Add the legend explicitly
legend_labels_share_europe = ['Prediction Line (Solid)', 'Dataset Value (Black␣
 ↪Dots)', 'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line_share_europe, forecast_points_share_europe[0],␣
 ↪prediction_start_share_europe, projected_data_share_europe],
          legend_labels_share_europe, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-01-31'))

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

# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
```

```
df_stock_share_cv_europe = cross_validation(stock_share_model_europe,␣
 ↪initial='730 days', period='365 days', horizon='365 days')

# Calculate performance metrics
df_stock_share_p_europe = performance_metrics(df_stock_share_cv_europe)

# Output performance metrics
print(df_stock_share_p_europe[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\4243811112.py:21:
SettingWithCopyWarning:
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
  df_europe_stock_share_proj['time'] =
pd.to_datetime(df_europe_stock_share_proj['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\4243811112.py:22:
SettingWithCopyWarning:
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
  df_europe_stock_share_proj['percentage (%)'] =
df_europe_stock_share_proj['value']
19:31:58 - cmdstanpy - INFO - Chain [1] start processing
19:31:58 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
19:32:00 - cmdstanpy - INFO - Chain [1] start processing
19:32:00 - cmdstanpy - INFO - Chain [1] done processing
19:32:00 - cmdstanpy - INFO - Chain [1] start processing
19:32:00 - cmdstanpy - INFO - Chain [1] done processing
19:32:00 - cmdstanpy - INFO - Chain [1] start processing
19:32:00 - cmdstanpy - INFO - Chain [1] done processing
19:32:00 - cmdstanpy - INFO - Chain [1] start processing
19:32:00 - cmdstanpy - INFO - Chain [1] done processing
19:32:01 - cmdstanpy - INFO - Chain [1] start processing
19:32:01 - cmdstanpy - INFO - Chain [1] done processing
19:32:01 - cmdstanpy - INFO - Chain [1] start processing
19:32:01 - cmdstanpy - INFO - Chain [1] done processing
19:32:01 - cmdstanpy - INFO - Chain [1] start processing
19:32:01 - cmdstanpy - INFO - Chain [1] done processing
19:32:01 - cmdstanpy - INFO - Chain [1] start processing
19:32:02 - cmdstanpy - INFO - Chain [1] done processing
19:32:02 - cmdstanpy - INFO - Chain [1] start processing
19:32:02 - cmdstanpy - INFO - Chain [1] done processing
19:32:02 - cmdstanpy - INFO - Chain [1] start processing
19:32:02 - cmdstanpy - INFO - Chain [1] done processing
19:32:02 - cmdstanpy - INFO - Chain [1] start processing
19:32:02 - cmdstanpy - INFO - Chain [1] done processing
19:32:03 - cmdstanpy - INFO - Chain [1] start processing
19:32:03 - cmdstanpy - INFO - Chain [1] done processing
```

```
19:32:03 - cmdstanpy - INFO - Chain [1] start processing
19:32:03 - cmdstanpy - INFO - Chain [1] done processing
19:32:03 - cmdstanpy - INFO - Chain [1] start processing
19:32:03 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	0.101678	3.336986	0.125088
1	364 days	0.265149	4.997749	0.344887
2	365 days	3.746373	1.702393	6.678545

[30]:
```python
from prophet.diagnostics import cross_validation, performance_metrics

# Filter Historical and Projection Data
df_europe_sasrH = df[(df['region'] == 'Europe') &
                     (df['parameter'] == 'EV sales share') &
                     (df['year'] < 2024)]  # Historical data

df_europe_sasrPro = df[(df['region'] == 'Europe') &
                       (df['parameter'] == 'EV sales share') &
                       (df['year'] >= 2024) &
                       (df['year'] <= 2035)]  # Projection data

# Prepare the combined data
df_combined_europe = pd.concat([df_europe_sasrH, df_europe_sasrPro])
df_combined_europe['ds'] = pd.to_datetime(df_combined_europe['year'],␣
 ↪format='%Y')
df_combined_europe['y'] = df_combined_europe['value']

# Fit the Prophet Model
model_europe = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
model_europe.fit(df_combined_europe)

# Cross-Validation and Performance Metrics
horizon_days_europe = (pd.to_datetime('2035-01-01') - pd.
 ↪to_datetime('2024-01-01')).days
df_cv_europe = cross_validation(model_europe, initial='730 days', period='365␣
 ↪days', horizon=f'{horizon_days_europe} days')
df_p_europe = performance_metrics(df_cv_europe)

# Prepare Year Labels for Horizon
start_year_europe = 2024  # Starting year for predictions
years_europe = [start_year_europe + (horizon.days // 365) for horizon in␣
 ↪df_p_europe['horizon']]

# Print Metrics Summary
print(df_p_europe[['horizon', 'mae', 'mape', 'rmse']])

# Plot Performance Metrics
```

```
plt.figure(figsize=(12, 6))
plt.plot(years_europe, df_p_europe['rmse'], label='RMSE', marker='o')
plt.plot(years_europe, df_p_europe['mae'], label='MAE', marker='x')
plt.plot(years_europe, df_p_europe['mape'], label='MAPE (%)', marker='s')
plt.title('Sales Share Performance Metrics by Year (Europe)', fontsize=18,␣
 ↪fontweight='bold')
plt.xlabel('Year', fontsize=14, fontweight='bold')
plt.ylabel('Error', fontsize=14, fontweight='bold')
plt.xticks(years_europe, fontsize=12)  # Set x-ticks to show years
plt.yticks(fontsize=12)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--')
plt.tight_layout()
plt.show()
```

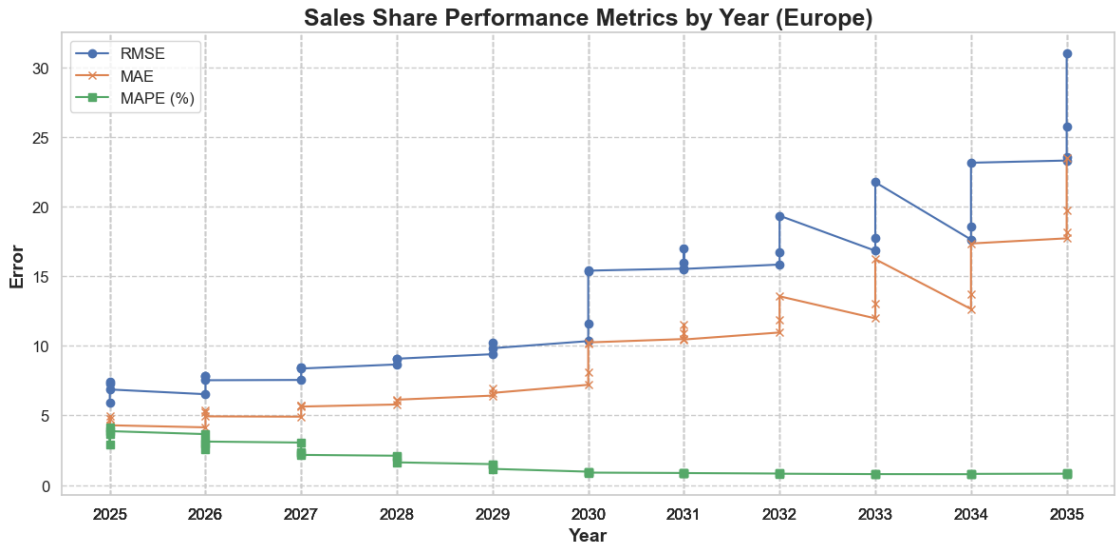
```
19:34:06 - cmdstanpy - INFO - Chain [1] start processing
19:34:06 - cmdstanpy - INFO - Chain [1] done processing
```

```
  0%|          | 0/13 [00:00<?, ?it/s]
```

```
19:34:06 - cmdstanpy - INFO - Chain [1] start processing
19:34:06 - cmdstanpy - INFO - Chain [1] done processing
19:34:07 - cmdstanpy - INFO - Chain [1] start processing
19:34:07 - cmdstanpy - INFO - Chain [1] done processing
19:34:07 - cmdstanpy - INFO - Chain [1] start processing
19:34:07 - cmdstanpy - INFO - Chain [1] done processing
19:34:07 - cmdstanpy - INFO - Chain [1] start processing
19:34:07 - cmdstanpy - INFO - Chain [1] done processing
19:34:08 - cmdstanpy - INFO - Chain [1] start processing
19:34:08 - cmdstanpy - INFO - Chain [1] done processing
19:34:08 - cmdstanpy - INFO - Chain [1] start processing
19:34:08 - cmdstanpy - INFO - Chain [1] done processing
19:34:08 - cmdstanpy - INFO - Chain [1] start processing
19:34:08 - cmdstanpy - INFO - Chain [1] done processing
19:34:08 - cmdstanpy - INFO - Chain [1] start processing
19:34:09 - cmdstanpy - INFO - Chain [1] done processing
19:34:09 - cmdstanpy - INFO - Chain [1] start processing
19:34:09 - cmdstanpy - INFO - Chain [1] done processing
19:34:09 - cmdstanpy - INFO - Chain [1] start processing
19:34:09 - cmdstanpy - INFO - Chain [1] done processing
19:34:09 - cmdstanpy - INFO - Chain [1] start processing
19:34:10 - cmdstanpy - INFO - Chain [1] done processing
19:34:10 - cmdstanpy - INFO - Chain [1] start processing
19:34:10 - cmdstanpy - INFO - Chain [1] done processing
19:34:10 - cmdstanpy - INFO - Chain [1] start processing
19:34:10 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	365 days	3.745061	4.145475	5.947974

1	366	days	4.958956	2.930296	7.394949
2	728	days	4.742816	3.653367	7.302810
3	729	days	4.296021	3.874642	6.869679
4	730	days	4.147307	3.660627	6.529935
5	731	days	5.354894	2.545392	7.870326
6	1093	days	5.233562	3.007098	7.817503
7	1094	days	4.937388	3.125941	7.534217
8	1095	days	4.913677	3.054293	7.555494
9	1096	days	5.723116	2.349191	8.446156
10	1458	days	5.701638	2.270415	8.436086
11	1459	days	5.641283	2.167079	8.374754
12	1460	days	5.791132	2.108228	8.670402
13	1461	days	6.040862	2.008766	9.043981
14	1824	days	6.135200	1.632223	9.082426
15	1825	days	6.429017	1.502891	9.406840
16	1826	days	6.937244	1.500833	10.244115
17	2189	days	6.622317	1.161172	9.837581
18	2190	days	7.205099	0.961797	10.346812
19	2191	days	8.109470	0.958173	11.584261
20	2192	days	10.133392	0.944091	15.377116
21	2554	days	10.251787	0.893534	15.410498
22	2555	days	10.495355	0.865943	15.558992
23	2556	days	10.925607	0.868602	15.975920
24	2557	days	11.553305	0.861195	17.025026
25	2919	days	10.462685	0.861037	15.536045
26	2920	days	10.965858	0.826253	15.850041
27	2921	days	11.910438	0.829848	16.732116
28	2922	days	13.567163	0.814533	19.360187
29	3285	days	11.982446	0.785875	16.840441
30	3286	days	13.059620	0.793295	17.790303
31	3287	days	16.236379	0.786058	21.765803
32	3650	days	12.653272	0.779922	17.647646
33	3651	days	13.737455	0.793530	18.565558
34	3652	days	17.363650	0.802460	23.168551
35	4015	days	17.735710	0.815355	23.327741
36	4016	days	18.163048	0.822145	23.608954
37	4017	days	19.721631	0.835696	25.789320
38	4018	days	23.536524	0.829052	31.009197



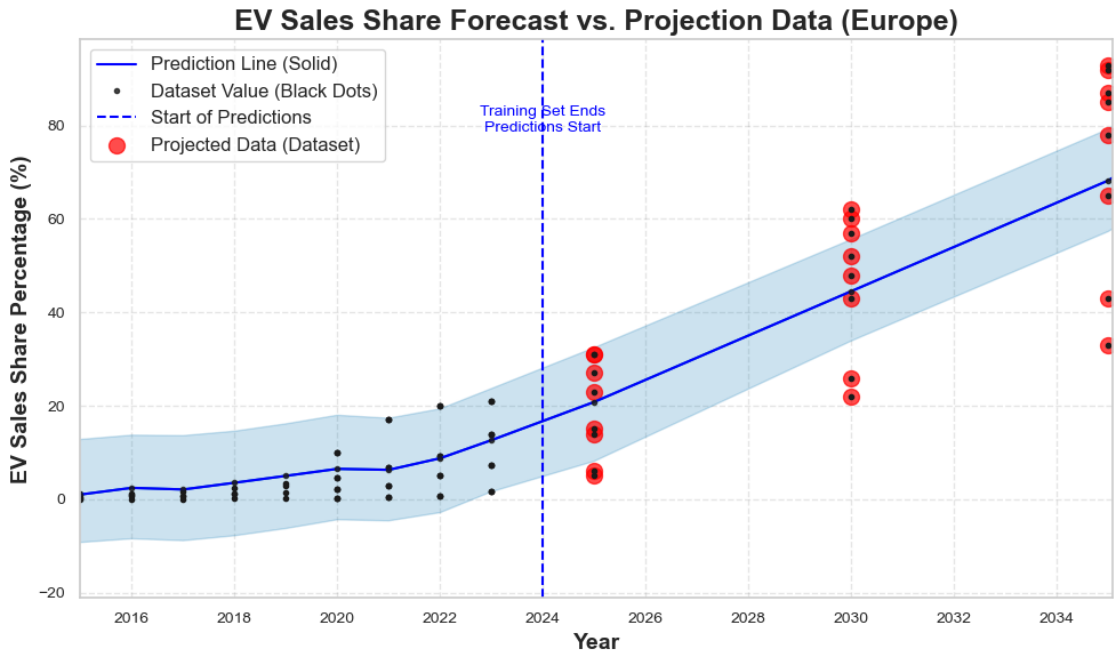
[28]:
```
#europe
from prophet.diagnostics import cross_validation, performance_metrics
from prophet import Prophet

# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV SALES SHARE Data Preparation for Europe (Updated variable names)
df_europe_sales_share_hist = df[(df['region'] == 'Europe') &
                                (df['parameter'] == 'EV sales share') &
                                (df['year'] < 2024)]  # Historical data␣
 ↪(before 2024)

df_europe_sales_share_proj = df[(df['region'] == 'Europe') &
                                (df['parameter'] == 'EV sales share') &
                                (df['year'] >= 2024) &
                                (df['year'] <= 2035)]  # Projection data (2024␣
 ↪to 2035)

# Prepare the projection data for plotting
df_europe_sales_share_proj['time'] = pd.
 ↪to_datetime(df_europe_sales_share_proj['year'], format='%Y')
df_europe_sales_share_proj['percentage (%)'] =␣
 ↪df_europe_sales_share_proj['value']

# Combine historical and projection data for training
df_sales_share_combined_europe = pd.concat([df_europe_sales_share_hist,␣
 ↪df_europe_sales_share_proj])
```

```
df_sales_share_combined_europe['time'] = pd.
 ↪to_datetime(df_sales_share_combined_europe['year'], format='%Y')
df_sales_share_combined_europe['percentage (%)'] =␣
 ↪df_sales_share_combined_europe['value']

# Step 2: Initialize and fit the Prophet model with yearly seasonality and␣
 ↪changepoint sensitivity
sales_share_model_europe = Prophet(yearly_seasonality=True,  # Enable yearly␣
 ↪seasonality
                                   changepoint_prior_scale=0.05)  # Adjust␣
 ↪changepoint sensitivity
sales_share_model_europe.fit(df_sales_share_combined_europe.
 ↪rename(columns={'time': 'ds', 'percentage (%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_sales_share_years_europe = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_sales_share_europe = sales_share_model_europe.
 ↪make_future_dataframe(periods=future_sales_share_years_europe, freq='Y')

# Restrict future data to end at 2035
future_sales_share_europe =␣
 ↪future_sales_share_europe[future_sales_share_europe['ds'] <= pd.
 ↪to_datetime('2035-12-31')]  # Future data only until 2035

# Make predictions
forecast_sales_share_europe = sales_share_model_europe.
 ↪predict(future_sales_share_europe)

# Set a lower bound for predictions
forecast_sales_share_europe['yhat'] = forecast_sales_share_europe['yhat'].
 ↪clip(lower=0)  # Clip negative predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot_share_europe = sales_share_model_europe.
 ↪plot(forecast_sales_share_europe, xlabel='Year', ylabel='Percentage (%)',␣
 ↪ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
prediction_line_share_europe, = ax.plot(forecast_sales_share_europe['ds'],␣
 ↪forecast_sales_share_europe['yhat'], color='blue', label='Prediction Line␣
 ↪(Solid)')
```

```
# Step 5: Overlay the projection data on the same plot
projected_data_share_europe = ax.scatter(df_europe_sales_share_proj['time'],␣
 ↪df_europe_sales_share_proj['percentage (%)'],
                                          color='red', marker='o', s=100,␣
 ↪label='Projected Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points_share_europe = ax.plot(forecast_sales_share_europe['ds'],␣
 ↪forecast_sales_share_europe['yhat'], 'k.', label='Forecast Data (Black␣
 ↪Dots)', alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
prediction_start_share_europe = ax.axvline(x=pd.to_datetime('2024-01-01'),␣
 ↪color='blue', linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Sales Share Forecast vs. Projection Data (Europe)',␣
 ↪fontsize=18, fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Sales Share Percentage (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')

# Step 8: Add the legend explicitly
legend_labels_share_europe = ['Prediction Line (Solid)', 'Dataset Value (Black␣
 ↪Dots)', 'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line_share_europe, forecast_points_share_europe[0],␣
 ↪prediction_start_share_europe, projected_data_share_europe],
          legend_labels_share_europe, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-01-31'))

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

# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
```

```
df_sales_share_cv_europe = cross_validation(sales_share_model_europe,␣
 ↪initial='730 days', period='365 days', horizon='365 days')

# Calculate performance metrics
df_sales_share_p_europe = performance_metrics(df_sales_share_cv_europe)

# Output performance metrics
print(df_sales_share_p_europe[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\2796719042.py:22:
SettingWithCopyWarning:
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
  df_europe_sales_share_proj['time'] =
pd.to_datetime(df_europe_sales_share_proj['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\2796719042.py:23:
SettingWithCopyWarning:
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
  df_europe_sales_share_proj['percentage (%)'] =
df_europe_sales_share_proj['value']
19:29:51 - cmdstanpy - INFO - Chain [1] start processing
19:29:52 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
19:29:53 - cmdstanpy - INFO - Chain [1] start processing
19:29:53 - cmdstanpy - INFO - Chain [1] done processing
19:29:53 - cmdstanpy - INFO - Chain [1] start processing
19:29:53 - cmdstanpy - INFO - Chain [1] done processing
19:29:53 - cmdstanpy - INFO - Chain [1] start processing
19:29:53 - cmdstanpy - INFO - Chain [1] done processing
19:29:53 - cmdstanpy - INFO - Chain [1] start processing
19:29:54 - cmdstanpy - INFO - Chain [1] done processing
19:29:54 - cmdstanpy - INFO - Chain [1] start processing
19:29:54 - cmdstanpy - INFO - Chain [1] done processing
19:29:54 - cmdstanpy - INFO - Chain [1] start processing
19:29:54 - cmdstanpy - INFO - Chain [1] done processing
19:29:54 - cmdstanpy - INFO - Chain [1] start processing
19:29:54 - cmdstanpy - INFO - Chain [1] done processing
19:29:55 - cmdstanpy - INFO - Chain [1] start processing
19:29:55 - cmdstanpy - INFO - Chain [1] done processing
19:29:55 - cmdstanpy - INFO - Chain [1] start processing
19:29:55 - cmdstanpy - INFO - Chain [1] done processing
19:29:55 - cmdstanpy - INFO - Chain [1] start processing
19:29:55 - cmdstanpy - INFO - Chain [1] done processing
19:29:55 - cmdstanpy - INFO - Chain [1] start processing
19:29:56 - cmdstanpy - INFO - Chain [1] done processing
19:29:56 - cmdstanpy - INFO - Chain [1] start processing
19:29:56 - cmdstanpy - INFO - Chain [1] done processing
```

```
19:29:56 - cmdstanpy - INFO - Chain [1] start processing
19:29:56 - cmdstanpy - INFO - Chain [1] done processing
19:29:57 - cmdstanpy - INFO - Chain [1] start processing
19:29:57 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	0.299046	11.392850	0.364544
1	364 days	2.078803	4.505954	3.297611
2	365 days	11.862849	1.217013	16.566749

[ ]:

[27]:
```
# Europe sales share accuracy
# SALES SHARES Accuracy
from prophet.diagnostics import cross_validation, performance_metrics
df_europe_sasrH = df[(df['region'] == 'Europe') &
                     (df['parameter'] == 'EV sales share') &
                     (df['year'] < 2024)]  # Historical data

df_europe_sasrPro = df[(df['region'] == 'Europe') &
                       (df['parameter'] == 'EV sales share') &
                       (df['year'] >= 2024) &
                       (df['year'] <= 2035)]  # Projection data

# Prepare the projection data for plotting
df_europe_sasrPro['ds'] = pd.to_datetime(df_europe_sasrPro['year'], format='%Y')
df_europe_sasrPro['y'] = df_europe_sasrPro['value']

df_combined_europe = pd.concat([df_europe_sasrH, df_europe_sasrPro])

df_combined_europe['ds'] = pd.to_datetime(df_combined_europe['year'],␣
 ↪format='%Y')
df_combined_europe['y'] = df_combined_europe['value']

model_europe = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
model_europe.fit(df_combined_europe)

horizon_days_europe = (pd.to_datetime('2035-01-01') - pd.
 ↪to_datetime('2024-01-01')).days

df_cv_europe = cross_validation(model_europe, initial='730 days', period='365␣
 ↪days', horizon=f'{horizon_days_europe} days')

df_p_europe = performance_metrics(df_cv_europe)

start_year_europe = 2024  # Starting year for predictions
```

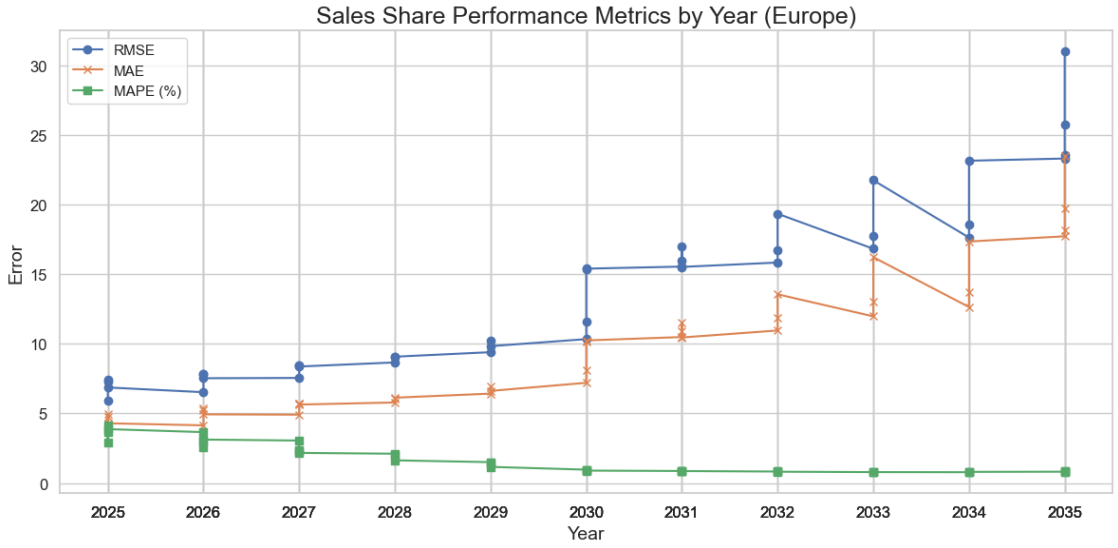
```
years_europe = [start_year_europe + (horizon.days // 365) for horizon in␣
 ↪df_p_europe['horizon']]

print(df_p_europe[['horizon', 'mae', 'mape', 'rmse']])
plt.figure(figsize=(12, 6))
plt.plot(years_europe, df_p_europe['rmse'], label='RMSE', marker='o')
plt.plot(years_europe, df_p_europe['mae'], label='MAE', marker='x')
plt.plot(years_europe, df_p_europe['mape'], label='MAPE (%)', marker='s')
plt.title('Sales Share Performance Metrics by Year (Europe)', fontsize=18)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Error', fontsize=14)
plt.xticks(years_europe, fontsize=12)  # Set x-ticks to show years
plt.yticks(fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1448346889.py:18:
SettingWithCopyWarning:
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
  df_europe_sasrPro['ds'] = pd.to_datetime(df_europe_sasrPro['year'],
format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1448346889.py:19:
SettingWithCopyWarning:
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
  df_europe_sasrPro['y'] = df_europe_sasrPro['value']
19:29:11 - cmdstanpy - INFO - Chain [1] start processing
19:29:11 - cmdstanpy - INFO - Chain [1] done processing
  0%|          | 0/13 [00:00<?, ?it/s]
19:29:11 - cmdstanpy - INFO - Chain [1] start processing
19:29:11 - cmdstanpy - INFO - Chain [1] done processing
19:29:11 - cmdstanpy - INFO - Chain [1] start processing
19:29:11 - cmdstanpy - INFO - Chain [1] done processing
19:29:11 - cmdstanpy - INFO - Chain [1] start processing
19:29:11 - cmdstanpy - INFO - Chain [1] done processing
19:29:12 - cmdstanpy - INFO - Chain [1] start processing
19:29:12 - cmdstanpy - INFO - Chain [1] done processing
19:29:12 - cmdstanpy - INFO - Chain [1] start processing
```

```
19:29:12 - cmdstanpy - INFO - Chain [1] done processing
19:29:12 - cmdstanpy - INFO - Chain [1] start processing
19:29:13 - cmdstanpy - INFO - Chain [1] done processing
19:29:13 - cmdstanpy - INFO - Chain [1] start processing
19:29:13 - cmdstanpy - INFO - Chain [1] done processing
19:29:13 - cmdstanpy - INFO - Chain [1] start processing
19:29:13 - cmdstanpy - INFO - Chain [1] done processing
19:29:13 - cmdstanpy - INFO - Chain [1] start processing
19:29:14 - cmdstanpy - INFO - Chain [1] done processing
19:29:14 - cmdstanpy - INFO - Chain [1] start processing
19:29:14 - cmdstanpy - INFO - Chain [1] done processing
19:29:14 - cmdstanpy - INFO - Chain [1] start processing
19:29:14 - cmdstanpy - INFO - Chain [1] done processing
19:29:14 - cmdstanpy - INFO - Chain [1] start processing
19:29:15 - cmdstanpy - INFO - Chain [1] done processing
19:29:15 - cmdstanpy - INFO - Chain [1] start processing
19:29:15 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	365 days	3.745061	4.145475	5.947974
1	366 days	4.958956	2.930296	7.394949
2	728 days	4.742816	3.653367	7.302810
3	729 days	4.296021	3.874642	6.869679
4	730 days	4.147307	3.660627	6.529935
5	731 days	5.354894	2.545392	7.870326
6	1093 days	5.233562	3.007098	7.817503
7	1094 days	4.937388	3.125941	7.534217
8	1095 days	4.913677	3.054293	7.555494
9	1096 days	5.723116	2.349191	8.446156
10	1458 days	5.701638	2.270415	8.436086
11	1459 days	5.641283	2.167079	8.374754
12	1460 days	5.791132	2.108228	8.670402
13	1461 days	6.040862	2.008766	9.043981
14	1824 days	6.135200	1.632223	9.082426
15	1825 days	6.429017	1.502891	9.406840
16	1826 days	6.937244	1.500833	10.244115
17	2189 days	6.622317	1.161172	9.837581
18	2190 days	7.205099	0.961797	10.346812
19	2191 days	8.109470	0.958173	11.584261
20	2192 days	10.133392	0.944091	15.377116
21	2554 days	10.251787	0.893534	15.410498
22	2555 days	10.495355	0.865943	15.558992
23	2556 days	10.925607	0.868602	15.975920
24	2557 days	11.553305	0.861195	17.025026
25	2919 days	10.462685	0.861037	15.536045
26	2920 days	10.965858	0.826253	15.850041
27	2921 days	11.910438	0.829848	16.732116
28	2922 days	13.567163	0.814533	19.360187

```
29 3285 days  11.982446  0.785875  16.840441
30 3286 days  13.059620  0.793295  17.790303
31 3287 days  16.236379  0.786058  21.765803
32 3650 days  12.653272  0.779922  17.647646
33 3651 days  13.737455  0.793530  18.565558
34 3652 days  17.363650  0.802460  23.168551
35 4015 days  17.735710  0.815355  23.327741
36 4016 days  18.163048  0.822145  23.608954
37 4017 days  19.721631  0.835696  25.789320
38 4018 days  23.536524  0.829052  31.009197
```



USA future Market

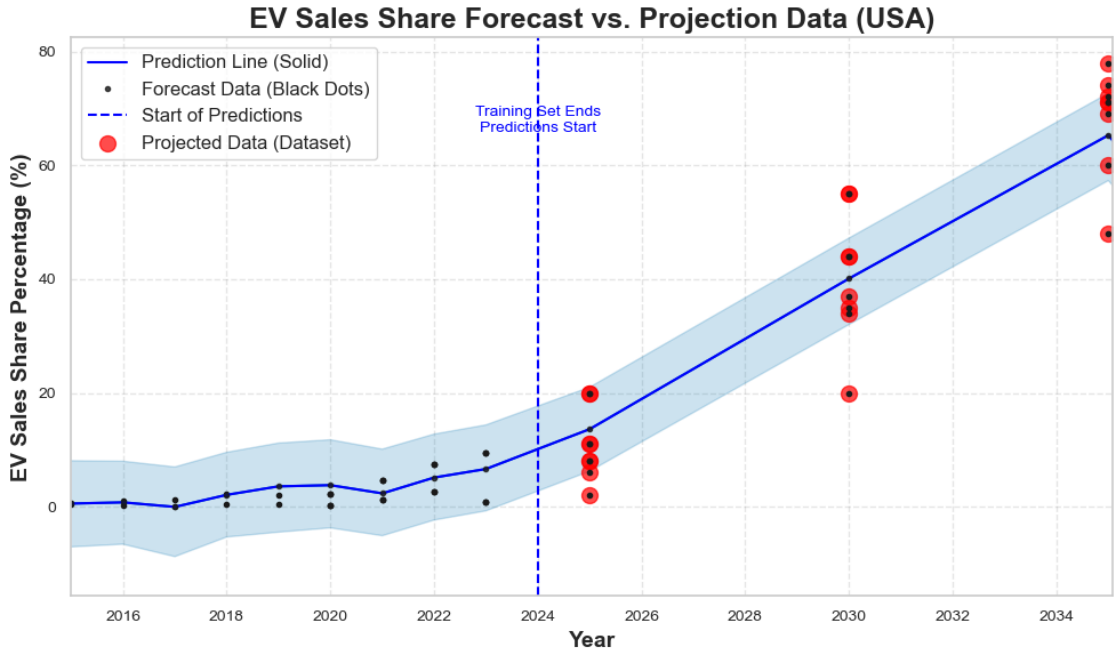
[24]:
```
sns.set(style="whitegrid")

df_usa_sales_share_hist = df[(df['region'] == 'USA') &
                             (df['parameter'] == 'EV sales share') &
                             (df['year'] < 2024)]  # Historical data (before␣
 ↪2024)

df_usa_sales_share_proj = df[(df['region'] == 'USA') &
                             (df['parameter'] == 'EV sales share') &
                             (df['year'] >= 2024) &
                             (df['year'] <= 2035)]  # Projection data (2024 to␣
 ↪2035)

# Prepare the projection data for plotting
df_usa_sales_share_proj['time'] = pd.
 ↪to_datetime(df_usa_sales_share_proj['year'], format='%Y')
```

```
df_usa_sales_share_proj['percentage (%)'] = df_usa_sales_share_proj['value']

# Combine historical and projection data for training
df_sales_share_combined_usa = pd.concat([df_usa_sales_share_hist,␣
 ↪df_usa_sales_share_proj])

df_sales_share_combined_usa['time'] = pd.
 ↪to_datetime(df_sales_share_combined_usa['year'], format='%Y')
df_sales_share_combined_usa['percentage (%)'] =␣
 ↪df_sales_share_combined_usa['value']

# Step 2: Initialize and fit the Prophet model with yearly seasonality and␣
 ↪changepoint sensitivity
sales_share_model_usa = Prophet(yearly_seasonality=True,  # Enable yearly␣
 ↪seasonality
                                changepoint_prior_scale=0.05)  # Adjust␣
 ↪changepoint sensitivity
sales_share_model_usa.fit(df_sales_share_combined_usa.rename(columns={'time':␣
 ↪'ds', 'percentage (%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_sales_share_years_usa = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_sales_share_usa = sales_share_model_usa.
 ↪make_future_dataframe(periods=future_sales_share_years_usa, freq='Y')

# Restrict future data to end at 2035
future_sales_share_usa = future_sales_share_usa[future_sales_share_usa['ds'] <=␣
 ↪pd.to_datetime('2035-12-31')]  # Future data only until 2035

# Make predictions
forecast_sales_share_usa = sales_share_model_usa.predict(future_sales_share_usa)

# Set a lower bound for predictions
forecast_sales_share_usa['yhat'] = forecast_sales_share_usa['yhat'].
 ↪clip(lower=0)

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot_share_usa = sales_share_model_usa.plot(forecast_sales_share_usa,␣
 ↪xlabel='Year', ylabel='Percentage (%)', ax=ax)

prediction_line_share_usa, = ax.plot(forecast_sales_share_usa['ds'],␣
 ↪forecast_sales_share_usa['yhat'], color='blue', label='Prediction Line␣
 ↪(Solid)')
```

```
projected_data_share_usa = ax.scatter(df_usa_sales_share_proj['time'],␣
 ↪df_usa_sales_share_proj['percentage (%)'],
                                      color='red', marker='o', s=100,␣
 ↪label='Projected Data (Dataset)', alpha=0.7)

forecast_points_share_usa = ax.plot(forecast_sales_share_usa['ds'],␣
 ↪forecast_sales_share_usa['yhat'], 'k.', label='Forecast Data (Black Dots)',␣
 ↪alpha=0.8)

prediction_start_share_usa = ax.axvline(x=pd.to_datetime('2024-01-01'),␣
 ↪color='blue', linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Sales Share Forecast vs. Projection Data (USA)', fontsize=18,␣
 ↪fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Sales Share Percentage (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')

# Step 8: Add the legend explicitly
legend_labels_share_usa = ['Prediction Line (Solid)', 'Forecast Data (Black␣
 ↪Dots)', 'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line_share_usa, forecast_points_share_usa[0],␣
 ↪prediction_start_share_usa, projected_data_share_usa],
          legend_labels_share_usa, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-01-31'))

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

# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
df_sales_share_cv_usa = cross_validation(sales_share_model_usa, initial='730␣
 ↪days', period='365 days', horizon='365 days')
```

```
# Calculate performance metrics
df_sales_share_p_usa = performance_metrics(df_sales_share_cv_usa)

# Output performance metrics
print(df_sales_share_p_usa[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_9000\1362382155.py:25:
SettingWithCopyWarning:
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
  df_usa_sales_share_proj['time'] =
pd.to_datetime(df_usa_sales_share_proj['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_9000\1362382155.py:26:
SettingWithCopyWarning:
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
  df_usa_sales_share_proj['percentage (%)'] = df_usa_sales_share_proj['value']
19:36:38 - cmdstanpy - INFO - Chain [1] start processing
19:36:39 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
19:36:40 - cmdstanpy - INFO - Chain [1] start processing
19:36:40 - cmdstanpy - INFO - Chain [1] done processing
19:36:40 - cmdstanpy - INFO - Chain [1] start processing
19:36:40 - cmdstanpy - INFO - Chain [1] done processing
19:36:40 - cmdstanpy - INFO - Chain [1] start processing
19:36:40 - cmdstanpy - INFO - Chain [1] done processing
19:36:40 - cmdstanpy - INFO - Chain [1] start processing
19:36:40 - cmdstanpy - INFO - Chain [1] done processing
19:36:41 - cmdstanpy - INFO - Chain [1] start processing
19:36:41 - cmdstanpy - INFO - Chain [1] done processing
19:36:41 - cmdstanpy - INFO - Chain [1] start processing
19:36:41 - cmdstanpy - INFO - Chain [1] done processing
19:36:41 - cmdstanpy - INFO - Chain [1] start processing
19:36:41 - cmdstanpy - INFO - Chain [1] done processing
19:36:41 - cmdstanpy - INFO - Chain [1] start processing
19:36:41 - cmdstanpy - INFO - Chain [1] done processing
19:36:41 - cmdstanpy - INFO - Chain [1] start processing
19:36:41 - cmdstanpy - INFO - Chain [1] done processing
19:36:41 - cmdstanpy - INFO - Chain [1] start processing
19:36:42 - cmdstanpy - INFO - Chain [1] done processing
19:36:42 - cmdstanpy - INFO - Chain [1] start processing
19:36:42 - cmdstanpy - INFO - Chain [1] done processing
19:36:42 - cmdstanpy - INFO - Chain [1] start processing
19:36:42 - cmdstanpy - INFO - Chain [1] done processing
```

```
19:36:42 - cmdstanpy - INFO - Chain [1] start processing
19:36:42 - cmdstanpy - INFO - Chain [1] done processing
19:36:42 - cmdstanpy - INFO - Chain [1] start processing
19:36:43 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	0.315825	13.558224	0.340383
1	364 days	0.890833	1.774018	0.934468
2	365 days	11.257472	0.714494	16.463593

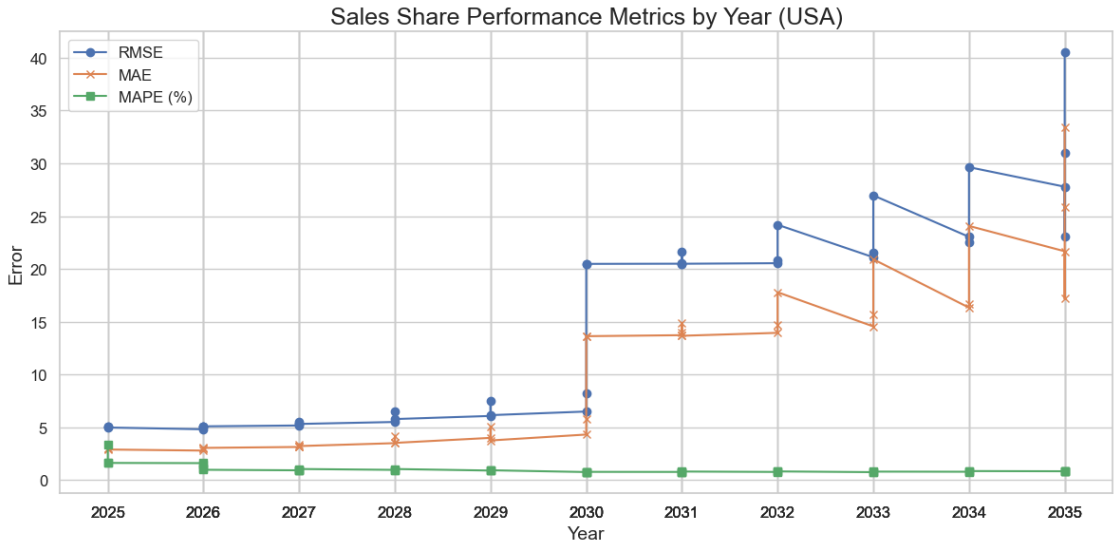
USA SALES SHARE (PREDICTION) ACCURACY GRAPH

[26]:
```python
# Step 1: Historical and Projection Data Preparation
df_usa_sasrH = df[(df['region'] == 'USA') &
                  (df['parameter'] == 'EV sales share') &
                  (df['category'] == 'Historical') &
                  (df['year'] < 2024)]

df_usa_sasrPro = df[(df['region'] == 'USA') &
                    (df['parameter'] == 'EV sales share') &
                    (df['year'] >= 2024) &
                    (df['year'] <= 2035)]

df_usa_sasrPro['ds'] = pd.to_datetime(df_usa_sasrPro['year'], format='%Y')
df_usa_sasrPro['y'] = df_usa_sasrPro['value']

df_combined_usa = pd.concat([df_usa_sasrH, df_usa_sasrPro])
df_combined_usa['ds'] = pd.to_datetime(df_combined_usa['year'], format='%Y')
df_combined_usa['y'] = df_combined_usa['value']

# Step 2: Initialize and fit the Prophet model
model_usa = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
model_usa.fit(df_combined_usa)

# Step 3: Cross-Validation
horizon_days_usa = (pd.to_datetime('2035-01-01') - pd.
 ↪to_datetime('2024-01-01')).days
df_cv_usa = cross_validation(model_usa, initial='730 days', period='365 days',␣
 ↪horizon=f'{horizon_days_usa} days')

# Step 4: Calculate Performance Metrics
df_p_usa = performance_metrics(df_cv_usa)

# Convert horizons to years for labeling
years_usa = 2024 + df_p_usa['horizon'].dt.days // 365

# Step 5: Plot Performance Metrics
plt.figure(figsize=(12, 6))  # Set figure size
```

```
plt.plot(years_usa, df_p_usa['rmse'], label='RMSE', marker='o')
plt.plot(years_usa, df_p_usa['mae'], label='MAE', marker='x')
plt.plot(years_usa, df_p_usa['mape'], label='MAPE (%)', marker='s')

# Add title and labels
plt.title('Sales Share Performance Metrics by Year (USA)', fontsize=18)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Error', fontsize=14)
plt.xticks(years_usa, fontsize=12)
plt.yticks(fontsize=12)

# Add legend and grid
plt.legend(fontsize=12)
plt.grid(True)

# Finalize and show the plot
plt.tight_layout()
plt.show()
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1879717843.py:20:
SettingWithCopyWarning:
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
  df_usa_sasrPro['ds'] = pd.to_datetime(df_usa_sasrPro['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1879717843.py:21:
SettingWithCopyWarning:
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
  df_usa_sasrPro['y'] = df_usa_sasrPro['value']
19:27:54 - cmdstanpy - INFO - Chain [1] start processing
19:27:54 - cmdstanpy - INFO - Chain [1] done processing
  0%|          | 0/13 [00:00<?, ?it/s]
19:27:54 - cmdstanpy - INFO - Chain [1] start processing
19:27:54 - cmdstanpy - INFO - Chain [1] done processing
19:27:54 - cmdstanpy - INFO - Chain [1] start processing
19:27:55 - cmdstanpy - INFO - Chain [1] done processing
19:27:55 - cmdstanpy - INFO - Chain [1] start processing
19:27:55 - cmdstanpy - INFO - Chain [1] done processing
19:27:55 - cmdstanpy - INFO - Chain [1] start processing
19:27:55 - cmdstanpy - INFO - Chain [1] done processing
19:27:55 - cmdstanpy - INFO - Chain [1] start processing
```

```
19:27:55 - cmdstanpy - INFO - Chain [1] done processing
19:27:55 - cmdstanpy - INFO - Chain [1] start processing
19:27:56 - cmdstanpy - INFO - Chain [1] done processing
19:27:56 - cmdstanpy - INFO - Chain [1] start processing
19:27:56 - cmdstanpy - INFO - Chain [1] done processing
19:27:56 - cmdstanpy - INFO - Chain [1] start processing
19:27:56 - cmdstanpy - INFO - Chain [1] done processing
19:27:56 - cmdstanpy - INFO - Chain [1] start processing
19:27:56 - cmdstanpy - INFO - Chain [1] done processing
19:27:56 - cmdstanpy - INFO - Chain [1] start processing
19:27:57 - cmdstanpy - INFO - Chain [1] done processing
19:27:57 - cmdstanpy - INFO - Chain [1] start processing
19:27:57 - cmdstanpy - INFO - Chain [1] done processing
19:27:57 - cmdstanpy - INFO - Chain [1] start processing
19:27:57 - cmdstanpy - INFO - Chain [1] done processing
19:27:57 - cmdstanpy - INFO - Chain [1] start processing
19:27:57 - cmdstanpy - INFO - Chain [1] done processing
```



[26]:
```python
#usa stock share
# EV STOCK SHARE [USA]
# Required imports
from prophet import Prophet  # Ensure Prophet is imported correctly
from prophet.diagnostics import cross_validation, performance_metrics

# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV STOCK SHARE Data Preparation (Updated variable names)
df_usa_stock_share_hist = df[(df['region'] == 'USA') &
```

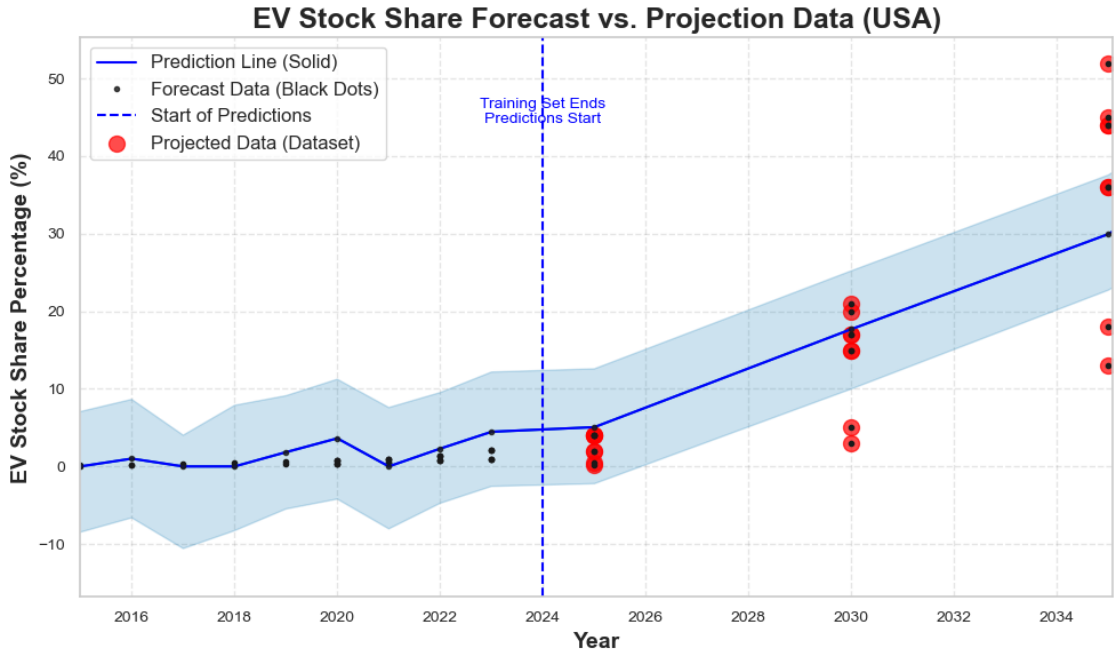
```
                                (df['parameter'] == 'EV stock share') &
                                (df['year'] < 2024)]  # Historical data (before␣
 ↪2024)

df_usa_stock_share_proj = df[(df['region'] == 'USA') &
                                (df['parameter'] == 'EV stock share') &
                                (df['year'] >= 2024) &
                                (df['year'] <= 2035)]  # Projection data (2024 to␣
 ↪2035)

# Prepare the projection data for plotting
df_usa_stock_share_proj['time'] = pd.
 ↪to_datetime(df_usa_stock_share_proj['year'], format='%Y')
df_usa_stock_share_proj['percentage (%)'] = df_usa_stock_share_proj['value']

# Combine historical and projection data for training
df_stock_share_combined_usa = pd.concat([df_usa_stock_share_hist,␣
 ↪df_usa_stock_share_proj])

df_stock_share_combined_usa['time'] = pd.
 ↪to_datetime(df_stock_share_combined_usa['year'], format='%Y')
df_stock_share_combined_usa['percentage (%)'] =␣
 ↪df_stock_share_combined_usa['value']

# Step 2: Initialize and fit the Prophet model with yearly seasonality and␣
 ↪changepoint sensitivity
stock_share_model_usa = Prophet(yearly_seasonality=True,  # Enable yearly␣
 ↪seasonality
                                changepoint_prior_scale=0.05)  # Adjust␣
 ↪changepoint sensitivity
stock_share_model_usa.fit(df_stock_share_combined_usa.rename(columns={'time':␣
 ↪'ds', 'percentage (%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_stock_share_years_usa = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_stock_share_usa = stock_share_model_usa.
 ↪make_future_dataframe(periods=future_stock_share_years_usa, freq='Y')

# Restrict future data to end at 2035
future_stock_share_usa = future_stock_share_usa[future_stock_share_usa['ds'] <=␣
 ↪pd.to_datetime('2035-12-31')]  # Future data only until 2035

# Make predictions
forecast_stock_share_usa = stock_share_model_usa.predict(future_stock_share_usa)
```

```
# Set a lower bound for predictions
forecast_stock_share_usa['yhat'] = forecast_stock_share_usa['yhat'].
 ↪clip(lower=0)  # Clip negative predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot_stock_share_usa = stock_share_model_usa.
 ↪plot(forecast_stock_share_usa, xlabel='Year', ylabel='Percentage (%)', ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
prediction_line_stock_share_usa, = ax.plot(forecast_stock_share_usa['ds'],␣
 ↪forecast_stock_share_usa['yhat'], color='blue', label='Prediction Line␣
 ↪(Solid)')

# Step 5: Overlay the projection data on the same plot
projected_data_stock_share_usa = ax.scatter(df_usa_stock_share_proj['time'],␣
 ↪df_usa_stock_share_proj['percentage (%)'],
                                            color='red', marker='o', s=100,␣
 ↪label='Projected Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points_stock_share_usa = ax.plot(forecast_stock_share_usa['ds'],␣
 ↪forecast_stock_share_usa['yhat'], 'k.', label='Forecast Data (Black Dots)',␣
 ↪alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
prediction_start_stock_share_usa = ax.axvline(x=pd.to_datetime('2024-01-01'),␣
 ↪color='blue', linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Stock Share Forecast vs. Projection Data (USA)', fontsize=18,␣
 ↪fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Stock Share Percentage (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')

# Step 8: Add the legend explicitly
```

```
legend_labels_stock_share_usa = ['Prediction Line (Solid)', 'Forecast Data␣
 ↪(Black Dots)', 'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line_stock_share_usa, forecast_points_stock_share_usa[0],␣
 ↪prediction_start_stock_share_usa, projected_data_stock_share_usa],
          legend_labels_stock_share_usa, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-01-31'))

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

# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
df_stock_share_cv_usa = cross_validation(stock_share_model_usa, initial='730␣
 ↪days', period='365 days', horizon='365 days')

# Calculate performance metrics
df_stock_share_p_usa = performance_metrics(df_stock_share_cv_usa)

# Output performance metrics
print(df_stock_share_p_usa[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_9000\3321416630.py:25:
SettingWithCopyWarning:
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
  df_usa_stock_share_proj['time'] =
pd.to_datetime(df_usa_stock_share_proj['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_9000\3321416630.py:26:
SettingWithCopyWarning:
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
  df_usa_stock_share_proj['percentage (%)'] = df_usa_stock_share_proj['value']
19:37:20 - cmdstanpy - INFO - Chain [1] start processing
19:37:21 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
```

```
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
19:37:22 - cmdstanpy - INFO - Chain [1] start processing
19:37:22 - cmdstanpy - INFO - Chain [1] done processing
19:37:22 - cmdstanpy - INFO - Chain [1] start processing
19:37:22 - cmdstanpy - INFO - Chain [1] done processing
19:37:22 - cmdstanpy - INFO - Chain [1] start processing
19:37:22 - cmdstanpy - INFO - Chain [1] done processing
19:37:22 - cmdstanpy - INFO - Chain [1] start processing
19:37:22 - cmdstanpy - INFO - Chain [1] done processing
19:37:22 - cmdstanpy - INFO - Chain [1] start processing
19:37:22 - cmdstanpy - INFO - Chain [1] done processing
19:37:23 - cmdstanpy - INFO - Chain [1] start processing
19:37:23 - cmdstanpy - INFO - Chain [1] done processing
19:37:23 - cmdstanpy - INFO - Chain [1] start processing
19:37:23 - cmdstanpy - INFO - Chain [1] done processing
19:37:23 - cmdstanpy - INFO - Chain [1] start processing
19:37:23 - cmdstanpy - INFO - Chain [1] done processing
19:37:23 - cmdstanpy - INFO - Chain [1] start processing
```

```
19:37:23 - cmdstanpy - INFO - Chain [1] done processing
19:37:23 - cmdstanpy - INFO - Chain [1] start processing
19:37:23 - cmdstanpy - INFO - Chain [1] done processing
19:37:24 - cmdstanpy - INFO - Chain [1] start processing
19:37:24 - cmdstanpy - INFO - Chain [1] done processing
19:37:24 - cmdstanpy - INFO - Chain [1] start processing
19:37:24 - cmdstanpy - INFO - Chain [1] done processing
19:37:24 - cmdstanpy - INFO - Chain [1] start processing
19:37:24 - cmdstanpy - INFO - Chain [1] done processing
19:37:24 - cmdstanpy - INFO - Chain [1] start processing
19:37:25 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	0.047961	0.511806	0.059881
1	364 days	0.169167	0.332921	0.203276
2	365 days	7.175086	0.646531	13.033608

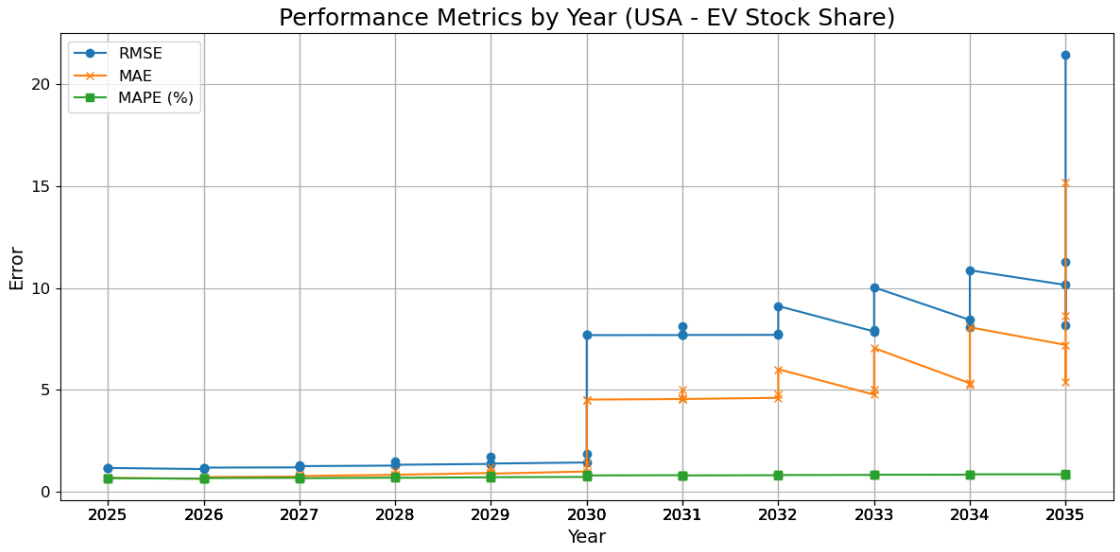
USA STOCK SHARE (PREDICTION) ACCURACY

[16]:
```python
from prophet.diagnostics import cross_validation, performance_metrics

# Assuming the dataframe `df` is already loaded and preprocessed as described

# Step 1: Historical and Projection Data Preparation
df_usa_stockH = df[(df['region'] == 'USA') &
                   (df['parameter'] == 'EV stock share') &
                   (df['category'] == 'Historical') &
                   (df['year'] < 2024)]

df_usa_stockPro = df[(df['region'] == 'USA') &
                     (df['parameter'] == 'EV stock share') &
                     (df['year'] >= 2024) &
                     (df['year'] <= 2035)]

df_usa_stockPro['ds'] = pd.to_datetime(df_usa_stockPro['year'], format='%Y')
df_usa_stockPro['y'] = df_usa_stockPro['value']

df_combined_usa = pd.concat([df_usa_stockH, df_usa_stockPro])
df_combined_usa['ds'] = pd.to_datetime(df_combined_usa['year'], format='%Y')
df_combined_usa['y'] = df_combined_usa['value']

# Step 2: Initialize and fit the Prophet model
model_usa = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
model_usa.fit(df_combined_usa)

# Step 3: Cross-Validation
horizon_days_usa = (pd.to_datetime('2035-01-01') - pd.
 ↪to_datetime('2024-01-01')).days
```

```
df_cv_usa = cross_validation(model_usa, initial='730 days', period='365 days',␣
 ↪horizon=f'{horizon_days_usa} days')

# Step 4: Calculate Performance Metrics
df_p_usa = performance_metrics(df_cv_usa)

# Convert horizons to years for labeling
years_usa = 2024 + df_p_usa['horizon'].dt.days // 365

# Step 5: Plot Performance Metrics
plt.figure(figsize=(12, 6))  # Set figure size
plt.plot(years_usa, df_p_usa['rmse'], label='RMSE', marker='o')
plt.plot(years_usa, df_p_usa['mae'], label='MAE', marker='x')
plt.plot(years_usa, df_p_usa['mape'], label='MAPE (%)', marker='s')

# Add title and labels
plt.title('Performance Metrics by Year (USA - EV Stock Share)', fontsize=18)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Error', fontsize=14)
plt.xticks(years_usa, fontsize=12)
plt.yticks(fontsize=12)

# Add legend and grid
plt.legend(fontsize=12)
plt.grid(True)

# Finalize and show the plot
plt.tight_layout()
plt.show()
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1253581727.py:20:
SettingWithCopyWarning:
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
  df_usa_stockPro['ds'] = pd.to_datetime(df_usa_stockPro['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1253581727.py:21:
SettingWithCopyWarning:
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
  df_usa_stockPro['y'] = df_usa_stockPro['value']
18:47:06 - cmdstanpy - INFO - Chain [1] start processing
18:47:06 - cmdstanpy - INFO - Chain [1] done processing
```

```
  0%|          | 0/13 [00:00<?, ?it/s]
18:47:06 - cmdstanpy - INFO - Chain [1] start processing
18:47:07 - cmdstanpy - INFO - Chain [1] done processing
18:47:07 - cmdstanpy - INFO - Chain [1] start processing
18:47:07 - cmdstanpy - INFO - Chain [1] done processing
18:47:07 - cmdstanpy - INFO - Chain [1] start processing
18:47:07 - cmdstanpy - INFO - Chain [1] done processing
18:47:07 - cmdstanpy - INFO - Chain [1] start processing
18:47:07 - cmdstanpy - INFO - Chain [1] done processing
18:47:07 - cmdstanpy - INFO - Chain [1] start processing
18:47:08 - cmdstanpy - INFO - Chain [1] done processing
18:47:08 - cmdstanpy - INFO - Chain [1] start processing
18:47:08 - cmdstanpy - INFO - Chain [1] done processing
18:47:08 - cmdstanpy - INFO - Chain [1] start processing
18:47:08 - cmdstanpy - INFO - Chain [1] done processing
18:47:08 - cmdstanpy - INFO - Chain [1] start processing
18:47:08 - cmdstanpy - INFO - Chain [1] done processing
18:47:08 - cmdstanpy - INFO - Chain [1] start processing
18:47:08 - cmdstanpy - INFO - Chain [1] done processing
18:47:09 - cmdstanpy - INFO - Chain [1] start processing
18:47:09 - cmdstanpy - INFO - Chain [1] done processing
18:47:09 - cmdstanpy - INFO - Chain [1] start processing
18:47:09 - cmdstanpy - INFO - Chain [1] done processing
18:47:09 - cmdstanpy - INFO - Chain [1] start processing
18:47:09 - cmdstanpy - INFO - Chain [1] done processing
18:47:09 - cmdstanpy - INFO - Chain [1] start processing
18:47:09 - cmdstanpy - INFO - Chain [1] done processing
```



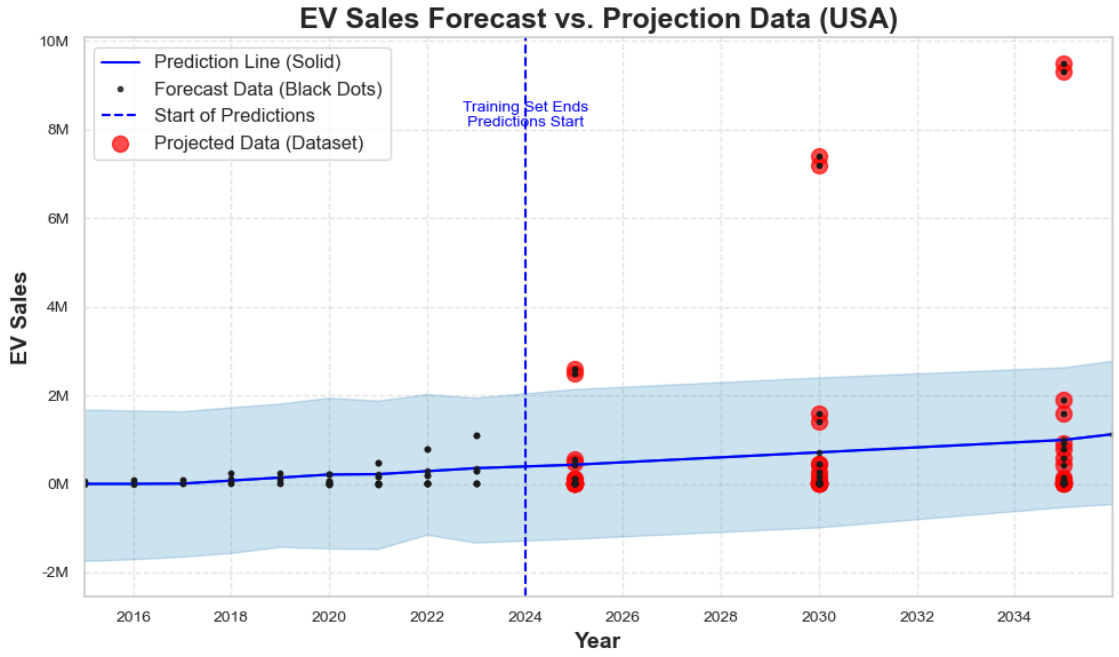
[182]:
```
#usa sales
# EV SALES [USA]
# Required imports
from prophet.diagnostics import cross_validation, performance_metrics

# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV SALES Data Preparation
df_usa_sales_hist = df[(df['region'] == 'USA') &
                       (df['parameter'] == 'EV sales') &
                       (df['year'] < 2024)]  # Historical data (before 2024)

df_usa_sales_proj = df[(df['region'] == 'USA') &
                       (df['parameter'] == 'EV sales') &
                       (df['year'] >= 2024) &
                       (df['year'] <= 2035)]  # Projection data (2024 to 2035)

# Prepare the projection data for plotting
df_usa_sales_proj['time'] = pd.to_datetime(df_usa_sales_proj['year'],␣
 ↪format='%Y')
df_usa_sales_proj['sales'] = df_usa_sales_proj['value']

# Combine historical and projection data for training
df_sales_combined_usa = pd.concat([df_usa_sales_hist, df_usa_sales_proj])

df_sales_combined_usa['time'] = pd.to_datetime(df_sales_combined_usa['year'],␣
 ↪format='%Y')
df_sales_combined_usa['sales'] = df_sales_combined_usa['value']

# Step 2: Initialize and fit the Prophet model
sales_model_usa = Prophet(yearly_seasonality=True,  # Enable yearly seasonality
                          changepoint_prior_scale=0.05)  # Adjust changepoint␣
 ↪sensitivity
sales_model_usa.fit(df_sales_combined_usa.rename(columns={'time': 'ds', 'sales':
 ↪ 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_sales_years_usa = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_sales_usa = sales_model_usa.
 ↪make_future_dataframe(periods=future_sales_years_usa, freq='Y')

# Restrict future data to end at 2035
future_sales_usa = future_sales_usa[future_sales_usa['ds'] <= pd.
 ↪to_datetime('2035-12-31')]  # Future data only until 2035
```

```
# Make predictions
forecast_sales_usa = sales_model_usa.predict(future_sales_usa)

# Set a lower bound for predictions
forecast_sales_usa['yhat'] = forecast_sales_usa['yhat'].clip(lower=0)  # Clip␣
 ↪negative predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot_sales_usa = sales_model_usa.plot(forecast_sales_usa,␣
 ↪xlabel='Year', ylabel='Sales', ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
prediction_line_sales_usa, = ax.plot(forecast_sales_usa['ds'],␣
 ↪forecast_sales_usa['yhat'], color='blue', label='Prediction Line (Solid)')

# Step 5: Overlay the projection data on the same plot
projected_data_sales_usa = ax.scatter(df_usa_sales_proj['time'],␣
 ↪df_usa_sales_proj['sales'],
                                      color='red', marker='o', s=100,␣
 ↪label='Projected Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points_sales_usa = ax.plot(forecast_sales_usa['ds'],␣
 ↪forecast_sales_usa['yhat'], 'k.', label='Forecast Data (Black Dots)',␣
 ↪alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
prediction_start_sales_usa = ax.axvline(x=pd.to_datetime('2024-01-01'),␣
 ↪color='blue', linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Sales Forecast vs. Projection Data (USA)', fontsize=18,␣
 ↪fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Sales', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')
```

```
def millions(x, pos):
    return f'{int(x / 1e6)}M'  # Convert to millions

formatter = FuncFormatter(millions)
plt.gca().yaxis.set_major_formatter(formatter)

# Step 8: Add the legend explicitly
legend_labels_sales_usa = ['Prediction Line (Solid)', 'Forecast Data (Black␣
 ↪Dots)', 'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line_sales_usa, forecast_points_sales_usa[0],␣
 ↪prediction_start_sales_usa, projected_data_sales_usa],
          legend_labels_sales_usa, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-12-31'))

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

# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
df_sales_cv_usa = cross_validation(sales_model_usa, initial='730 days',␣
 ↪period='365 days', horizon='365 days')

# Calculate performance metrics
df_sales_p_usa = performance_metrics(df_sales_cv_usa)

# Output performance metrics
print(df_sales_p_usa[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\3239168336.py:25:
SettingWithCopyWarning:
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
  df_usa_sales_proj['time'] = pd.to_datetime(df_usa_sales_proj['year'],
format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\3239168336.py:26:
SettingWithCopyWarning:
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
  df_usa_sales_proj['sales'] = df_usa_sales_proj['value']
01:28:27 - cmdstanpy - INFO - Chain [1] start processing
01:28:27 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
01:28:27 - cmdstanpy - INFO - Chain [1] start processing
01:28:27 - cmdstanpy - INFO - Chain [1] done processing
01:28:28 - cmdstanpy - INFO - Chain [1] start processing
01:28:28 - cmdstanpy - INFO - Chain [1] done processing
01:28:28 - cmdstanpy - INFO - Chain [1] start processing
01:28:28 - cmdstanpy - INFO - Chain [1] done processing
01:28:28 - cmdstanpy - INFO - Chain [1] start processing
01:28:28 - cmdstanpy - INFO - Chain [1] done processing
01:28:29 - cmdstanpy - INFO - Chain [1] start processing
01:28:29 - cmdstanpy - INFO - Chain [1] done processing
```

```
01:28:29 - cmdstanpy - INFO - Chain [1] start processing
01:28:29 - cmdstanpy - INFO - Chain [1] done processing
01:28:29 - cmdstanpy - INFO - Chain [1] start processing
01:28:29 - cmdstanpy - INFO - Chain [1] done processing
01:28:29 - cmdstanpy - INFO - Chain [1] start processing
01:28:29 - cmdstanpy - INFO - Chain [1] done processing
01:28:30 - cmdstanpy - INFO - Chain [1] start processing
01:28:30 - cmdstanpy - INFO - Chain [1] done processing
01:28:30 - cmdstanpy - INFO - Chain [1] start processing
01:28:30 - cmdstanpy - INFO - Chain [1] done processing
01:28:30 - cmdstanpy - INFO - Chain [1] start processing
01:28:30 - cmdstanpy - INFO - Chain [1] done processing
01:28:30 - cmdstanpy - INFO - Chain [1] start processing
01:28:30 - cmdstanpy - INFO - Chain [1] done processing
01:28:30 - cmdstanpy - INFO - Chain [1] start processing
01:28:31 - cmdstanpy - INFO - Chain [1] done processing
01:28:31 - cmdstanpy - INFO - Chain [1] start processing
01:28:31 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	31375.833284	372.347060	3.402066e+04
1	364 days	78325.828479	467.020198	9.028457e+04
2	365 days	655478.215746	234.964121	1.559360e+06

[158]:
```python
#usa sales accuracy
# USA EV SALES Accuracy
from prophet.diagnostics import cross_validation, performance_metrics
df_usa_sales_hist = df[(df['region'] == 'USA') &
                       (df['parameter'] == 'EV sales') &
                       (df['year'] < 2024)]  # Historical data

df_usa_sales_proj = df[(df['region'] == 'USA') &
                       (df['parameter'] == 'EV sales') &
                       (df['year'] >= 2024) &
                       (df['year'] <= 2035)]  # Projection data

# Prepare the projection data for plotting
df_usa_sales_proj['ds'] = pd.to_datetime(df_usa_sales_proj['year'], format='%Y')
df_usa_sales_proj['y'] = df_usa_sales_proj['value']

# Combine historical and projection data for training
df_combined_usa = pd.concat([df_usa_sales_hist, df_usa_sales_proj])

df_combined_usa['ds'] = pd.to_datetime(df_combined_usa['year'], format='%Y')
df_combined_usa['y'] = df_combined_usa['value']

# Step 2: Initialize and fit the Prophet model
model_usa = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
```

```
model_usa.fit(df_combined_usa)

# Step 3: Cross-Validation to assess model performance with different horizons
# Calculate the total number of days from 2024 to 2035
horizon_days_usa = (pd.to_datetime('2035-01-01') - pd.
 ↪to_datetime('2024-01-01')).days

# Perform cross-validation with horizon set to the calculated days
df_cv_usa = cross_validation(model_usa, initial='730 days', period='365 days',␣
 ↪horizon=f'{horizon_days_usa} days')

# Step 4: Calculate performance metrics
df_p_usa = performance_metrics(df_cv_usa)

# Generate year labels based on the prediction horizons
start_year_usa = 2024  # Starting year for predictions

# Convert horizons from days to years
years_usa = [start_year_usa + (horizon.days // 365) for horizon in␣
 ↪df_p_usa['horizon']]

# Output performance metrics
print(df_p_usa[['horizon', 'mae', 'mape', 'rmse']])

# Optional: Plot the performance metrics with years on the x-axis
plt.figure(figsize=(12, 6))
plt.plot(years_usa, df_p_usa['rmse'], label='RMSE', marker='o')
plt.plot(years_usa, df_p_usa['mae'], label='MAE', marker='x')
plt.plot(years_usa, df_p_usa['mape'], label='MAPE (%)', marker='s')
plt.title('Performance Metrics by Year (EV Sales - USA)', fontsize=18)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Error', fontsize=14)
plt.xticks(years_usa, fontsize=12)  # Set x-ticks to show years
plt.yticks(fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

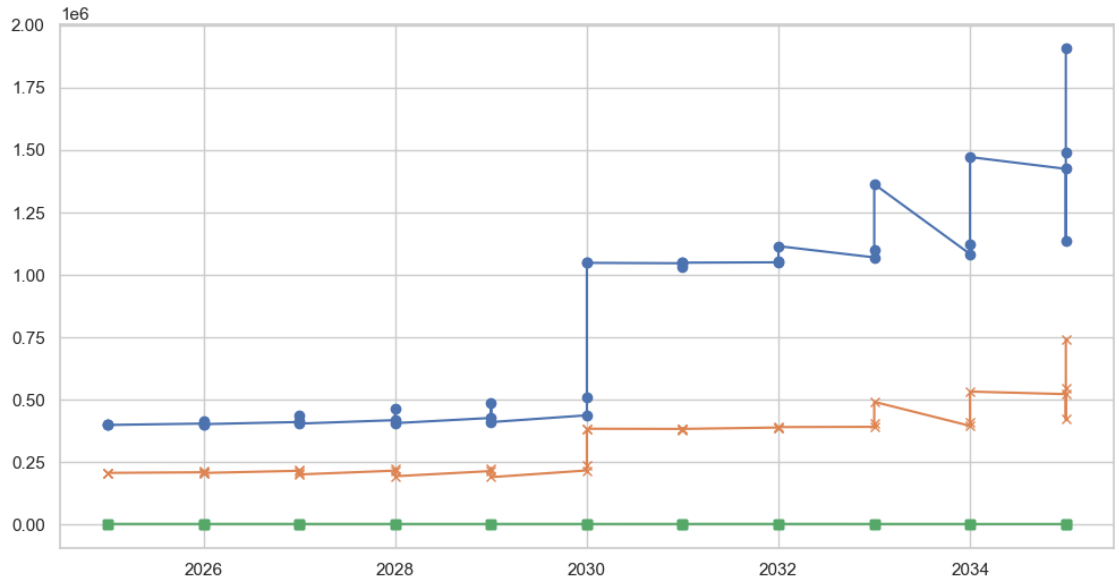
```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\2608185827.py:22:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_usa_sales_proj['ds'] = pd.to_datetime(df_usa_sales_proj['year'],
```

```
format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\2608185827.py:23:
SettingWithCopyWarning:
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
  df_usa_sales_proj['y'] = df_usa_sales_proj['value']
00:35:30 - cmdstanpy - INFO - Chain [1] start processing
00:35:30 - cmdstanpy - INFO - Chain [1] done processing

  0%|          | 0/13 [00:00<?, ?it/s]

00:35:30 - cmdstanpy - INFO - Chain [1] start processing
00:35:30 - cmdstanpy - INFO - Chain [1] done processing
00:35:31 - cmdstanpy - INFO - Chain [1] start processing
00:35:31 - cmdstanpy - INFO - Chain [1] done processing
00:35:31 - cmdstanpy - INFO - Chain [1] start processing
00:35:31 - cmdstanpy - INFO - Chain [1] done processing
00:35:31 - cmdstanpy - INFO - Chain [1] start processing
00:35:31 - cmdstanpy - INFO - Chain [1] done processing
00:35:31 - cmdstanpy - INFO - Chain [1] start processing
00:35:31 - cmdstanpy - INFO - Chain [1] done processing
00:35:32 - cmdstanpy - INFO - Chain [1] start processing
00:35:32 - cmdstanpy - INFO - Chain [1] done processing
00:35:32 - cmdstanpy - INFO - Chain [1] start processing
00:35:32 - cmdstanpy - INFO - Chain [1] done processing
00:35:32 - cmdstanpy - INFO - Chain [1] start processing
00:35:32 - cmdstanpy - INFO - Chain [1] done processing
00:35:33 - cmdstanpy - INFO - Chain [1] start processing
00:35:33 - cmdstanpy - INFO - Chain [1] done processing
00:35:33 - cmdstanpy - INFO - Chain [1] start processing
00:35:33 - cmdstanpy - INFO - Chain [1] done processing
00:35:33 - cmdstanpy - INFO - Chain [1] start processing
00:35:33 - cmdstanpy - INFO - Chain [1] done processing
00:35:34 - cmdstanpy - INFO - Chain [1] start processing
00:35:34 - cmdstanpy - INFO - Chain [1] done processing
00:35:34 - cmdstanpy - INFO - Chain [1] start processing
00:35:34 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	366 days	204105.218902	396.831003	3.968745e+05
1	728 days	204551.582188	381.278196	3.969100e+05
2	729 days	205025.844299	376.074932	3.970715e+05
3	730 days	207481.722555	386.841302	4.026263e+05
4	731 days	210845.311719	378.930951	4.111416e+05
5	1093 days	203868.660055	372.095746	3.999494e+05
6	1094 days	204780.667359	355.386656	4.003848e+05

```
7  1095 days  213474.848789  357.492152  4.089558e+05
8  1096 days  215806.034241  327.696453  4.343508e+05
9  1458 days  202072.617296  325.273618  4.129059e+05
10 1459 days  198806.944968  346.486543  4.025720e+05
11 1460 days  214005.182557  319.574670  4.163706e+05
12 1461 days  221821.908940  268.239747  4.616534e+05
13 1824 days  191593.174856  284.611492  4.043892e+05
14 1825 days  212350.621576  288.082113  4.252334e+05
15 1826 days  223599.655310  235.648345  4.855994e+05
16 2189 days  187851.584420  308.958963  4.085256e+05
17 2190 days  214808.454548  314.216987  4.357026e+05
18 2191 days  234233.734537  254.586533  5.069130e+05
19 2192 days  381127.017692  230.349517  1.047126e+06
20 2554 days  381838.215883  288.178742  1.046667e+06
21 2555 days  381176.789304  336.322063  1.045075e+06
22 2556 days  374744.385506  336.048665  1.028830e+06
23 2557 days  382522.364634  328.953491  1.047458e+06
24 2919 days  381339.888180  331.136264  1.047375e+06
25 2920 days  386797.639313  388.720529  1.048888e+06
26 2921 days  384588.991781  373.979836  1.055168e+06
27 2922 days  388505.751097  371.407860  1.113609e+06
28 3285 days  389443.605780  319.141580  1.068478e+06
29 3286 days  401957.205207  219.806968  1.098645e+06
30 3287 days  489691.952692  215.557622  1.362116e+06
31 3650 days  393515.502959  197.314220  1.082047e+06
32 3651 days  408368.790751   88.922861  1.122929e+06
33 3652 days  530852.008861   43.083545  1.470525e+06
34 4015 days  520605.152652   44.671820  1.423329e+06
35 4016 days  419611.054018   49.084862  1.132860e+06
36 4017 days  545173.175194   27.015619  1.490641e+06
37 4018 days  741735.782369   26.915989  1.907933e+06
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[158], line 59
     57 plt.plot(years_usa, df_p_usa['mae'], label='MAE', marker='x')
     58 plt.plot(years_usa, df_p_usa['mape'], label='MAPE (%)', marker='s')
---> 59 plt.title('Performance Metrics by Year (EV Sales - USA)', fontsize=18)
     60 plt.xlabel('Year', fontsize=14)
     61 plt.ylabel('Error', fontsize=14)

TypeError: 'str' object is not callable
```



[ ]:

India furute Market

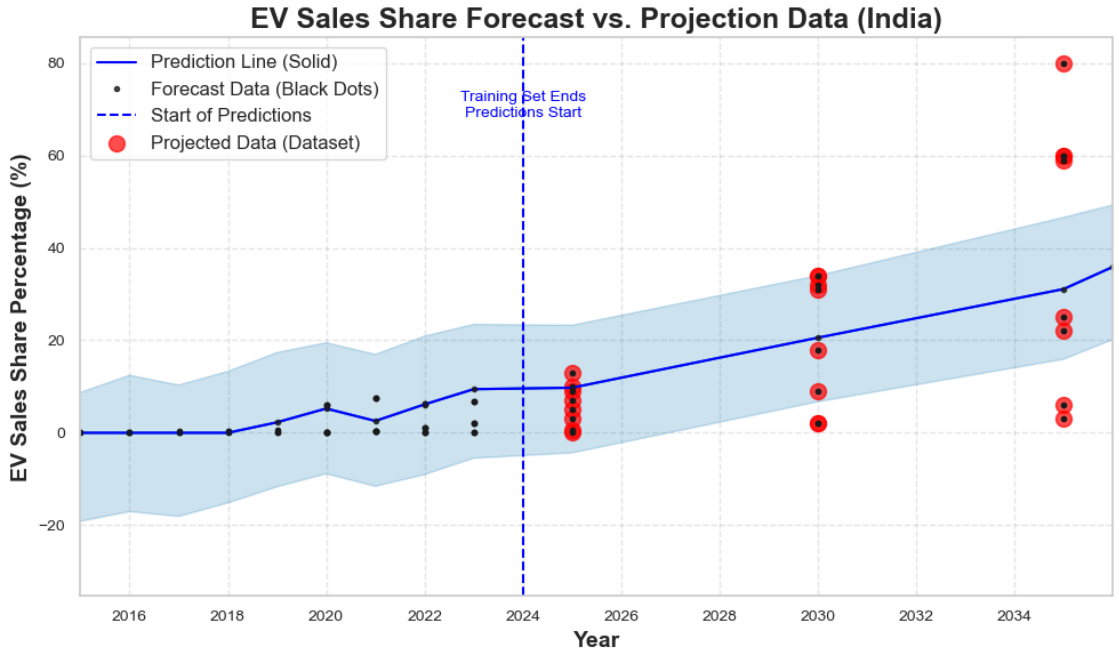
[149]:
```
#sales share [india]

# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV SALES SHARE Data Preparation (Updated variable names)
df_india_sales_share_hist = df[(df['region'] == 'India') &
                               (df['parameter'] == 'EV sales share') &
                               (df['year'] < 2024)]  # Historical data (before␣
 ↪2024)

df_india_sales_share_proj = df[(df['region'] == 'India') &
                               (df['parameter'] == 'EV sales share') &
                               (df['year'] >= 2024) &
                               (df['year'] <= 2035)]  # Projection data (2024␣
 ↪to 2035)

# Prepare the projection data for plotting
df_india_sales_share_proj['time'] = pd.
 ↪to_datetime(df_india_sales_share_proj['year'], format='%Y')
df_india_sales_share_proj['percentage (%)'] = df_india_sales_share_proj['value']

# Combine historical and projection data for training
```

```
df_sales_share_combined = pd.concat([df_india_sales_share_hist,␣
 ↪df_india_sales_share_proj])

df_sales_share_combined['time'] = pd.
 ↪to_datetime(df_sales_share_combined['year'], format='%Y')
df_sales_share_combined['percentage (%)'] = df_sales_share_combined['value']

# Step 2: Initialize and fit the Prophet model with yearly seasonality and␣
 ↪changepoint sensitivity
sales_share_model = Prophet(yearly_seasonality=True,  # Enable yearly␣
 ↪seasonality
                            changepoint_prior_scale=0.05)  # Adjust changepoint␣
 ↪sensitivity
sales_share_model.fit(df_sales_share_combined.rename(columns={'time': 'ds',␣
 ↪'percentage (%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_sales_share_years = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_sales_share = sales_share_model.
 ↪make_future_dataframe(periods=future_sales_share_years, freq='Y')

# Restrict future data to end at 2035
future_sales_share = future_sales_share[future_sales_share['ds'] <= pd.
 ↪to_datetime('2035-12-31')]  # Future data only until 2035

# Make predictions
forecast_sales_share = sales_share_model.predict(future_sales_share)

# Set a lower bound for predictions
forecast_sales_share['yhat'] = forecast_sales_share['yhat'].clip(lower=0)  #␣
 ↪Clip negative predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot_share = sales_share_model.plot(forecast_sales_share,␣
 ↪xlabel='Year', ylabel='Percentage (%)', ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
prediction_line_share, = ax.plot(forecast_sales_share['ds'],␣
 ↪forecast_sales_share['yhat'], color='blue', label='Prediction Line (Solid)')

# Step 5: Overlay the projection data on the same plot
projected_data_share = ax.scatter(df_india_sales_share_proj['time'],␣
 ↪df_india_sales_share_proj['percentage (%)'],
```

```
                                    color='red', marker='o', s=100,␣
 ↪label='Projected Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points_share = ax.plot(forecast_sales_share['ds'],␣
 ↪forecast_sales_share['yhat'], 'k.', label='Forecast Data (Black Dots)',␣
 ↪alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
prediction_start_share = ax.axvline(x=pd.to_datetime('2024-01-01'),␣
 ↪color='blue', linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Sales Share Forecast vs. Projection Data (India)',␣
 ↪fontsize=18, fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Sales Share Percentage (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')

# Step 8: Add the legend explicitly
legend_labels_share = ['Prediction Line (Solid)', 'Forecast Data (Black Dots)',␣
 ↪'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line_share, forecast_points_share[0],␣
 ↪prediction_start_share, projected_data_share],
          legend_labels_share, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-12-31'))

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

# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
df_sales_share_cv = cross_validation(sales_share_model, initial='730 days',␣
 ↪period='365 days', horizon='365 days')
```

```
# Calculate performance metrics
df_sales_share_p = performance_metrics(df_sales_share_cv)

# Output performance metrics
print(df_sales_share_p[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\1709995603.py:24:
SettingWithCopyWarning:
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
  df_india_sales_share_proj['time'] =
pd.to_datetime(df_india_sales_share_proj['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\1709995603.py:25:
SettingWithCopyWarning:
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
  df_india_sales_share_proj['percentage (%)'] =
df_india_sales_share_proj['value']
00:24:17 - cmdstanpy - INFO - Chain [1] start processing
00:24:18 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
00:24:19 - cmdstanpy - INFO - Chain [1] start processing
00:24:19 - cmdstanpy - INFO - Chain [1] done processing
00:24:19 - cmdstanpy - INFO - Chain [1] start processing
00:24:19 - cmdstanpy - INFO - Chain [1] done processing
00:24:19 - cmdstanpy - INFO - Chain [1] start processing
00:24:19 - cmdstanpy - INFO - Chain [1] done processing
00:24:19 - cmdstanpy - INFO - Chain [1] start processing
00:24:19 - cmdstanpy - INFO - Chain [1] done processing
00:24:19 - cmdstanpy - INFO - Chain [1] start processing
00:24:19 - cmdstanpy - INFO - Chain [1] done processing
00:24:19 - cmdstanpy - INFO - Chain [1] start processing
00:24:20 - cmdstanpy - INFO - Chain [1] done processing
00:24:20 - cmdstanpy - INFO - Chain [1] start processing
00:24:20 - cmdstanpy - INFO - Chain [1] done processing
00:24:20 - cmdstanpy - INFO - Chain [1] start processing
00:24:20 - cmdstanpy - INFO - Chain [1] done processing
00:24:20 - cmdstanpy - INFO - Chain [1] start processing
00:24:20 - cmdstanpy - INFO - Chain [1] done processing
00:24:20 - cmdstanpy - INFO - Chain [1] start processing
00:24:20 - cmdstanpy - INFO - Chain [1] done processing
00:24:21 - cmdstanpy - INFO - Chain [1] start processing
00:24:21 - cmdstanpy - INFO - Chain [1] done processing
00:24:21 - cmdstanpy - INFO - Chain [1] start processing
00:24:21 - cmdstanpy - INFO - Chain [1] done processing
```

```
00:24:21 - cmdstanpy - INFO - Chain [1] start processing
00:24:21 - cmdstanpy - INFO - Chain [1] done processing
00:24:21 - cmdstanpy - INFO - Chain [1] start processing
00:24:22 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	0.029536	0.725476	0.039669
1	364 days	1.059356	0.988159	2.361782
2	365 days	8.789415	5.212211	15.634974

[18]:
```
# Filter for EV sales share
df_india_sasrH = df[(df['region'] == 'India') &
                    (df['parameter'] == 'EV sales share') &
                    (df['year'] < 2024)]  # Historical data

df_india_sasrPro = df[(df['region'] == 'India') &
                      (df['parameter'] == 'EV sales share') &
                      (df['year'] >= 2024) &
                      (df['year'] <= 2035)]  # Projection data

# Prepare the projection data for plotting
df_india_sasrPro['ds'] = pd.to_datetime(df_india_sasrPro['year'], format='%Y')
df_india_sasrPro['y'] = df_india_sasrPro['value']

# Combine historical and projection data for training
df_combined2 = pd.concat([df_india_sasrH, df_india_sasrPro])

df_combined2['ds'] = pd.to_datetime(df_combined2['year'], format='%Y')
df_combined2['y'] = df_combined2['value']

# Step 2: Initialize and fit the Prophet model
model = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
model.fit(df_combined2)

# Step 3: Cross-Validation to assess model performance with different horizons
# Calculate the total number of days from 2024 to 2035
horizon_days = (pd.to_datetime('2035-01-01') - pd.to_datetime('2024-01-01')).
 ↪days

# Perform cross-validation with horizon set to the calculated days
df_cv = cross_validation(model, initial='730 days', period='365 days',␣
 ↪horizon=f'{horizon_days} days')

# Step 4: Calculate performance metrics
df_p = performance_metrics(df_cv)

# Generate year labels based on the prediction horizons
start_year = 2024  # Starting year for predictions
```

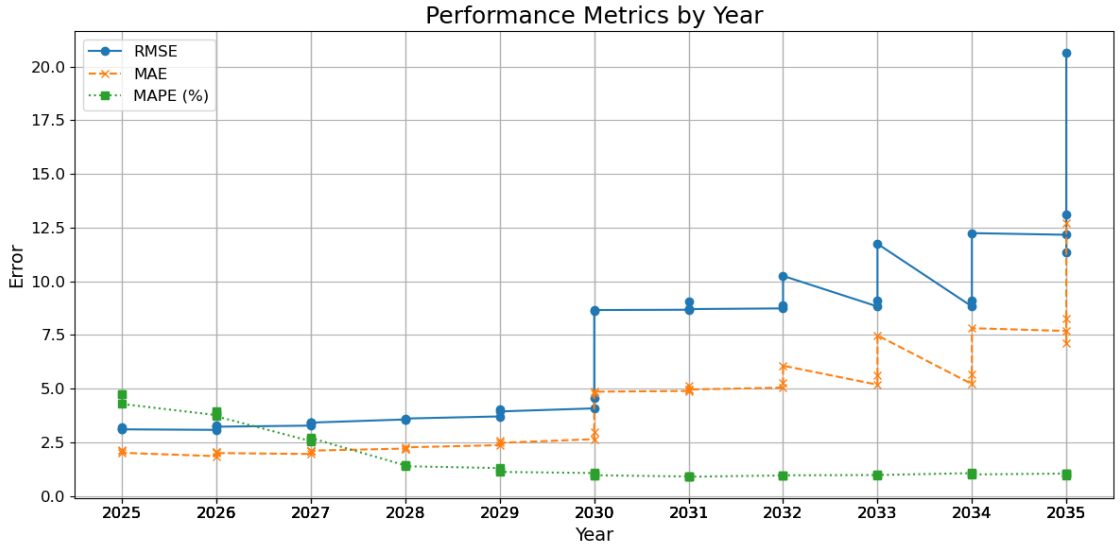
```
# Convert horizons from days to years
years = [start_year + (horizon.days // 365) for horizon in df_p['horizon']]

# Output performance metrics
print(df_p[['horizon', 'mae', 'mape', 'rmse']])

# Optional: Plot the performance metrics with years on the x-axis
plt.figure(figsize=(12, 6))
plt.plot(years, df_p['rmse'], label='RMSE', marker='o', linestyle='-')
plt.plot(years, df_p['mae'], label='MAE', marker='x', linestyle='--')
plt.plot(years, df_p['mape'], label='MAPE (%)', marker='s', linestyle=':')
plt.title('Sale Share Performance Metrics by Year', fontsize=18)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Error', fontsize=14)
plt.xticks(years, fontsize=12)  # Set x-ticks to show years
plt.yticks(fontsize=12)

# Add legend with adjusted fontsize
plt.legend(fontsize=12, loc='upper left')  # Specify location if needed
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1139271681.py:19:
SettingWithCopyWarning:
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
  df_india_sasrPro['ds'] = pd.to_datetime(df_india_sasrPro['year'], format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1139271681.py:20:
SettingWithCopyWarning:
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
  df_india_sasrPro['y'] = df_india_sasrPro['value']
19:14:28 - cmdstanpy - INFO - Chain [1] start processing
19:14:28 - cmdstanpy - INFO - Chain [1] done processing
  0%|          | 0/13 [00:00<?, ?it/s]
19:14:28 - cmdstanpy - INFO - Chain [1] start processing
19:14:28 - cmdstanpy - INFO - Chain [1] done processing
19:14:28 - cmdstanpy - INFO - Chain [1] start processing
19:14:28 - cmdstanpy - INFO - Chain [1] done processing
```

```
19:14:29 - cmdstanpy - INFO - Chain [1] start processing
19:14:29 - cmdstanpy - INFO - Chain [1] done processing
19:14:29 - cmdstanpy - INFO - Chain [1] start processing
19:14:29 - cmdstanpy - INFO - Chain [1] done processing
19:14:29 - cmdstanpy - INFO - Chain [1] start processing
19:14:29 - cmdstanpy - INFO - Chain [1] done processing
19:14:29 - cmdstanpy - INFO - Chain [1] start processing
19:14:30 - cmdstanpy - INFO - Chain [1] done processing
19:14:30 - cmdstanpy - INFO - Chain [1] start processing
19:14:30 - cmdstanpy - INFO - Chain [1] done processing
19:14:30 - cmdstanpy - INFO - Chain [1] start processing
19:14:30 - cmdstanpy - INFO - Chain [1] done processing
19:14:30 - cmdstanpy - INFO - Chain [1] start processing
19:14:30 - cmdstanpy - INFO - Chain [1] done processing
19:14:31 - cmdstanpy - INFO - Chain [1] start processing
19:14:31 - cmdstanpy - INFO - Chain [1] done processing
19:14:31 - cmdstanpy - INFO - Chain [1] start processing
19:14:31 - cmdstanpy - INFO - Chain [1] done processing
19:14:31 - cmdstanpy - INFO - Chain [1] start processing
19:14:31 - cmdstanpy - INFO - Chain [1] done processing
19:14:31 - cmdstanpy - INFO - Chain [1] start processing
19:14:31 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	366 days	2.132877	4.757627	3.177765
1	728 days	2.077881	4.725168	3.130420
2	729 days	2.007753	4.281934	3.100877
3	730 days	1.850788	3.763404	3.076105
4	731 days	2.070537	3.912131	3.280507
5	1093 days	2.034640	3.946299	3.249092
6	1094 days	1.995566	3.705517	3.223888
7	1095 days	1.953015	2.548478	3.274705
8	1096 days	2.101328	2.622037	3.411340
9	1458 days	2.100867	2.626021	3.411327
10	1459 days	2.108468	2.695419	3.414543
11	1460 days	2.204938	1.421765	3.561708
12	1461 days	2.205600	1.419482	3.563550
13	1824 days	2.261461	1.380246	3.605151
14	1825 days	2.363835	1.296317	3.702768
15	1826 days	2.568193	1.182965	4.016249
16	2189 days	2.471648	1.117654	3.934699
17	2190 days	2.643695	1.065699	4.078327
18	2191 days	2.986772	1.021197	4.549180
19	2192 days	4.786126	0.936538	8.634442
20	2554 days	4.855758	0.962934	8.658591
21	2555 days	4.887898	0.903362	8.672873
22	2556 days	4.952212	0.898082	8.717287
23	2557 days	5.149608	0.899144	9.076295

```
24 2919 days    4.955222  0.893109    8.703380
25 2920 days    5.044773  0.953586    8.735555
26 2921 days    5.272301  0.954791    8.886063
27 2922 days    6.066694  0.959948   10.250053
28 3285 days    5.179430  0.972281    8.825441
29 3286 days    5.633925  0.967854    9.121119
30 3287 days    7.477622  0.973122   11.744143
31 3650 days    5.211544  1.061434    8.826574
32 3651 days    5.655914  1.040645    9.117200
33 3652 days    7.813113  0.999602   12.241837
34 4015 days    7.684166  1.041481   12.165052
35 4016 days    7.109764  1.048903   11.350446
36 4017 days    8.279404  1.030593   13.122211
37 4018 days   12.708530  0.971369   20.655875
```



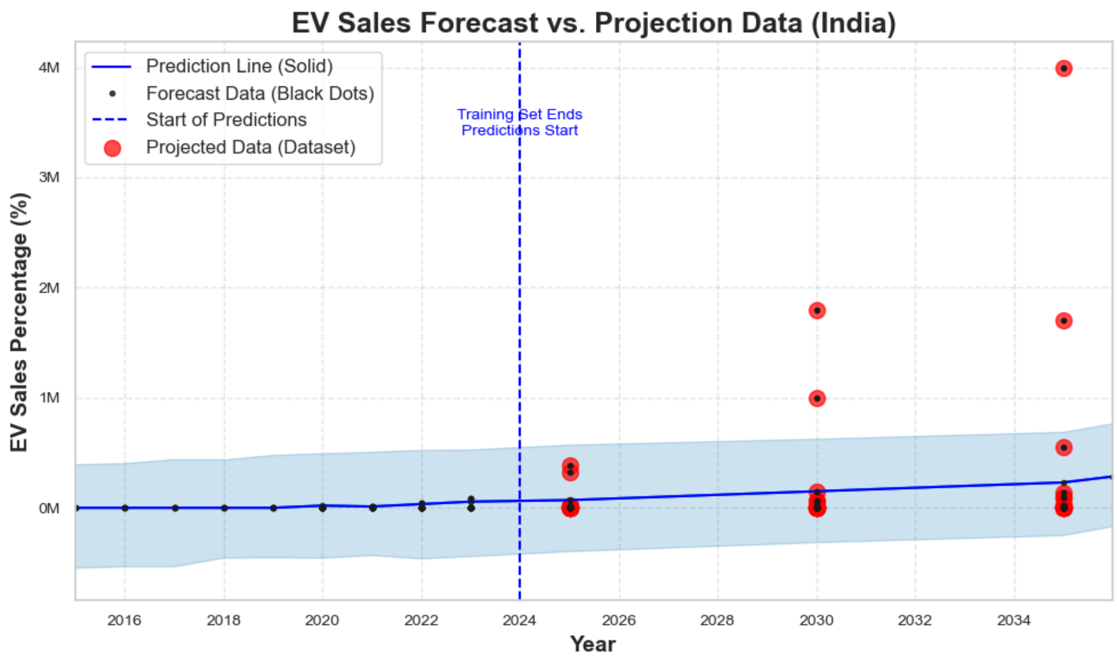
[177]:
```
#india ev sales
# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV SALES Data Preparation (Updated variable names)
df_india_sales_hist = df[(df['region'] == 'India') &
                         (df['parameter'] == 'EV sales') &
                         (df['year'] < 2024)]  # Historical data (before 2024)

df_india_sales_proj = df[(df['region'] == 'India') &
                         (df['parameter'] == 'EV sales') &
                         (df['year'] >= 2024) &
                         (df['year'] <= 2035)]  # Projection data (2024 to 2035)
```

```
# Prepare the projection data for plotting
df_india_sales_proj['time'] = pd.to_datetime(df_india_sales_proj['year'],␣
 ↪format='%Y')
df_india_sales_proj['percentage (%)'] = df_india_sales_proj['value']

# Combine historical and projection data for training
df_sales_combined = pd.concat([df_india_sales_hist, df_india_sales_proj])

df_sales_combined['time'] = pd.to_datetime(df_sales_combined['year'],␣
 ↪format='%Y')
df_sales_combined['percentage (%)'] = df_sales_combined['value']

# Step 2: Initialize and fit the Prophet model with yearly seasonality and␣
 ↪changepoint sensitivity
sales_model = Prophet(yearly_seasonality=True,  # Enable yearly seasonality
                      changepoint_prior_scale=0.05)  # Adjust changepoint␣
 ↪sensitivity
sales_model.fit(df_sales_combined.rename(columns={'time': 'ds', 'percentage␣
 ↪(%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_sales_years = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_sales = sales_model.make_future_dataframe(periods=future_sales_years,␣
 ↪freq='Y')

# Restrict future data to end at 2035
future_sales = future_sales[future_sales['ds'] <= pd.to_datetime('2035-12-31')]␣
 ↪ # Future data only until 2035

# Make predictions
forecast_sales = sales_model.predict(future_sales)

# Set a lower bound for predictions
forecast_sales['yhat'] = forecast_sales['yhat'].clip(lower=0)  # Clip negative␣
 ↪predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot = sales_model.plot(forecast_sales, xlabel='Year',␣
 ↪ylabel='Percentage (%)', ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
prediction_line, = ax.plot(forecast_sales['ds'], forecast_sales['yhat'],␣
 ↪color='blue', label='Prediction Line (Solid)')
```

```
# Step 5: Overlay the projection data on the same plot
projected_data = ax.scatter(df_india_sales_proj['time'],␣
 ↪df_india_sales_proj['percentage (%)'],
                             color='red', marker='o', s=100, label='Projected␣
 ↪Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points = ax.plot(forecast_sales['ds'], forecast_sales['yhat'], 'k.',␣
 ↪label='Forecast Data (Black Dots)', alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
prediction_start = ax.axvline(x=pd.to_datetime('2024-01-01'), color='blue',␣
 ↪linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Sales Forecast vs. Projection Data (India)', fontsize=18,␣
 ↪fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Sales Percentage (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')

def millions(x, pos):
    return f'{int(x / 1e6)}M'  # Convert to millions

formatter = FuncFormatter(millions)
plt.gca().yaxis.set_major_formatter(formatter)


# Step 8: Add the legend explicitly
legend_labels = ['Prediction Line (Solid)', 'Forecast Data (Black Dots)',␣
 ↪'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line, forecast_points[0], prediction_start,␣
 ↪projected_data], legend_labels, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-12-31'))

# Show the plot
```

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

# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
df_sales_cv = cross_validation(sales_model, initial='730 days', period='365␣
 ↪days', horizon='365 days')

# Calculate performance metrics
df_sales_p = performance_metrics(df_sales_cv)

# Output performance metrics
print(df_sales_p[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\1230162116.py:24:
SettingWithCopyWarning:
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
  df_india_sales_proj['time'] = pd.to_datetime(df_india_sales_proj['year'],
format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\1230162116.py:25:
SettingWithCopyWarning:
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
  df_india_sales_proj['percentage (%)'] = df_india_sales_proj['value']
01:19:12 - cmdstanpy - INFO - Chain [1] start processing
01:19:12 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
01:19:12 - cmdstanpy - INFO - Chain [1] start processing
01:19:12 - cmdstanpy - INFO - Chain [1] done processing
01:19:13 - cmdstanpy - INFO - Chain [1] start processing
01:19:13 - cmdstanpy - INFO - Chain [1] done processing
01:19:13 - cmdstanpy - INFO - Chain [1] start processing
01:19:13 - cmdstanpy - INFO - Chain [1] done processing
01:19:13 - cmdstanpy - INFO - Chain [1] start processing
01:19:13 - cmdstanpy - INFO - Chain [1] done processing
01:19:13 - cmdstanpy - INFO - Chain [1] start processing
01:19:13 - cmdstanpy - INFO - Chain [1] done processing
01:19:13 - cmdstanpy - INFO - Chain [1] start processing
01:19:13 - cmdstanpy - INFO - Chain [1] done processing
01:19:13 - cmdstanpy - INFO - Chain [1] start processing
01:19:13 - cmdstanpy - INFO - Chain [1] done processing
01:19:14 - cmdstanpy - INFO - Chain [1] start processing
01:19:14 - cmdstanpy - INFO - Chain [1] done processing
01:19:14 - cmdstanpy - INFO - Chain [1] start processing
01:19:14 - cmdstanpy - INFO - Chain [1] done processing
01:19:14 - cmdstanpy - INFO - Chain [1] start processing
01:19:14 - cmdstanpy - INFO - Chain [1] done processing
01:19:14 - cmdstanpy - INFO - Chain [1] start processing
01:19:14 - cmdstanpy - INFO - Chain [1] done processing
01:19:14 - cmdstanpy - INFO - Chain [1] start processing
01:19:15 - cmdstanpy - INFO - Chain [1] done processing
```

```
01:19:15 - cmdstanpy - INFO - Chain [1] start processing
01:19:15 - cmdstanpy - INFO - Chain [1] done processing
01:19:15 - cmdstanpy - INFO - Chain [1] start processing
01:19:15 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	364 days	879.197447	5.824250	1218.801498
1	365 days	113315.279152	220.312398	436153.070714

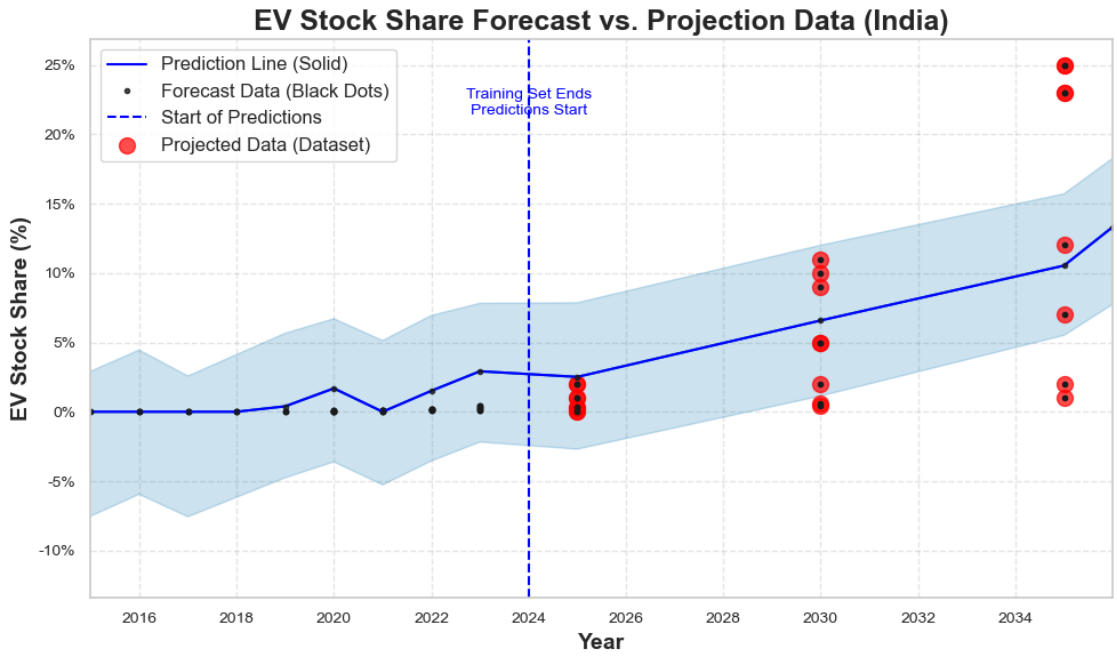
[24]:
```
# Required imports
# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV STOCK Data Preparation
df_india_stock_hist = df[(df['region'] == 'India') &
                         (df['parameter'] == 'EV stock share') &
                         (df['year'] < 2024)]  # Historical data (before 2024)

df_india_stock_proj = df[(df['region'] == 'India') &
                         (df['parameter'] == 'EV stock share') &
                         (df['year'] >= 2024) &
                         (df['year'] <= 2035)]  # Projection data (2024 to 2035)

# Prepare the projection data for plotting
df_india_stock_proj['time'] = pd.to_datetime(df_india_stock_proj['year'],␣
 ↪format='%Y')
df_india_stock_proj['percentage (%)'] = df_india_stock_proj['value']

# Combine historical and projection data for training
df_stock_combined = pd.concat([df_india_stock_hist, df_india_stock_proj])

df_stock_combined['time'] = pd.to_datetime(df_stock_combined['year'],␣
 ↪format='%Y')
df_stock_combined['percentage (%)'] = df_stock_combined['value']

# Step 2: Initialize and fit the Prophet model
stock_model = Prophet(yearly_seasonality=True,  # Enable yearly seasonality
                      changepoint_prior_scale=0.05)  # Adjust changepoint␣
 ↪sensitivity
stock_model.fit(df_stock_combined.rename(columns={'time': 'ds', 'percentage␣
 ↪(%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_stock_years = 2035 - 2023  # 12 years
future_stock = stock_model.make_future_dataframe(periods=future_stock_years,␣
 ↪freq='Y')
```

```
# Restrict future data to end at 2035
future_stock = future_stock[future_stock['ds'] <= pd.to_datetime('2035-12-31')]

# Make predictions
forecast_stock = stock_model.predict(future_stock)

# Clip negative predictions to zero
forecast_stock['yhat'] = forecast_stock['yhat'].clip(lower=0)

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot = stock_model.plot(forecast_stock, xlabel='Year',␣
 ↪ylabel='Percentage (%)', ax=ax)

# Extract the actual prediction line for the legend
prediction_line, = ax.plot(forecast_stock['ds'], forecast_stock['yhat'],␣
 ↪color='blue', label='Prediction Line (Solid)')

# Step 5: Overlay the projection data on the same plot
projected_data = ax.scatter(df_india_stock_proj['time'],␣
 ↪df_india_stock_proj['percentage (%)'],
                            color='red', marker='o', s=100, label='Projected␣
 ↪Data (Dataset)', alpha=0.7)

# Add black dots for actual forecast points
forecast_points = ax.plot(forecast_stock['ds'], forecast_stock['yhat'], 'k.',␣
 ↪label='Forecast Data (Black Dots)', alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
prediction_start = ax.axvline(x=pd.to_datetime('2024-01-01'), color='blue',␣
 ↪linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Stock Share Forecast vs. Projection Data (India)',␣
 ↪fontsize=18, fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Stock Share (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')
```

```
# Format the y-axis to show percentages
def percent_formatter(x, pos):
    return f'{x:.0f}%'  # Convert to percentage format

formatter = FuncFormatter(percent_formatter)
plt.gca().yaxis.set_major_formatter(formatter)

# Step 8: Add the legend explicitly
legend_labels = ['Prediction Line (Solid)', 'Forecast Data (Black Dots)',␣
 ↪'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line, forecast_points[0], prediction_start,␣
 ↪projected_data], legend_labels, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-12-31'))

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

# Step 9: Cross-Validation to assess model performance
df_stock_cv = cross_validation(stock_model, initial='730 days', period='365␣
 ↪days', horizon='365 days')

# Calculate performance metrics
df_stock_p = performance_metrics(df_stock_cv)

# Output performance metrics
print(df_stock_p[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\3479404947.py:24:
SettingWithCopyWarning:
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
  df_india_stock_proj['time'] = pd.to_datetime(df_india_stock_proj['year'],
format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\3479404947.py:25:
SettingWithCopyWarning:
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
  df_india_stock_proj['percentage (%)'] = df_india_stock_proj['value']
```

```
19:25:35 - cmdstanpy - INFO - Chain [1] start processing
19:25:36 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
19:25:37 - cmdstanpy - INFO - Chain [1] start processing
19:25:37 - cmdstanpy - INFO - Chain [1] done processing
19:25:37 - cmdstanpy - INFO - Chain [1] start processing
19:25:37 - cmdstanpy - INFO - Chain [1] done processing
19:25:37 - cmdstanpy - INFO - Chain [1] start processing
19:25:37 - cmdstanpy - INFO - Chain [1] done processing
19:25:37 - cmdstanpy - INFO - Chain [1] start processing
19:25:37 - cmdstanpy - INFO - Chain [1] done processing
19:25:38 - cmdstanpy - INFO - Chain [1] start processing
19:25:38 - cmdstanpy - INFO - Chain [1] done processing
19:25:38 - cmdstanpy - INFO - Chain [1] start processing
19:25:38 - cmdstanpy - INFO - Chain [1] done processing
```

```
19:25:38 - cmdstanpy - INFO - Chain [1] start processing
19:25:38 - cmdstanpy - INFO - Chain [1] done processing
19:25:38 - cmdstanpy - INFO - Chain [1] start processing
19:25:38 - cmdstanpy - INFO - Chain [1] done processing
19:25:38 - cmdstanpy - INFO - Chain [1] start processing
19:25:39 - cmdstanpy - INFO - Chain [1] done processing
19:25:39 - cmdstanpy - INFO - Chain [1] start processing
19:25:39 - cmdstanpy - INFO - Chain [1] done processing
19:25:39 - cmdstanpy - INFO - Chain [1] start processing
19:25:39 - cmdstanpy - INFO - Chain [1] done processing
19:25:39 - cmdstanpy - INFO - Chain [1] start processing
19:25:39 - cmdstanpy - INFO - Chain [1] done processing
19:25:40 - cmdstanpy - INFO - Chain [1] start processing
19:25:40 - cmdstanpy - INFO - Chain [1] done processing
19:25:40 - cmdstanpy - INFO - Chain [1] start processing
19:25:40 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	0.005287	1.031739	0.006557
1	364 days	0.018496	0.554439	0.023441
2	365 days	2.746751	0.813742	6.088792

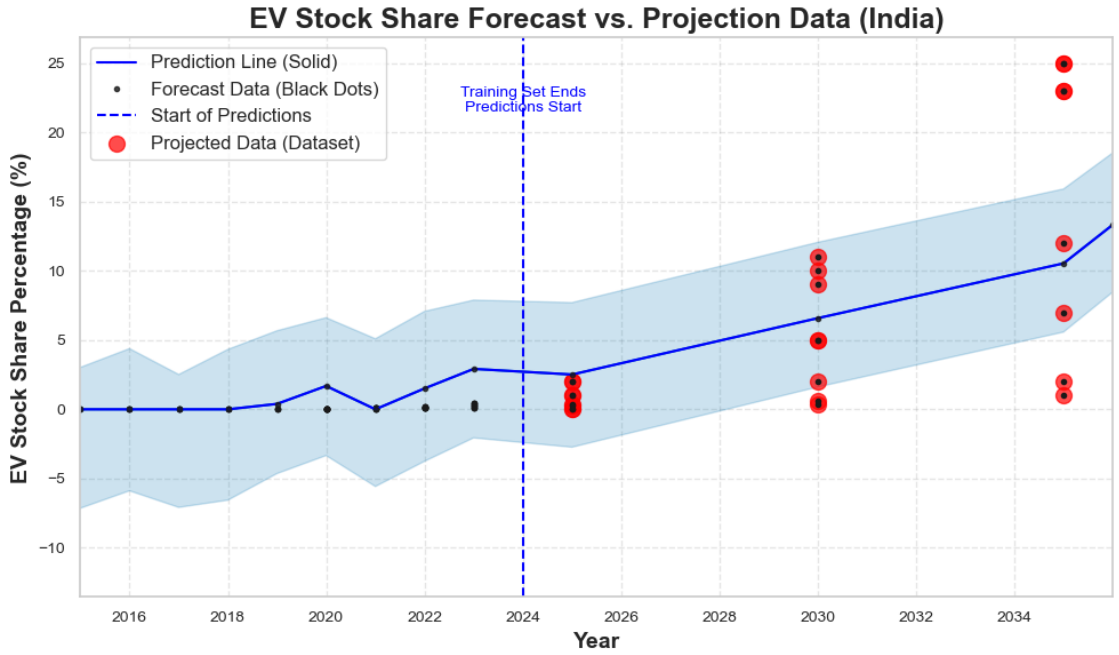
[94]:
```
# Set a Seaborn style for the plot
sns.set(style="whitegrid")

# EV STOCK SHARE Data Preparation (Updated variable names)
df_india_stock_hist = df[(df['region'] == 'India') &
                         (df['parameter'] == 'EV stock share') &
                         (df['year'] < 2024)]  # Historical data (before 2024)

df_india_stock_proj = df[(df['region'] == 'India') &
                         (df['parameter'] == 'EV stock share') &
                         (df['year'] >= 2024) &
                         (df['year'] <= 2035)]  # Projection data (2024 to 2035)

# Prepare the projection data for plotting
df_india_stock_proj['time'] = pd.to_datetime(df_india_stock_proj['year'],␣
 ↪format='%Y')
df_india_stock_proj['percentage (%)'] = df_india_stock_proj['value']

# Combine historical and projection data for training
df_stock_combined = pd.concat([df_india_stock_hist, df_india_stock_proj])

df_stock_combined['time'] = pd.to_datetime(df_stock_combined['year'],␣
 ↪format='%Y')
df_stock_combined['percentage (%)'] = df_stock_combined['value']
```

```
# Step 2: Initialize and fit the Prophet model with yearly seasonality and␣
 ↪changepoint sensitivity
stock_model = Prophet(yearly_seasonality=True,  # Enable yearly seasonality
                      changepoint_prior_scale=0.05)  # Adjust changepoint␣
 ↪sensitivity
stock_model.fit(df_stock_combined.rename(columns={'time': 'ds', 'percentage␣
 ↪(%)': 'y'}))

# Step 3: Create a future DataFrame for predictions until 2035
future_stock_years = 2035 - 2023  # 12 years

# Create future dataframe with only periods needed until 2035
future_stock = stock_model.make_future_dataframe(periods=future_stock_years,␣
 ↪freq='Y')

# Restrict future data to end at 2035
future_stock = future_stock[future_stock['ds'] <= pd.to_datetime('2035-12-31')]␣
 ↪ # Future data only until 2035

# Make predictions
forecast_stock = stock_model.predict(future_stock)

# Set a lower bound for predictions
forecast_stock['yhat'] = forecast_stock['yhat'].clip(lower=0)  # Clip negative␣
 ↪predictions to zero

# Step 4: Plot the forecast
fig, ax = plt.subplots(figsize=(10, 6))
forecast_plot = stock_model.plot(forecast_stock, xlabel='Year',␣
 ↪ylabel='Percentage (%)', ax=ax)

# Extract the actual prediction line for the solid blue line in the legend
prediction_line, = ax.plot(forecast_stock['ds'], forecast_stock['yhat'],␣
 ↪color='blue', label='Prediction Line (Solid)')

# Step 5: Overlay the projection data on the same plot
projected_data = ax.scatter(df_india_stock_proj['time'],␣
 ↪df_india_stock_proj['percentage (%)'],
                            color='red', marker='o', s=100, label='Projected␣
 ↪Data (Dataset)', alpha=0.7)

# Add the black dots for actual forecast points
forecast_points = ax.plot(forecast_stock['ds'], forecast_stock['yhat'], 'k.',␣
 ↪label='Forecast Data (Black Dots)', alpha=0.8)

# Step 6: Add a vertical line indicating the start of predictions
```

```
prediction_start = ax.axvline(x=pd.to_datetime('2024-01-01'), color='blue',␣
 ↪linestyle='--', label='Start of Predictions')

# Annotate the vertical line
ax.text(pd.to_datetime('2024-01-01'), ax.get_ylim()[1] * 0.8, 'Training Set␣
 ↪Ends\nPredictions Start',
        color='blue', fontsize=10, ha='center')

# Step 7: Customize the plot
ax.set_title('EV Stock Share Forecast vs. Projection Data (India)',␣
 ↪fontsize=18, fontweight='bold')
ax.set_xlabel('Year', fontsize=14, fontweight='bold')
ax.set_ylabel('EV Stock Share Percentage (%)', fontsize=14, fontweight='bold')
ax.tick_params(axis='x', labelsize=10)
ax.tick_params(axis='y', labelsize=10)
ax.grid(True, linestyle='--')

# Step 8: Add the legend explicitly
legend_labels = ['Prediction Line (Solid)', 'Dataset Value (Black Dots)',␣
 ↪'Start of Predictions', 'Projected Data (Dataset)']
ax.legend([prediction_line, forecast_points[0], prediction_start,␣
 ↪projected_data], legend_labels, loc='upper left', fontsize=12)

# Limit the x-axis to show data only until 2035
ax.set_xlim(pd.to_datetime('2015-01-01'), pd.to_datetime('2035-12-31'))

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

# Step 9: Cross-Validation to assess model performance

# Perform cross-validation on the model
df_stock_cv = cross_validation(stock_model, initial='730 days', period='365␣
 ↪days', horizon='365 days')

# Calculate performance metrics
df_stock_p = performance_metrics(df_stock_cv)

# Output performance metrics
print(df_stock_p[['horizon', 'mae', 'mape', 'rmse']])
```

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\3821631950.py:23:
SettingWithCopyWarning:
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
  df_india_stock_proj['time'] = pd.to_datetime(df_india_stock_proj['year'],
format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_20220\3821631950.py:24:
SettingWithCopyWarning:
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
  df_india_stock_proj['percentage (%)'] = df_india_stock_proj['value']
23:19:39 - cmdstanpy - INFO - Chain [1] start processing
23:19:39 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:72:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  fcst_t = fcst['ds'].dt.to_pydatetime()
C:\Users\Abhilove Goyal\anaconda3\Lib\site-packages\prophet\plot.py:73:
FutureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated,
in a future version this will return a Series containing python datetime objects
instead of an ndarray. To retain the old behavior, call `np.array` on the result
  ax.plot(m.history['ds'].dt.to_pydatetime(), m.history['y'], 'k.',
```




```
  0%|          | 0/14 [00:00<?, ?it/s]
```

```
23:19:40 - cmdstanpy - INFO - Chain [1] start processing
23:19:40 - cmdstanpy - INFO - Chain [1] done processing
23:19:41 - cmdstanpy - INFO - Chain [1] start processing
23:19:41 - cmdstanpy - INFO - Chain [1] done processing
23:19:41 - cmdstanpy - INFO - Chain [1] start processing
23:19:41 - cmdstanpy - INFO - Chain [1] done processing
23:19:41 - cmdstanpy - INFO - Chain [1] start processing
23:19:41 - cmdstanpy - INFO - Chain [1] done processing
23:19:41 - cmdstanpy - INFO - Chain [1] start processing
23:19:41 - cmdstanpy - INFO - Chain [1] done processing
23:19:41 - cmdstanpy - INFO - Chain [1] start processing
23:19:41 - cmdstanpy - INFO - Chain [1] done processing
23:19:42 - cmdstanpy - INFO - Chain [1] start processing
23:19:42 - cmdstanpy - INFO - Chain [1] done processing
23:19:42 - cmdstanpy - INFO - Chain [1] start processing
23:19:42 - cmdstanpy - INFO - Chain [1] done processing
23:19:42 - cmdstanpy - INFO - Chain [1] start processing
23:19:42 - cmdstanpy - INFO - Chain [1] done processing
23:19:42 - cmdstanpy - INFO - Chain [1] start processing
23:19:42 - cmdstanpy - INFO - Chain [1] done processing
23:19:43 - cmdstanpy - INFO - Chain [1] start processing
23:19:43 - cmdstanpy - INFO - Chain [1] done processing
23:19:43 - cmdstanpy - INFO - Chain [1] start processing
23:19:43 - cmdstanpy - INFO - Chain [1] done processing
23:19:43 - cmdstanpy - INFO - Chain [1] start processing
23:19:43 - cmdstanpy - INFO - Chain [1] done processing
23:19:44 - cmdstanpy - INFO - Chain [1] start processing
23:19:44 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	363 days	0.005287	1.031739	0.006557
1	364 days	0.018496	0.554439	0.023441
2	365 days	2.746751	0.813742	6.088792

[25]:
```python
# Assuming 'df' is your original DataFrame with columns: region, parameter, year, and value
# Filter for EV stock share
df_india_stockH = df[(df['region'] == 'India') &
                     (df['parameter'] == 'EV stock share') &
                     (df['year'] < 2024)]  # Historical data

df_india_stockPro = df[(df['region'] == 'India') &
                       (df['parameter'] == 'EV stock share') &
                       (df['year'] >= 2024) &
                       (df['year'] <= 2035)]  # Projection data

# Prepare the projection data for plotting
df_india_stockPro['ds'] = pd.to_datetime(df_india_stockPro['year'], format='%Y')
```

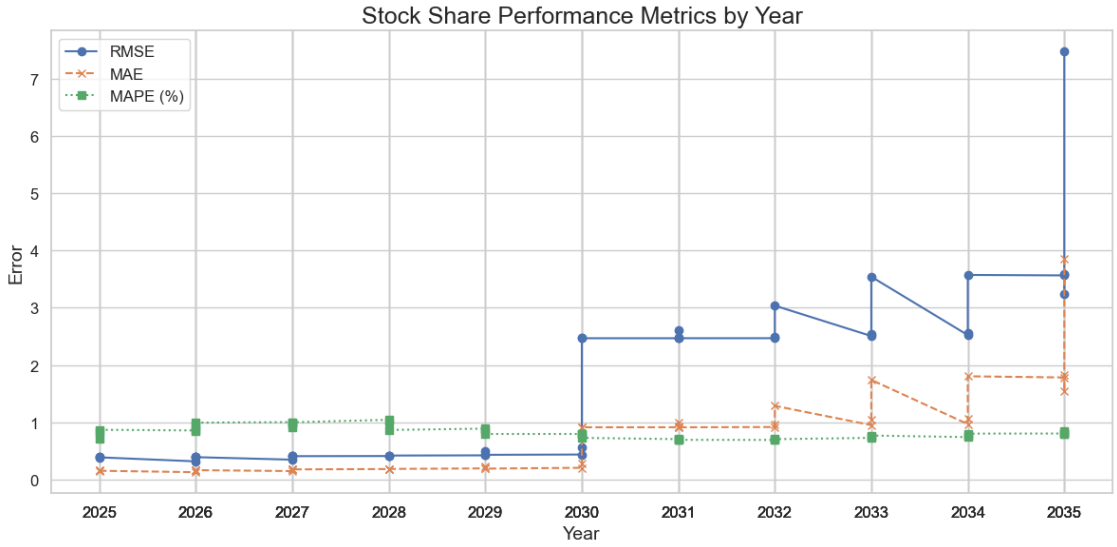
```
df_india_stockPro['y'] = df_india_stockPro['value']

# Combine historical and projection data for training
df_combined_stock = pd.concat([df_india_stockH, df_india_stockPro])

df_combined_stock['ds'] = pd.to_datetime(df_combined_stock['year'], format='%Y')
df_combined_stock['y'] = df_combined_stock['value']

# Step 2: Initialize and fit the Prophet model
model = Prophet(yearly_seasonality=True, changepoint_prior_scale=0.05)
model.fit(df_combined_stock)

# Step 3: Cross-Validation to assess model performance with different horizons
# Calculate the total number of days from 2024 to 2035
horizon_days = (pd.to_datetime('2035-01-01') - pd.to_datetime('2024-01-01')).
 ↪days

# Perform cross-validation with horizon set to the calculated days
df_cv = cross_validation(model, initial='730 days', period='365 days',␣
 ↪horizon=f'{horizon_days} days')

# Step 4: Calculate performance metrics
df_p = performance_metrics(df_cv)

# Generate year labels based on the prediction horizons
start_year = 2024  # Starting year for predictions

# Convert horizons from days to years
years = [start_year + (horizon.days // 365) for horizon in df_p['horizon']]

# Output performance metrics
print(df_p[['horizon', 'mae', 'mape', 'rmse']])

# Optional: Plot the performance metrics with years on the x-axis
plt.figure(figsize=(12, 6))
plt.plot(years, df_p['rmse'], label='RMSE', marker='o', linestyle='-')
plt.plot(years, df_p['mae'], label='MAE', marker='x', linestyle='--')
plt.plot(years, df_p['mape'], label='MAPE (%)', marker='s', linestyle=':')
plt.title('Stock Share Performance Metrics by Year', fontsize=18)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Error', fontsize=14)
plt.xticks(years, fontsize=12)  # Set x-ticks to show years
plt.yticks(fontsize=12)

# Add legend with adjusted fontsize
plt.legend(fontsize=12, loc='upper left')  # Specify location if needed
plt.grid(True)
```

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

```
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1668482084.py:19:
SettingWithCopyWarning:
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
  df_india_stockPro['ds'] = pd.to_datetime(df_india_stockPro['year'],
format='%Y')
C:\Users\Abhilove Goyal\AppData\Local\Temp\ipykernel_11188\1668482084.py:20:
SettingWithCopyWarning:
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
  df_india_stockPro['y'] = df_india_stockPro['value']
19:25:48 - cmdstanpy - INFO - Chain [1] start processing
19:25:48 - cmdstanpy - INFO - Chain [1] done processing
  0%|          | 0/13 [00:00<?, ?it/s]
19:25:49 - cmdstanpy - INFO - Chain [1] start processing
19:25:49 - cmdstanpy - INFO - Chain [1] done processing
19:25:49 - cmdstanpy - INFO - Chain [1] start processing
19:25:49 - cmdstanpy - INFO - Chain [1] done processing
19:25:49 - cmdstanpy - INFO - Chain [1] start processing
19:25:49 - cmdstanpy - INFO - Chain [1] done processing
19:25:49 - cmdstanpy - INFO - Chain [1] start processing
19:25:49 - cmdstanpy - INFO - Chain [1] done processing
19:25:49 - cmdstanpy - INFO - Chain [1] start processing
19:25:50 - cmdstanpy - INFO - Chain [1] done processing
19:25:50 - cmdstanpy - INFO - Chain [1] start processing
19:25:50 - cmdstanpy - INFO - Chain [1] done processing
19:25:50 - cmdstanpy - INFO - Chain [1] start processing
19:25:50 - cmdstanpy - INFO - Chain [1] done processing
19:25:50 - cmdstanpy - INFO - Chain [1] start processing
19:25:50 - cmdstanpy - INFO - Chain [1] done processing
19:25:51 - cmdstanpy - INFO - Chain [1] start processing
19:25:51 - cmdstanpy - INFO - Chain [1] done processing
19:25:51 - cmdstanpy - INFO - Chain [1] start processing
19:25:51 - cmdstanpy - INFO - Chain [1] done processing
19:25:51 - cmdstanpy - INFO - Chain [1] start processing
19:25:51 - cmdstanpy - INFO - Chain [1] done processing
19:25:51 - cmdstanpy - INFO - Chain [1] start processing
19:25:52 - cmdstanpy - INFO - Chain [1] done processing
```

```
19:25:52 - cmdstanpy - INFO - Chain [1] start processing
19:25:52 - cmdstanpy - INFO - Chain [1] done processing
```

	horizon	mae	mape	rmse
0	366 days	0.155263	0.704311	0.388053
1	728 days	0.155027	0.824703	0.388029
2	729 days	0.152638	0.868349	0.386826
3	730 days	0.127366	0.856721	0.316604
4	731 days	0.165239	0.844171	0.391225
5	1093 days	0.165016	0.949745	0.391200
6	1094 days	0.162629	0.993022	0.390189
7	1095 days	0.147361	1.002247	0.346089
8	1096 days	0.176610	0.919287	0.407416
9	1458 days	0.176795	0.973372	0.407423
10	1459 days	0.176726	0.999890	0.407431
11	1460 days	0.182246	1.042837	0.409640
12	1461 days	0.184707	1.012913	0.418130
13	1824 days	0.184131	0.865038	0.418084
14	1825 days	0.193905	0.888920	0.421635
15	1826 days	0.230571	0.868698	0.495209
16	2189 days	0.189799	0.795054	0.430831
17	2190 days	0.204285	0.795583	0.435905
18	2191 days	0.274123	0.791612	0.561873
19	2192 days	0.912132	0.789591	2.467550
20	2554 days	0.912085	0.727049	2.467547
21	2555 days	0.912428	0.704691	2.467656
22	2556 days	0.925200	0.701676	2.472397
23	2557 days	1.001363	0.701207	2.613185
24	2919 days	0.911434	0.692663	2.468057
25	2920 days	0.916564	0.691069	2.468360
26	2921 days	0.964765	0.692612	2.485879
27	2922 days	1.286662	0.704048	3.037307
28	3285 days	0.949741	0.727647	2.506370
29	3286 days	1.046950	0.745976	2.540635
30	3287 days	1.739658	0.767646	3.542338
31	3650 days	0.967899	0.740399	2.523209
32	3651 days	1.065515	0.761225	2.557073
33	3652 days	1.802666	0.801401	3.573682
34	4015 days	1.782618	0.803248	3.565572
35	4016 days	1.543177	0.793197	3.234829
36	4017 days	1.825948	0.811226	3.590003
37	4018 days	3.849092	0.838198	7.474666



The prediciton graphs are not made for all parameters, for eg-> Charging points, stock.

Limitation: -The dataset doesn't have info about 2w/3w vehicles -Predicting ev sales and accuracy

You can also find this dataset at IEA, i used IEA outlook global research as reference

[ ]:

[ ]: