Table of Contents

[0.0.1 Key Observations: 24](#_Toc202619647)

[0.0.2 Background Explanation: 25](#_Toc202619648)

[0.1 Hourly Normalized EV Charging Demand by Season 31](#_Toc202619649)

[0.1.1 Visualization Details: 31](#_Toc202619650)

[0.1.2 Customizations: 32](#_Toc202619651)

[0.1.3 Purpose: 32](#_Toc202619652)

[0.2 Hourly Normalized EV Charging Demand (Aggregated) 33](#_Toc202619653)

[0.2.1 Visualization Details: 33](#_Toc202619654)

[0.2.2 Customizations: 34](#_Toc202619655)

[0.2.3 Purpose: 34](#_Toc202619656)

[1 SHAP Analysis for EV Charging Demand Prediction 35](#_Toc202619657)

[1.1 Data Preprocessing 35](#_Toc202619658)

[1.1.1 Features & Target 35](#_Toc202619659)

[2 EV Charging Demand Prediction with SHAP Analysis 36](#_Toc202619660)

[2.1 Data Preparation and Feature Engineering 36](#_Toc202619661)

[2.2 Data Preprocessing Pipeline 36](#_Toc202619662)

[2.3 Feature Standardization and Data Splitting 36](#_Toc202619663)

[2.4 Model Training and SHAP Explainer Setup 36](#_Toc202619664)

[2.5 SHAP Visualization Analysis 37](#_Toc202619665)

[2.6 SHAP Values Data Frame Creation 37](#_Toc202619666)

[3 SHAP Force Plot Visualization 42](#_Toc202619667)

[3.1 Interactive SHAP Force Plot Setup 42](#_Toc202619668)

[3.2 Understanding the Force Plot 42](#_Toc202619669)

[4 SHAP Summary Plot Analysis 43](#_Toc202619670)

[4.1 Interpreting the Summary Plot 43](#_Toc202619671)

[5 Neural Network Model for EV Charging Demand Prediction 45](#_Toc202619672)

[5.1 Library Imports and Dependencies 45](#_Toc202619673)

[5.2 Feature Selection and Target Definition 45](#_Toc202619674)

[5.3 Data Preprocessing Pipeline 45](#_Toc202619675)

[5.4 Train-Test Split Configuration 46](#_Toc202619676)

[5.5 Neural Network Architecture and Training 46](#_Toc202619677)

[5.6 Model Evaluation and Metrics 46](#_Toc202619678)

[5.7 Training Progress Visualization 46](#_Toc202619679)

[5.8 Model Performance Summary 47](#_Toc202619680)

[5.9 Prediction Comparison Visualization 47](#_Toc202619681)

[6 Decision Tree Classification for Day Period Prediction 55](#_Toc202619682)

[6.1 Library Imports and Dependencies 55](#_Toc202619683)

[6.2 Feature Selection and Target Definition 55](#_Toc202619684)

[6.3 Train-Test Split Configuration 55](#_Toc202619685)

[6.4 Decision Tree Model Configuration 56](#_Toc202619686)

[6.5 Model Training and Prediction 56](#_Toc202619687)

[6.6 Accuracy Assessment 56](#_Toc202619688)

[6.7 Confusion Matrix Analysis 56](#_Toc202619689)

[6.8 Decision Tree Visualization 57](#_Toc202619690)

[6.8.1 What does this Decision Tree outcome suggest? 61](#_Toc202619691)

[6.8.2 What do the different colors in the Decision Tree mean? 62](#_Toc202619692)

[6.8.3 Why is the Gini Index used and what is its purpose? 62](#_Toc202619693)

[6.8.4 Cricket Analogy 62](#_Toc202619694)

[7 Decision Tree Classification for Grid Availability Predictio 64](#_Toc202619695)

[7.1 Feature Selection and Target Definition 64](#_Toc202619696)

[7.2 Train-Test Split Configuration 65](#_Toc202619697)

[7.3 Decision Tree Model Configuration 65](#_Toc202619698)

[7.4 Confusion Matrix Analysis 65](#_Toc202619699)

[7.5 Decision Tree Visualization 65](#_Toc202619700)

'''  
Suppresses all warning messages in the notebook to provide cleaner output during execution.  
This is useful when running code that may generate non-critical warnings, ensuring that the notebook output remains uncluttered and easier to read.  
'''  
  
import warnings  
warnings.filterwarnings('ignore')

'''  
This cell imports the 'os' module and prints the current working directory.  
It helps to verify the file path context for reading or writing files in the notebook environment.  
'''  
import os  
print(os.getcwd())

g:\DIYguru\Data-Science-and-Engineering-Analytics\Projects\Main\_Project\_ML\EV Charging Patterns

'''  
Created a function 'load\_data\_excel' to load excel into python  
'''  
  
import pandas as pd  
import numpy as np  
import os  
  
def load\_data\_csv(file\_path):  
 """  
 Load data from an csv file and return a DataFrame.  
 """  
 if not os.path.exists(file\_path):  
 raise FileNotFoundError(f"The file {file\_path} does not exist.")  
   
 df = pd.read\_csv(file\_path)  
 return df

'''  
Loads EV charging station datasets from CSV files for two different stations using the custom load\_data\_csv function.  
This prepares the data for further analysis and processing in subsequent steps.  
'''  
  
# https://www.kaggle.com/datasets/datasetengineer/ev-charging-station-data-california-region?utm\_source=chatgpt.com  
  
#df1 = load\_data\_csv(r"G:\DIYguru\Notes and Sample Data\EV Charging Station Data- California Region\Charging station\_A\_Calif.csv")  
df1 = load\_data\_csv(r"G:\DIYguru\Notes and Sample Data\EV Charging Station Data- California Region\Charging station\_B\_\_Calif.csv")  
df2 = load\_data\_csv(r"G:\DIYguru\Notes and Sample Data\EV Charging Station Data- California Region\Charging station\_C\_\_Calif.csv")

'''  
Concatenates the two dataframes, df1 and df2, into a single dataframe df.  
This operation combines the datasets from two different EV charging stations into one unified dataset for further analysis.  
The index is reset to ensure a continuous sequence after concatenation.  
'''  
  
# concatenate the dataframes  
df = pd.concat([df1, df2], ignore\_index=True)

'''  
Checked unique values in both the dataframe  
'''  
  
for i in df.columns:  
 print(f"Unique values in column '{i}':")  
 print(df[i].unique(),"\n")

Unique values in column 'Date':  
['2021-01-01' '2021-01-02' '2021-01-03' ... '2024-05-29' '2024-05-30'  
 '2024-05-31']   
  
Unique values in column 'Time':  
['00:00:00' '01:00:00' '02:00:00' '03:00:00' '04:00:00' '05:00:00'  
 '06:00:00' '07:00:00' '08:00:00' '09:00:00' '10:00:00' '11:00:00'  
 '12:00:00' '13:00:00' '14:00:00' '15:00:00' '16:00:00' '17:00:00'  
 '18:00:00' '19:00:00' '20:00:00' '21:00:00' '22:00:00' '23:00:00']   
  
Unique values in column 'EV Charging Demand (kW)':  
[0.11236204 0.28521429 0.21959818 ... 0.23297838 0.07779743 0.2481701 ]   
  
Unique values in column 'Solar Energy Production (kW)':  
[0.12538804 0.05269671 0.10503542 ... 0.04761178 0.05229399 0.00880412]   
  
Unique values in column 'Wind Energy Production (kW)':  
[0.00910519 0.10758936 0.0439957 ... 0.13834387 0.27850439 0.18543791]   
  
Unique values in column 'Electricity Price ($/kWh)':  
[0.13731049 0.12510459 0.10666085 ... 0.1780538 0.17047302 0.10660294]   
  
Unique values in column 'Grid Availability':  
['Available' 'Unavailable']   
  
Unique values in column 'Weather Conditions':  
['Partly Cloudy' 'Sunny' 'Cloudy' 'Clear' 'Rainy']   
  
Unique values in column 'Battery Storage (kWh)':  
[16.53240777 39.10693019 6.11269143 ... 7.73881941 45.4788398  
 15.01814069]   
  
Unique values in column 'Charging Station Capacity (kW)':  
[21.76342185 31.21502758 46.48911616 ... 17.60406674 8.63806753  
 14.27427145]   
  
Unique values in column 'EV Charging Efficiency (%)':  
[97.3263759 88.54691271 89.87297148 ... 80.31515304 98.46691092  
 99.89052108]   
  
Unique values in column 'Number of EVs Charging':  
[6 7 4 5 9 8 2 1 3]   
  
Unique values in column 'Peak Demand (kW)':  
[0.15168033 0.57343266 0.97848217 ... 0.97122722 0.9229278 0.12162385]   
  
Unique values in column 'Renewable Energy Usage (%)':  
[25.03906598 55.64989943 79.97078295 ... 86.82746215 84.76931863  
 77.48112723]   
  
Unique values in column 'Grid Stability Index':  
[0.73114739 1.494387 1.10929346 ... 1.24461115 0.56112754 1.07690558]   
  
Unique values in column 'Carbon Emissions (kgCO2/kWh)':  
[0.27494407 0.48125091 0.14607863 ... 0.35722855 0.28634342 0.35170394]   
  
Unique values in column 'Power Outages (hours)':  
[1.88920926 0.2773707 0.64264441 ... 1.41103498 1.61914218 0.91412292]   
  
Unique values in column 'Energy Savings ($)':  
[4.56258096 0.21510366 0.02996864 ... 4.40542951 0.76756544 3.03338893]   
  
Unique values in column 'Total Renewable Energy Production (kW)':  
[0.13449323 0.16028607 0.14903113 ... 0.18595565 0.33079839 0.19424203]   
  
Unique values in column 'Effective Charging Capacity (kW)':  
[21.18154976 27.63994322 41.78115011 ... 14.13873314 8.50563826  
 14.25864413]   
  
Unique values in column 'Adjusted Charging Demand (kW)':  
[0.0281344 0.15872147 0.17561439 ... 0.20228921 0.06594835 0.19228499]   
  
Unique values in column 'Net Energy Cost ($)':  
[0.01542849 0.03568162 0.02342253 ... 0.04148269 0.01326236 0.02645566]   
  
Unique values in column 'Carbon Footprint Reduction (kgCO2)':  
[0.02315789 0.06087479 0.00642509 ... 0.01096305 0.00339291 0.01965501]   
  
Unique values in column 'Renewable Energy Efficiency':  
[0.00634955 0.00579907 0.00356695 ... 0.01315221 0.03889166 0.01362276]

'''  
Ensures that the 'Date' column in the dataframe is in datetime format.  
This conversion is necessary for accurate time-based analysis and prevents errors in downstream operations that require datetime types.  
Invalid parsing will result in NaT (Not a Time) for problematic entries.  
'''  
  
# fix date format for date column Date  
df['Date'] = pd.to\_datetime(df['Date'], errors='coerce')

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 59810 entries, 0 to 59809  
Data columns (total 24 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Date 59810 non-null datetime64[ns]  
 1 Time 59810 non-null object   
 2 EV Charging Demand (kW) 59810 non-null float64   
 3 Solar Energy Production (kW) 59810 non-null float64   
 4 Wind Energy Production (kW) 59810 non-null float64   
 5 Electricity Price ($/kWh) 59810 non-null float64   
 6 Grid Availability 59810 non-null object   
 7 Weather Conditions 59810 non-null object   
 8 Battery Storage (kWh) 59810 non-null float64   
 9 Charging Station Capacity (kW) 59810 non-null float64   
 10 EV Charging Efficiency (%) 59810 non-null float64   
 11 Number of EVs Charging 59810 non-null int64   
 12 Peak Demand (kW) 59810 non-null float64   
 13 Renewable Energy Usage (%) 59810 non-null float64   
 14 Grid Stability Index 59810 non-null float64   
 15 Carbon Emissions (kgCO2/kWh) 59810 non-null float64   
 16 Power Outages (hours) 59810 non-null float64   
 17 Energy Savings ($) 59810 non-null float64   
 18 Total Renewable Energy Production (kW) 59810 non-null float64   
 19 Effective Charging Capacity (kW) 59810 non-null float64   
 20 Adjusted Charging Demand (kW) 59810 non-null float64   
 21 Net Energy Cost ($) 59810 non-null float64   
 22 Carbon Footprint Reduction (kgCO2) 59810 non-null float64   
 23 Renewable Energy Efficiency 59810 non-null float64   
dtypes: datetime64[ns](1), float64(19), int64(1), object(3)  
memory usage: 11.0+ MB

df.head()

Date Time EV Charging Demand (kW) Solar Energy Production (kW) \  
0 2021-01-01 00:00:00 0.112362 0.125388   
1 2021-01-01 01:00:00 0.285214 0.052697   
2 2021-01-01 02:00:00 0.219598 0.105035   
3 2021-01-01 03:00:00 0.179598 0.073839   
4 2021-01-01 04:00:00 0.046806 0.068614   
  
 Wind Energy Production (kW) Electricity Price ($/kWh) Grid Availability \  
0 0.009105 0.137310 Available   
1 0.107589 0.125105 Available   
2 0.043996 0.106661 Available   
3 0.275727 0.072209 Available   
4 0.059824 0.091090 Available   
  
 Weather Conditions Battery Storage (kWh) Charging Station Capacity (kW) \  
0 Partly Cloudy 16.532408 21.763422   
1 Sunny 39.106930 31.215028   
2 Cloudy 6.112691 46.489116   
3 Partly Cloudy 30.041088 49.675029   
4 Partly Cloudy 45.085422 21.166182   
  
 ... Grid Stability Index Carbon Emissions (kgCO2/kWh) \  
0 ... 0.731147 0.274944   
1 ... 1.494387 0.481251   
2 ... 1.109293 0.146079   
3 ... 0.847219 0.475255   
4 ... 1.452466 0.319261   
  
 Power Outages (hours) Energy Savings ($) \  
0 1.889209 4.562581   
1 0.277371 0.215104   
2 0.642644 0.029969   
3 0.546680 1.384950   
4 1.935850 1.872170   
  
 Total Renewable Energy Production (kW) Effective Charging Capacity (kW) \  
0 0.134493 21.181550   
1 0.160286 27.639943   
2 0.149031 41.781150   
3 0.349567 46.078097   
4 0.128438 17.733986   
  
 Adjusted Charging Demand (kW) Net Energy Cost ($) \  
0 0.028134 0.015428   
1 0.158721 0.035682   
2 0.175614 0.023423   
3 0.004072 0.012969   
4 0.045315 0.004264   
  
 Carbon Footprint Reduction (kgCO2) Renewable Energy Efficiency   
0 0.023158 0.006350   
1 0.060875 0.005799   
2 0.006425 0.003567   
3 0.083420 0.007586   
4 0.000476 0.007242   
  
[5 rows x 24 columns]

from sklearn.preprocessing import MinMaxScaler  
  
# Normalize (Min-Max)  
scaler\_minmax = MinMaxScaler()  
df['EV Charging Efficiency (%) - Normalized'] = scaler\_minmax.fit\_transform(df[['EV Charging Efficiency (%)']]).round(2)

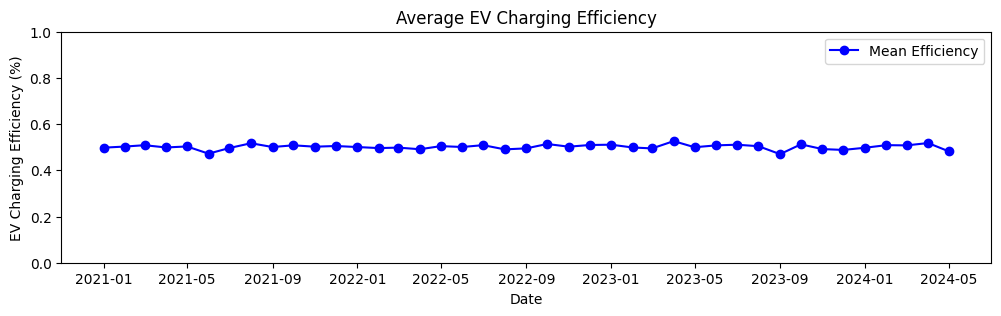
df\_charging\_efficiency = df.groupby(df['Date'].dt.to\_period('M'))['EV Charging Efficiency (%)'].mean().reset\_index()  
df\_charging\_efficiency

Date EV Charging Efficiency (%)  
0 2021-01 89.958806  
1 2021-02 90.056477  
2 2021-03 90.175206  
3 2021-04 89.981847  
4 2021-05 90.069777  
5 2021-06 89.442198  
6 2021-07 89.931319  
7 2021-08 90.345421  
8 2021-09 90.017416  
9 2021-10 90.166764  
10 2021-11 90.045631  
11 2021-12 90.103390  
12 2022-01 90.015701  
13 2022-02 89.926263  
14 2022-03 89.963057  
15 2022-04 89.819643  
16 2022-05 90.094384  
17 2022-06 90.022538  
18 2022-07 90.180894  
19 2022-08 89.811577  
20 2022-09 89.901928  
21 2022-10 90.278914  
22 2022-11 90.043838  
23 2022-12 90.191614  
24 2023-01 90.219238  
25 2023-02 89.976901  
26 2023-03 89.895897  
27 2023-04 90.529745  
28 2023-05 89.998251  
29 2023-06 90.157921  
30 2023-07 90.217578  
31 2023-08 90.087278  
32 2023-09 89.401034  
33 2023-10 90.250923  
34 2023-11 89.835838  
35 2023-12 89.765036  
36 2024-01 89.951231  
37 2024-02 90.176650  
38 2024-03 90.160447  
39 2024-04 90.361861  
40 2024-05 89.635407

df\_charging\_efficiency = df.groupby(df['Date'].dt.to\_period('M'))['EV Charging Efficiency (%) - Normalized'].mean().reset\_index()  
df\_charging\_efficiency

Date EV Charging Efficiency (%) - Normalized  
0 2021-01 0.498051  
1 2021-02 0.502976  
2 2021-03 0.508737  
3 2021-04 0.498958  
4 2021-05 0.503333  
5 2021-06 0.472111  
6 2021-07 0.496546  
7 2021-08 0.517245  
8 2021-09 0.500847  
9 2021-10 0.508239  
10 2021-11 0.502319  
11 2021-12 0.505242  
12 2022-01 0.500820  
13 2022-02 0.496220  
14 2022-03 0.498306  
15 2022-04 0.491236  
16 2022-05 0.504731  
17 2022-06 0.501042  
18 2022-07 0.508938  
19 2022-08 0.490685  
20 2022-09 0.495028  
21 2022-10 0.513992  
22 2022-11 0.502319  
23 2022-12 0.509543  
24 2023-01 0.510981  
25 2023-02 0.498839  
26 2023-03 0.494866  
27 2023-04 0.526444  
28 2023-05 0.499987  
29 2023-06 0.507931  
30 2023-07 0.510927  
31 2023-08 0.504395  
32 2023-09 0.470014  
33 2023-10 0.512460  
34 2023-11 0.491917  
35 2023-12 0.488306  
36 2024-01 0.497688  
37 2024-02 0.508966  
38 2024-03 0.507944  
39 2024-04 0.518083  
40 2024-05 0.481775

'''  
Plot the average EV charging efficiency over time using a line plot.  
'''  
  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(12, 3))  
plt.plot(df\_charging\_efficiency['Date'].dt.to\_timestamp(), df\_charging\_efficiency['EV Charging Efficiency (%) - Normalized'], color='blue', marker='o', label='Mean Efficiency')  
plt.ylim(0, 1) # Normalized values between 0 and 1  
plt.title('Average EV Charging Efficiency')  
plt.xlabel('Date')  
plt.ylabel('EV Charging Efficiency (%)')  
plt.legend()  
plt.show()



df['Weather Conditions'].unique()

array(['Partly Cloudy', 'Sunny', 'Cloudy', 'Clear', 'Rainy'], dtype=object)

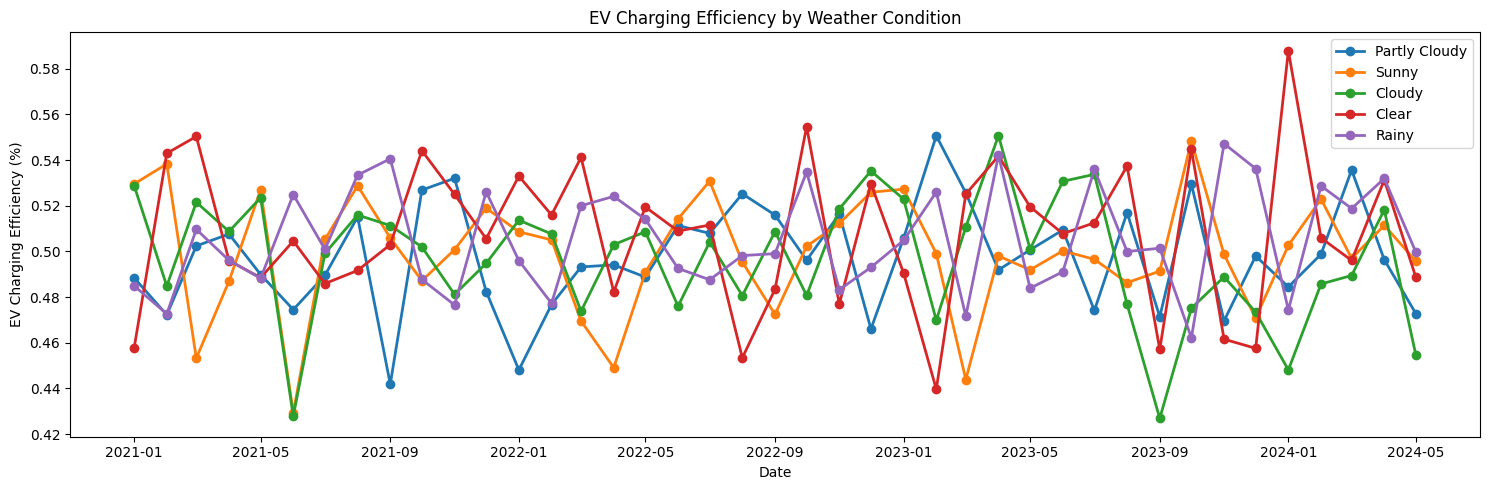
df\_weather = df.groupby([df['Date'].dt.to\_period('M'),'Weather Conditions'])['EV Charging Efficiency (%)'].mean().reset\_index()  
df\_weather

Date Weather Conditions EV Charging Efficiency (%)  
0 2021-01 Clear 89.152133  
1 2021-01 Cloudy 90.573190  
2 2021-01 Partly Cloudy 89.766215  
3 2021-01 Rainy 89.696676  
4 2021-01 Sunny 90.589128  
.. ... ... ...  
200 2024-05 Clear 89.773079  
201 2024-05 Cloudy 89.090572  
202 2024-05 Partly Cloudy 89.451986  
203 2024-05 Rainy 89.995265  
204 2024-05 Sunny 89.911689  
  
[205 rows x 3 columns]

df\_weather = df.groupby([df['Date'].dt.to\_period('M'),'Weather Conditions'])['EV Charging Efficiency (%) - Normalized'].mean().reset\_index()  
df\_weather

Date Weather Conditions EV Charging Efficiency (%) - Normalized  
0 2021-01 Clear 0.457688  
1 2021-01 Cloudy 0.528839  
2 2021-01 Partly Cloudy 0.488431  
3 2021-01 Rainy 0.484919  
4 2021-01 Sunny 0.529539  
.. ... ... ...  
200 2024-05 Clear 0.488603  
201 2024-05 Cloudy 0.454516  
202 2024-05 Partly Cloudy 0.472714  
203 2024-05 Rainy 0.499630  
204 2024-05 Sunny 0.495677  
  
[205 rows x 3 columns]

'''  
Plot the average EV charging efficiency over time for different weather conditions using a line plot.  
Each weather condition is plotted as a separate line to visualize trends and differences in efficiency.  
'''  
  
import matplotlib.pyplot as plt  
  
# List of weather conditions to plot  
weather\_conditions = ['Partly Cloudy', 'Sunny', 'Cloudy', 'Clear', 'Rainy']  
  
# Create a single plot  
plt.figure(figsize=(15, 5))  
  
# Plot each weather condition on the same axes  
for i in weather\_conditions:  
 condition\_df = df\_weather[df\_weather['Weather Conditions'] == i]  
 plt.plot(  
 condition\_df['Date'].dt.to\_timestamp(),   
 condition\_df['EV Charging Efficiency (%) - Normalized'],   
 linewidth=2,  
 marker='o',  
 label=i  
 )  
  
# Chart formatting  
plt.title('EV Charging Efficiency by Weather Condition')  
plt.xlabel('Date')  
plt.ylabel('EV Charging Efficiency (%)')  
plt.legend()  
plt.tight\_layout()  
plt.show()



'''  
This code processes a DataFrame `df` by:  
- Converting the 'Time' column from string to datetime and extracting the hour component into a new column 'Hour'.  
- Defining a function `get\_day\_period(hour)` to categorize each hour into a part of the day:  
 - Night: 21–23 or 0–3  
 - Morning: 4–11  
 - Afternoon: 12–16  
 - Evening: 17–20  
 - Unknown: if hour is outside expected range  
- Applying this function to the 'Hour' column to create a new column 'Day Period' that labels each record with its respective part of the day.  
'''  
  
  
import pandas as pd  
  
# Convert 'Time' column to datetime and extract hour  
df['Hour'] = pd.to\_datetime(df['Time'], format='%H:%M:%S').dt.hour  
  
# Function to determine part of the day  
def get\_day\_period(hour):  
 if 0 <= hour <= 3 or 21 <= hour <= 23:  
 return 'Night'  
 elif 4 <= hour <= 11:  
 return 'Morning'  
 elif 12 <= hour <= 16:  
 return 'Afternoon'  
 elif 17 <= hour <= 20:  
 return 'Evening'  
 else:  
 return 'Unknown' # just in case  
  
# Apply function to assign 'Day Period'  
df['Day Period'] = df['Hour'].apply(get\_day\_period)

df['Day Period'].unique()

array(['Night', 'Morning', 'Afternoon', 'Evening'], dtype=object)

from sklearn.preprocessing import MinMaxScaler  
  
# Normalize (Min-Max)  
scaler\_minmax = MinMaxScaler()  
df['EV Charging Demand (kW) - Normalized'] = scaler\_minmax.fit\_transform(df[['EV Charging Demand (kW)']]).round(2)

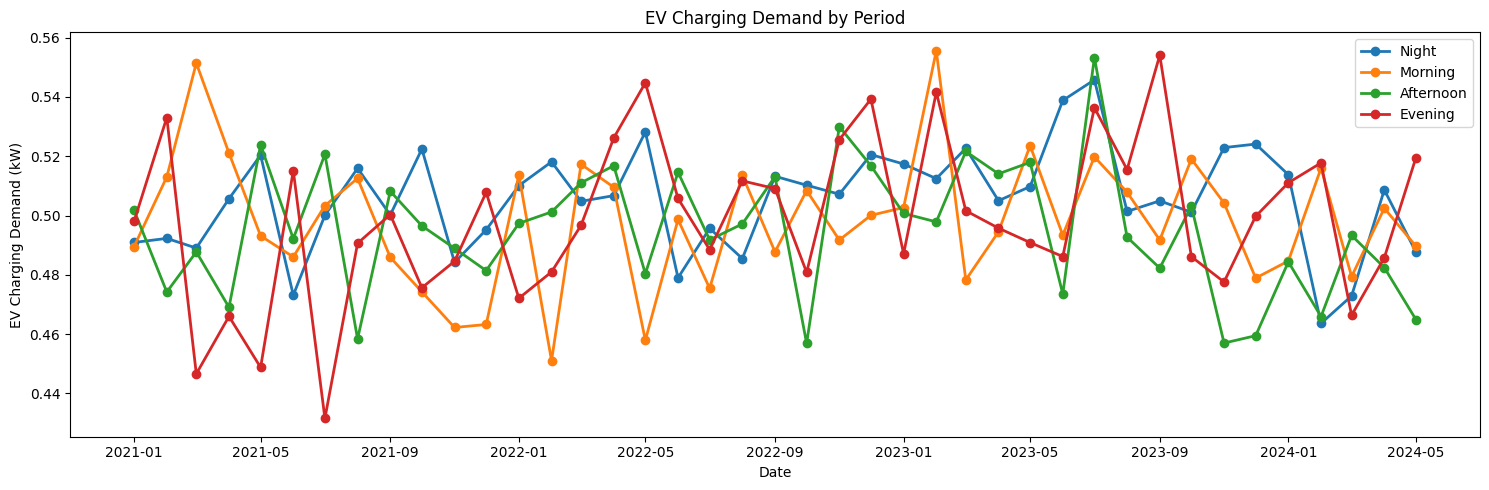
df\_period = df.groupby([df['Date'].dt.to\_period('M'),'Day Period'])['EV Charging Demand (kW)'].mean().reset\_index()  
df\_period

Date Day Period EV Charging Demand (kW)  
0 2021-01 Afternoon 0.150594  
1 2021-01 Evening 0.149428  
2 2021-01 Morning 0.146796  
3 2021-01 Night 0.147207  
4 2021-02 Afternoon 0.142247  
.. ... ... ...  
159 2024-04 Night 0.152574  
160 2024-05 Afternoon 0.139622  
161 2024-05 Evening 0.155982  
162 2024-05 Morning 0.146877  
163 2024-05 Night 0.146383  
  
[164 rows x 3 columns]

df\_period = df.groupby([df['Date'].dt.to\_period('M'),'Day Period'])['EV Charging Demand (kW) - Normalized'].mean().reset\_index()  
df\_period

Date Day Period EV Charging Demand (kW) - Normalized  
0 2021-01 Afternoon 0.501742  
1 2021-01 Evening 0.498065  
2 2021-01 Morning 0.489556  
3 2021-01 Night 0.490922  
4 2021-02 Afternoon 0.474214  
.. ... ... ...  
159 2024-04 Night 0.508667  
160 2024-05 Afternoon 0.464933  
161 2024-05 Evening 0.519500  
162 2024-05 Morning 0.489667  
163 2024-05 Night 0.487725  
  
[164 rows x 3 columns]

'''  
Plot the average EV charging demand over time for different periods of the day using a line plot.  
Each period (Night, Morning, Afternoon, Evening) is plotted as a separate line to visualize trends and differences in demand.  
'''  
  
import matplotlib.pyplot as plt  
  
# List of periods conditions to plot  
period = ['Night', 'Morning', 'Afternoon', 'Evening']  
  
# Create a single plot  
plt.figure(figsize=(15, 5))  
  
# Plot each periods condition on the same axes  
for i in period:  
 condition\_df = df\_period[df\_period['Day Period'] == i]  
 plt.plot(  
 condition\_df['Date'].dt.to\_timestamp(),   
 condition\_df['EV Charging Demand (kW) - Normalized'],   
 linewidth=2,  
 marker='o',  
 label=i  
 )  
  
# Chart formatting  
plt.title('EV Charging Demand by Period')  
plt.xlabel('Date')  
plt.ylabel('EV Charging Demand (kW)')  
plt.legend()  
plt.tight\_layout()  
plt.show()



from sklearn.preprocessing import MinMaxScaler  
  
# Normalize (Min-Max)  
scaler\_minmax = MinMaxScaler()  
df['Total Renewable Energy Production (kW) - Normalized'] = scaler\_minmax.fit\_transform(df[['Total Renewable Energy Production (kW)']]).round(2)

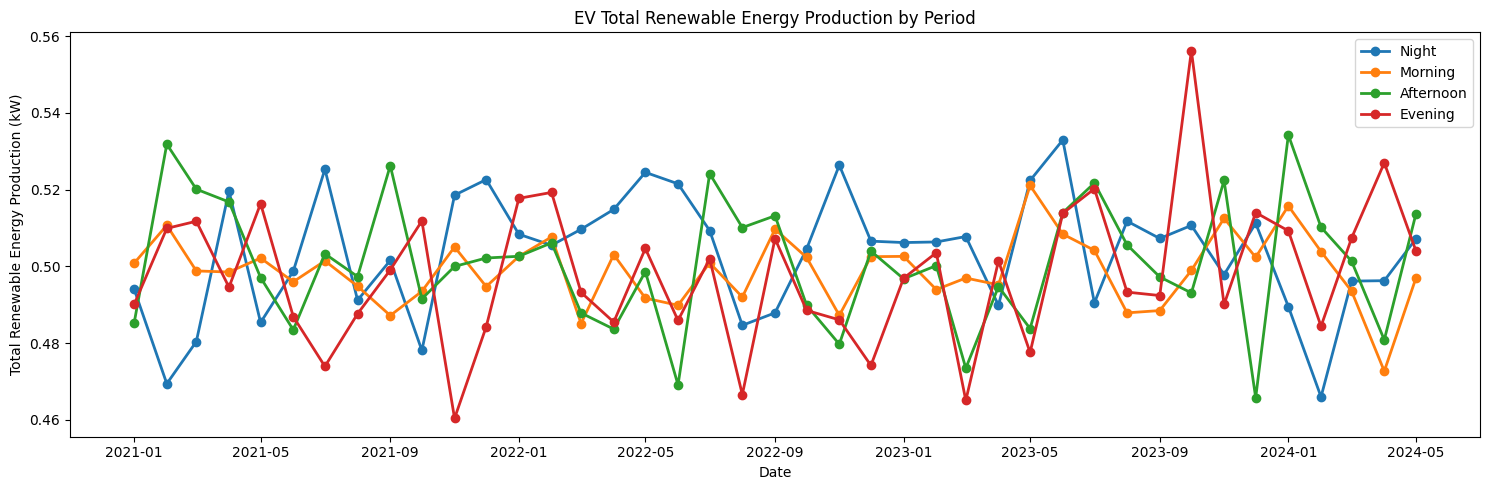
# Group by month and day period, then calculate the mean of 'Total Renewable Energy Production (kW)'  
df\_period = df.groupby([df['Date'].dt.to\_period('M'),'Day Period'])['Total Renewable Energy Production (kW)'].mean().reset\_index()  
df\_period

Date Day Period Total Renewable Energy Production (kW)  
0 2021-01 Afternoon 0.290988  
1 2021-01 Evening 0.293721  
2 2021-01 Morning 0.300473  
3 2021-01 Night 0.296199  
4 2021-02 Afternoon 0.318795  
.. ... ... ...  
159 2024-04 Night 0.297569  
160 2024-05 Afternoon 0.308193  
161 2024-05 Evening 0.302174  
162 2024-05 Morning 0.297947  
163 2024-05 Night 0.304057  
  
[164 rows x 3 columns]

# Group by month and day period, then calculate the mean of 'Total Renewable Energy Production (kW)'  
df\_period = df.groupby([df['Date'].dt.to\_period('M'),'Day Period'])['Total Renewable Energy Production (kW) - Normalized'].mean().reset\_index()  
df\_period

Date Day Period Total Renewable Energy Production (kW) - Normalized  
0 2021-01 Afternoon 0.485226   
1 2021-01 Evening 0.490081   
2 2021-01 Morning 0.500847   
3 2021-01 Night 0.494101   
4 2021-02 Afternoon 0.531857   
.. ... ... ...   
159 2024-04 Night 0.496286   
160 2024-05 Afternoon 0.513667   
161 2024-05 Evening 0.504083   
162 2024-05 Morning 0.496958   
163 2024-05 Night 0.507204   
  
[164 rows x 3 columns]

'''  
Plot the average total renewable energy production over time for different periods of the day using a line plot.  
Each period (Night, Morning, Afternoon, Evening) is plotted as a separate line to visualize trends and differences in renewable energy production.  
'''  
  
import matplotlib.pyplot as plt  
  
# List of periods conditions to plot  
period = ['Night', 'Morning', 'Afternoon', 'Evening']  
  
# Create a single plot  
plt.figure(figsize=(15, 5))  
  
# Plot each periods condition on the same axes  
for i in period:  
 condition\_df = df\_period[df\_period['Day Period'] == i]  
 plt.plot(  
 condition\_df['Date'].dt.to\_timestamp(),   
 condition\_df['Total Renewable Energy Production (kW) - Normalized'],   
 linewidth=2,  
 marker='o',  
 label=i  
 )  
  
# Chart formatting  
plt.title('EV Total Renewable Energy Production by Period')  
plt.xlabel('Date')  
plt.ylabel('Total Renewable Energy Production (kW)')  
plt.legend()  
plt.tight\_layout()  
plt.show()



# Group by month and day period, then calculate the mean of 'Total Renewable Energy Production (kW)'  
df\_period = df.groupby([df['Date'].dt.to\_period('M'),'Day Period'])['Total Renewable Energy Production (kW)'].mean().reset\_index()  
df\_period

Date Day Period Total Renewable Energy Production (kW)  
0 2021-01 Afternoon 0.290988  
1 2021-01 Evening 0.293721  
2 2021-01 Morning 0.300473  
3 2021-01 Night 0.296199  
4 2021-02 Afternoon 0.318795  
.. ... ... ...  
159 2024-04 Night 0.297569  
160 2024-05 Afternoon 0.308193  
161 2024-05 Evening 0.302174  
162 2024-05 Morning 0.297947  
163 2024-05 Night 0.304057  
  
[164 rows x 3 columns]

import pandas as pd  
  
def add\_season\_column(df, date\_col='Date'):  
 """  
 Adds a 'Season' column to the DataFrame based on the month from a date column.  
  
 Parameters:  
 df (pd.DataFrame): The DataFrame containing the date column.  
 date\_col (str): The name of the column containing datetime values.  
  
 Returns:  
 pd.DataFrame: Original DataFrame with an added 'Season' column.  
 """  
 # Ensure the date column is in datetime format  
 df[date\_col] = pd.to\_datetime(df[date\_col])  
  
 # Map months to seasons  
 df['Season'] = df[date\_col].dt.month % 12 // 3  
 season\_map = {0: 'Winter', 1: 'Spring', 2: 'Summer', 3: 'Autumn'}  
 df['Season'] = df['Season'].map(season\_map)  
  
 return df

add\_season\_column(df, date\_col='Date')

Date Time EV Charging Demand (kW) \  
0 2021-01-01 00:00:00 0.112362   
1 2021-01-01 01:00:00 0.285214   
2 2021-01-01 02:00:00 0.219598   
3 2021-01-01 03:00:00 0.179598   
4 2021-01-01 04:00:00 0.046806   
... ... ... ...   
59805 2024-05-30 20:00:00 0.173522   
59806 2024-05-30 21:00:00 0.194685   
59807 2024-05-30 22:00:00 0.232978   
59808 2024-05-30 23:00:00 0.077797   
59809 2024-05-31 00:00:00 0.248170   
  
 Solar Energy Production (kW) Wind Energy Production (kW) \  
0 0.125388 0.009105   
1 0.052697 0.107589   
2 0.105035 0.043996   
3 0.073839 0.275727   
4 0.068614 0.059824   
... ... ...   
59805 0.144697 0.109528   
59806 0.153407 0.208440   
59807 0.047612 0.138344   
59808 0.052294 0.278504   
59809 0.008804 0.185438   
  
 Electricity Price ($/kWh) Grid Availability Weather Conditions \  
0 0.137310 Available Partly Cloudy   
1 0.125105 Available Sunny   
2 0.106661 Available Cloudy   
3 0.072209 Available Partly Cloudy   
4 0.091090 Available Partly Cloudy   
... ... ... ...   
59805 0.137830 Available Sunny   
59806 0.171891 Available Clear   
59807 0.178054 Available Partly Cloudy   
59808 0.170473 Available Partly Cloudy   
59809 0.106603 Available Sunny   
  
 Battery Storage (kWh) Charging Station Capacity (kW) ... \  
0 16.532408 21.763422 ...   
1 39.106930 31.215028 ...   
2 6.112691 46.489116 ...   
3 30.041088 49.675029 ...   
4 45.085422 21.166182 ...   
... ... ... ...   
59805 30.755102 49.248154 ...   
59806 47.379324 17.980083 ...   
59807 7.738819 17.604067 ...   
59808 45.478840 8.638068 ...   
59809 15.018141 14.274271 ...   
  
 Adjusted Charging Demand (kW) Net Energy Cost ($) \  
0 0.028134 0.015428   
1 0.158721 0.035682   
2 0.175614 0.023423   
3 0.004072 0.012969   
4 0.045315 0.004264   
... ... ...   
59805 0.146345 0.023916   
59806 0.156887 0.033465   
59807 0.202289 0.041483   
59808 0.065948 0.013262   
59809 0.192285 0.026456   
  
 Carbon Footprint Reduction (kgCO2) Renewable Energy Efficiency \  
0 0.023158 0.006350   
1 0.060875 0.005799   
2 0.006425 0.003567   
3 0.083420 0.007586   
4 0.000476 0.007242   
... ... ...   
59805 0.009767 0.006238   
59806 0.012332 0.023265   
59807 0.010963 0.013152   
59808 0.003393 0.038892   
59809 0.019655 0.013623   
  
 EV Charging Efficiency (%) - Normalized Hour Day Period \  
0 0.87 0 Night   
1 0.43 1 Night   
2 0.49 2 Night   
3 0.64 3 Night   
4 0.19 4 Morning   
... ... ... ...   
59805 0.14 20 Evening   
59806 0.33 21 Night   
59807 0.02 22 Night   
59808 0.92 23 Night   
59809 0.99 0 Night   
  
 EV Charging Demand (kW) - Normalized \  
0 0.37   
1 0.95   
2 0.73   
3 0.60   
4 0.16   
... ...   
59805 0.58   
59806 0.65   
59807 0.78   
59808 0.26   
59809 0.83   
  
 Total Renewable Energy Production (kW) - Normalized Season   
0 0.22 Winter   
1 0.27 Winter   
2 0.25 Winter   
3 0.58 Winter   
4 0.21 Winter   
... ... ...   
59805 0.42 Spring   
59806 0.60 Spring   
59807 0.31 Spring   
59808 0.55 Spring   
59809 0.32 Spring   
  
[59810 rows x 30 columns]

df['Season'].unique()

array(['Winter', 'Spring', 'Summer', 'Autumn'], dtype=object)

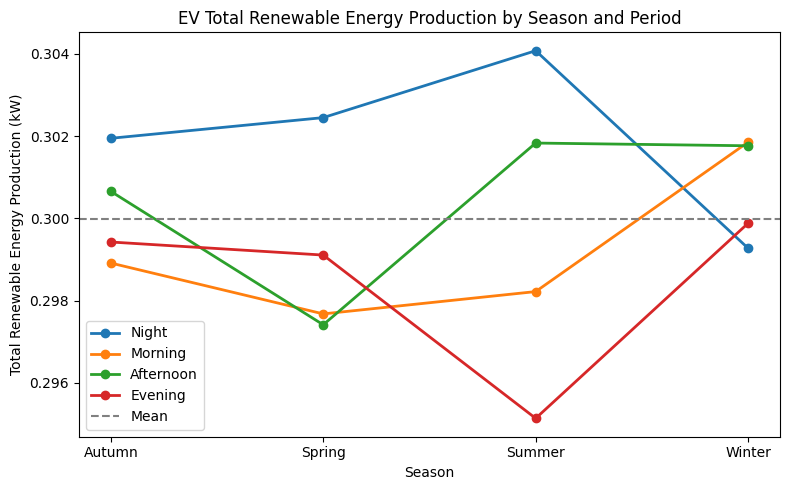
df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 59810 entries, 0 to 59809  
Data columns (total 30 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Date 59810 non-null datetime64[ns]  
 1 Time 59810 non-null object   
 2 EV Charging Demand (kW) 59810 non-null float64   
 3 Solar Energy Production (kW) 59810 non-null float64   
 4 Wind Energy Production (kW) 59810 non-null float64   
 5 Electricity Price ($/kWh) 59810 non-null float64   
 6 Grid Availability 59810 non-null object   
 7 Weather Conditions 59810 non-null object   
 8 Battery Storage (kWh) 59810 non-null float64   
 9 Charging Station Capacity (kW) 59810 non-null float64   
 10 EV Charging Efficiency (%) 59810 non-null float64   
 11 Number of EVs Charging 59810 non-null int64   
 12 Peak Demand (kW) 59810 non-null float64   
 13 Renewable Energy Usage (%) 59810 non-null float64   
 14 Grid Stability Index 59810 non-null float64   
 15 Carbon Emissions (kgCO2/kWh) 59810 non-null float64   
 16 Power Outages (hours) 59810 non-null float64   
 17 Energy Savings ($) 59810 non-null float64   
 18 Total Renewable Energy Production (kW) 59810 non-null float64   
 19 Effective Charging Capacity (kW) 59810 non-null float64   
 20 Adjusted Charging Demand (kW) 59810 non-null float64   
 21 Net Energy Cost ($) 59810 non-null float64   
 22 Carbon Footprint Reduction (kgCO2) 59810 non-null float64   
 23 Renewable Energy Efficiency 59810 non-null float64   
 24 EV Charging Efficiency (%) - Normalized 59810 non-null float64   
 25 Hour 59810 non-null int32   
 26 Day Period 59810 non-null object   
 27 EV Charging Demand (kW) - Normalized 59810 non-null float64   
 28 Total Renewable Energy Production (kW) - Normalized 59810 non-null float64   
 29 Season 59810 non-null object   
dtypes: datetime64[ns](1), float64(22), int32(1), int64(1), object(5)  
memory usage: 13.5+ MB

# Group by month and day period, then calculate the mean of 'Total Renewable Energy Production (kW)' and EV Charging Demand (kW)   
  
df\_season = df.groupby(['Season', 'Day Period']).agg({  
 'Total Renewable Energy Production (kW)': 'mean',  
 'EV Charging Demand (kW)': 'mean',  
 'Electricity Price ($/kWh)': 'mean',   
 'Number of EVs Charging': 'count' ,  
 'Carbon Emissions (kgCO2/kWh)': 'mean'   
}).reset\_index()  
  
df\_season

Season Day Period Total Renewable Energy Production (kW) \  
0 Autumn Afternoon 0.300657   
1 Autumn Evening 0.299425   
2 Autumn Morning 0.298913   
3 Autumn Night 0.301947   
4 Spring Afternoon 0.297412   
5 Spring Evening 0.299107   
6 Spring Morning 0.297676   
7 Spring Night 0.302450   
8 Summer Afternoon 0.301831   
9 Summer Evening 0.295136   
10 Summer Morning 0.298219   
11 Summer Night 0.304077   
12 Winter Afternoon 0.301765   
13 Winter Evening 0.299882   
14 Winter Morning 0.301854   
15 Winter Night 0.299272   
  
 EV Charging Demand (kW) Electricity Price ($/kWh) \  
0 0.147820 0.125899   
1 0.149755 0.124428   
2 0.147508 0.124740   
3 0.152212 0.125863   
4 0.149607 0.124320   
5 0.147207 0.127809   
6 0.150436 0.124897   
7 0.151559 0.124951   
8 0.149860 0.125134   
9 0.149327 0.122158   
10 0.150340 0.124296   
11 0.151182 0.124023   
12 0.146785 0.124295   
13 0.152309 0.126251   
14 0.149020 0.124881   
15 0.151669 0.124371   
  
 Number of EVs Charging Carbon Emissions (kgCO2/kWh)   
0 2730 0.298758   
1 2184 0.304551   
2 4368 0.297482   
3 3822 0.299313   
4 3670 0.295380   
5 2936 0.299069   
6 5872 0.299101   
7 5140 0.302939   
8 2760 0.302143   
9 2208 0.301272   
10 4416 0.299569   
11 3864 0.295912   
12 3300 0.300233   
13 2640 0.298876   
14 5280 0.301204   
15 4620 0.300411

'''  
Plot the average total renewable energy production by season and period of the day using a line plot.  
Each period (Night, Morning, Afternoon, Evening) is plotted as a separate line to visualize trends and differences in renewable energy production across seasons.  
A horizontal dashed line indicates the overall mean value for reference.  
'''  
  
import matplotlib.pyplot as plt  
  
# List of seasons to plot  
season = ['Winter', 'Spring', 'Summer', 'Autumn']  
mean\_value = df\_season['Total Renewable Energy Production (kW)'].mean()  
# Create a single plot  
plt.figure(figsize=(8, 5))  
  
# Plot each periods condition on the same axes  
for i in period:  
 condition\_df = df\_season[df\_season['Day Period'] == i]  
 plt.plot(  
 condition\_df['Season'],   
 condition\_df['Total Renewable Energy Production (kW)'],   
 linewidth=2,  
 marker='o',  
 label=i  
 )  
  
# Add a horizontal mean line  
plt.axhline(mean\_value, color='gray', linestyle='--', linewidth=1.5, label='Mean')  
  
# Chart formatting  
plt.title('EV Total Renewable Energy Production by Season and Period')  
plt.xlabel('Season')  
plt.ylabel('Total Renewable Energy Production (kW)')  
plt.legend()  
plt.tight\_layout()  
plt.show()



The image shows how **total renewable energy production for EV charging** varies by **season** (Winter, Spring, Summer, Autumn) and **period of the day** (Night, Morning, Afternoon, Evening).

### 0.0.1 Key Observations:

* **Nighttime** generally has the highest renewable energy production, especially in Summer and Spring.
* **Morning and Afternoon** periods are relatively stable, with a slight dip in Spring and a rise in Winter.
* **Evening** shows the lowest production in Summer but recovers in Winter and Autumn.
* The **dashed line** represents the overall mean renewable energy production for reference.

### 0.0.2 Background Explanation:

Grid electricity production from renewables is influenced by both **seasonal changes** (like sunlight hours, temperature, and weather) and **time of day** (solar and wind availability). For example:

* **Summer nights** may benefit from stored solar energy or favorable wind conditions.
* **Winter mornings and afternoons** may see higher production due to clearer skies or more consistent wind.
* **Evenings in Summer** may drop due to reduced solar input and higher temperatures affecting efficiency.

**Conclusion:**  
Grid renewable energy production is not constant; it fluctuates with both season and time of day, impacting the availability of clean energy for EV charging. This highlights the importance of aligning charging strategies with periods of higher renewable generation to maximize sustainability.

df['Month'] = df['Date'].dt.month

df.columns

Index(['Date', 'Time', 'EV Charging Demand (kW)',  
 'Solar Energy Production (kW)', 'Wind Energy Production (kW)',  
 'Electricity Price ($/kWh)', 'Grid Availability', 'Weather Conditions',  
 'Battery Storage (kWh)', 'Charging Station Capacity (kW)',  
 'EV Charging Efficiency (%)', 'Number of EVs Charging',  
 'Peak Demand (kW)', 'Renewable Energy Usage (%)',  
 'Grid Stability Index', 'Carbon Emissions (kgCO2/kWh)',  
 'Power Outages (hours)', 'Energy Savings ($)',  
 'Total Renewable Energy Production (kW)',  
 'Effective Charging Capacity (kW)', 'Adjusted Charging Demand (kW)',  
 'Net Energy Cost ($)', 'Carbon Footprint Reduction (kgCO2)',  
 'Renewable Energy Efficiency',  
 'EV Charging Efficiency (%) - Normalized', 'Hour', 'Day Period',  
 'EV Charging Demand (kW) - Normalized',  
 'Total Renewable Energy Production (kW) - Normalized', 'Season',  
 'Month'],  
 dtype='object')

# Group by month and day period, then calculate the mean of 'EV Charging Demand (kW)'  
df\_demand = df[['Date','Hour','Month','Time','Season','EV Charging Demand (kW)']]  
df\_demand

Date Hour Month Time Season EV Charging Demand (kW)  
0 2021-01-01 0 1 00:00:00 Winter 0.112362  
1 2021-01-01 1 1 01:00:00 Winter 0.285214  
2 2021-01-01 2 1 02:00:00 Winter 0.219598  
3 2021-01-01 3 1 03:00:00 Winter 0.179598  
4 2021-01-01 4 1 04:00:00 Winter 0.046806  
... ... ... ... ... ... ...  
59805 2024-05-30 20 5 20:00:00 Spring 0.173522  
59806 2024-05-30 21 5 21:00:00 Spring 0.194685  
59807 2024-05-30 22 5 22:00:00 Spring 0.232978  
59808 2024-05-30 23 5 23:00:00 Spring 0.077797  
59809 2024-05-31 0 5 00:00:00 Spring 0.248170  
  
[59810 rows x 6 columns]

df\_demand\_group = df\_demand.groupby(['Hour','Month','Season'])['EV Charging Demand (kW)'].mean().reset\_index()  
df\_demand\_group

Hour Month Season EV Charging Demand (kW)  
0 0 1 Winter 0.142705  
1 0 2 Winter 0.152314  
2 0 3 Spring 0.150169  
3 0 4 Spring 0.140316  
4 0 5 Spring 0.146437  
.. ... ... ... ...  
283 23 8 Summer 0.162869  
284 23 9 Autumn 0.154006  
285 23 10 Autumn 0.162504  
286 23 11 Autumn 0.146481  
287 23 12 Winter 0.161230  
  
[288 rows x 4 columns]

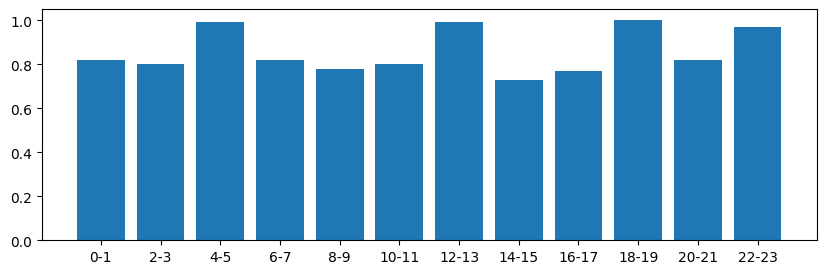
from sklearn.preprocessing import StandardScaler, MinMaxScaler  
  
# Standardize  
scaler\_std = StandardScaler()  
df\_demand\_group['EV Charging Demand (kW) - Standardized'] = scaler\_std.fit\_transform(df\_demand\_group[['EV Charging Demand (kW)']]).round(2)  
  
# Normalize (Min-Max)  
scaler\_minmax = MinMaxScaler()  
df\_demand\_group['EV Charging Demand (kW) - Normalized'] = scaler\_minmax.fit\_transform(df\_demand\_group[['EV Charging Demand (kW)']]).round(2)

# Function to map hour group number to label  
def hour\_label(hour):  
 labels = {  
 0: '0-1',  
 1: '0-1',  
 2: '2-3',  
 3: '2-3',  
 4: '4-5',  
 5: '4-5',  
 6: '6-7',  
 7: '6-7',  
 8: '8-9',  
 9: '8-9',  
 10: '10-11',  
 11: '10-11',  
 12: '12-13',  
 13: '12-13',  
 14: '14-15',  
 15: '14-15',  
 16: '16-17',  
 17: '16-17',  
 18: '18-19',  
 19: '18-19',  
 20: '20-21',  
 21: '20-21',  
 22: '22-23',  
 23: '22-23'  
 }  
 return labels.get(hour, 'Unknown')  
  
# Apply function to assign hour label  
df\_demand\_group['Hour Label'] = df\_demand\_group['Hour'].apply(hour\_label)

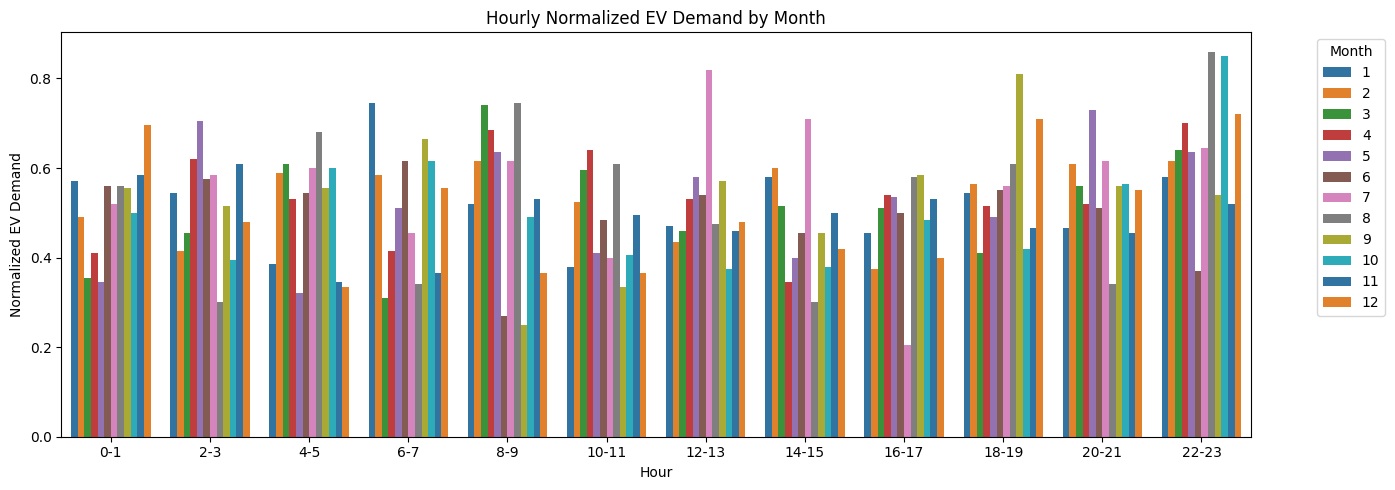
df\_demand\_group.sort\_values(by='Month')

Hour Month Season EV Charging Demand (kW) \  
24 2 1 Winter 0.153702   
264 22 1 Winter 0.152494   
36 3 1 Winter 0.148587   
48 4 1 Winter 0.144460   
60 5 1 Winter 0.142310   
.. ... ... ... ...   
83 6 12 Winter 0.158907   
35 2 12 Winter 0.146212   
47 3 12 Winter 0.149536   
263 21 12 Winter 0.146951   
23 1 12 Winter 0.152277   
  
 EV Charging Demand (kW) - Standardized \  
24 0.45   
264 0.31   
36 -0.15   
48 -0.64   
60 -0.89   
.. ...   
83 1.07   
35 -0.43   
47 -0.04   
263 -0.35   
23 0.28   
  
 EV Charging Demand (kW) - Normalized Hour Label   
24 0.60 2-3   
264 0.58 22-23   
36 0.49 2-3   
48 0.41 4-5   
60 0.36 4-5   
.. ... ...   
83 0.71 6-7   
35 0.45 2-3   
47 0.51 2-3   
263 0.46 20-21   
23 0.57 0-1   
  
[288 rows x 7 columns]

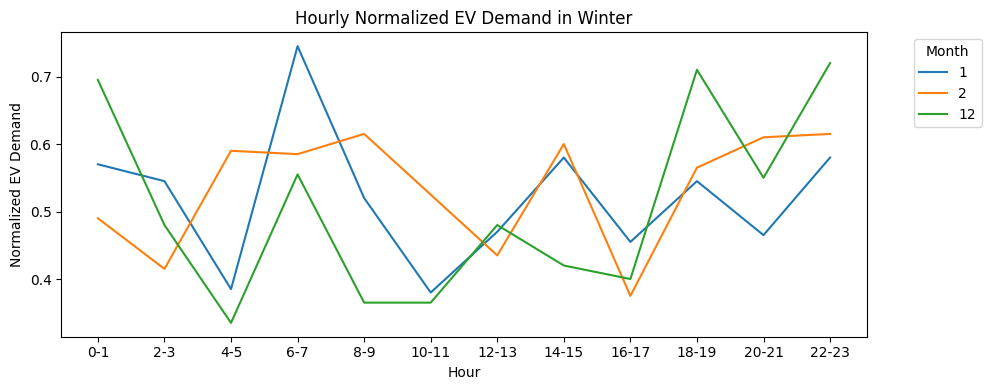
import matplotlib.pyplot as plt  
  
X = df\_demand\_group['Hour Label']  
Y = df\_demand\_group['EV Charging Demand (kW) - Normalized']  
  
plt.figure(figsize=(10, 3))  
plt.bar(X,Y)  
plt.show()

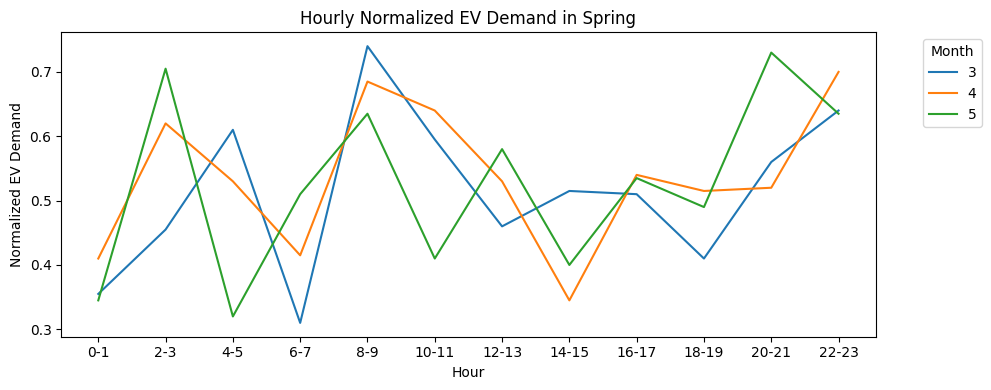


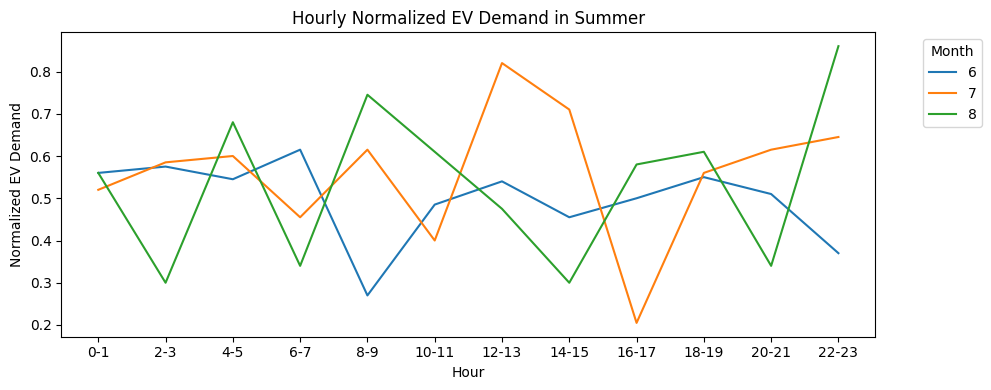
'''  
This code generates a bar plot to visualize hourly normalized EV charging demand by month:  
- Imports necessary libraries: seaborn, matplotlib, pandas, numpy.  
- Creates a figure with specified size using matplotlib.  
- Uses seaborn's `barplot` to plot:  
 - X-axis: 'Hour Label' (time of day)  
 - Y-axis: 'EV Charging Demand (kW) - Normalized'  
 - Hue: 'Month', to differentiate months with distinct colors  
 - Bars are shown side-by-side for each hour and month (dodge=True)  
 - Confidence intervals are disabled (ci=None)  
 - Uses the 'tab10' color palette for better distinction.  
- Adds axis labels, title, and legend positioned outside the plot.  
- Adjusts layout to fit all elements neatly and displays the plot.  
'''  
  
import seaborn as sns  
import matplotlib.pyplot as plt  
import pandas as pd  
import numpy as np  
  
# Plot using seaborn  
plt.figure(figsize=(14, 5))  
sns.barplot(  
 data=df\_demand\_group,  
 x='Hour Label',  
 y='EV Charging Demand (kW) - Normalized',  
 hue='Month',  
 dodge=True, # Side-by-side bars  
 palette='tab10',  
 ci=None # Disable confidence intervals  
)  
  
# Formatting  
plt.xlabel('Hour')  
plt.ylabel('Normalized EV Demand')  
plt.title('Hourly Normalized EV Demand by Month')  
plt.legend(title='Month', bbox\_to\_anchor=(1.05, 1), loc='upper left')  
plt.tight\_layout()  
plt.show()

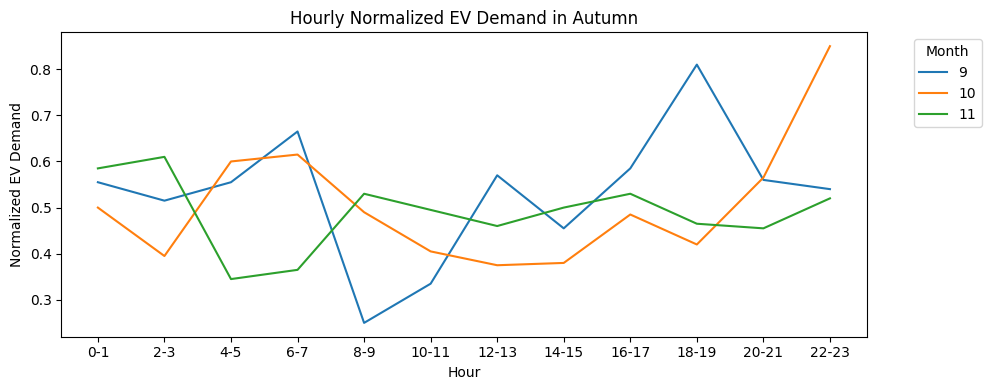


import seaborn as sns  
import matplotlib.pyplot as plt  
import pandas as pd  
import numpy as np  
  
season = ['Winter', 'Spring', 'Summer', 'Autumn']  
  
for i in season:  
 df\_condition = df\_demand\_group[df\_demand\_group['Season'] == i]  
  
 # Plot using seaborn  
 plt.figure(figsize=(10, 4))  
 sns.lineplot(  
 data=df\_condition,  
 x='Hour Label',  
 y='EV Charging Demand (kW) - Normalized',  
 hue='Month',  
 palette='tab10',  
 ci=None # Disable confidence intervals  
 )  
  
 # Formatting  
 plt.xlabel('Hour')  
 plt.ylabel('Normalized EV Demand')  
 plt.title(f'Hourly Normalized EV Demand in {i}')  
 plt.legend(title='Month', bbox\_to\_anchor=(1.05, 1), loc='upper left')  
 plt.tight\_layout()  
 plt.show()









## 0.1 Hourly Normalized EV Charging Demand by Season

This visualization displays the hourly normalized EV charging demand patterns across different seasons, with each season shown in a separate subplot.

### 0.1.1 Visualization Details:

* **Plot Type**: 2x2 grid of line plots (one for each season)
* **Figure Size**: 14x6 inches
* **Shared Axes**: All subplots share the same x and y axis scales
* **Data Representation**:
  + X-axis: Hour of day (0-23)
  + Y-axis: Normalized EV charging demand (kW)
  + Lines colored by month (using 'tab10' palette)
* **Confidence Intervals**: Disabled for cleaner visualization

### 0.1.2 Customizations:

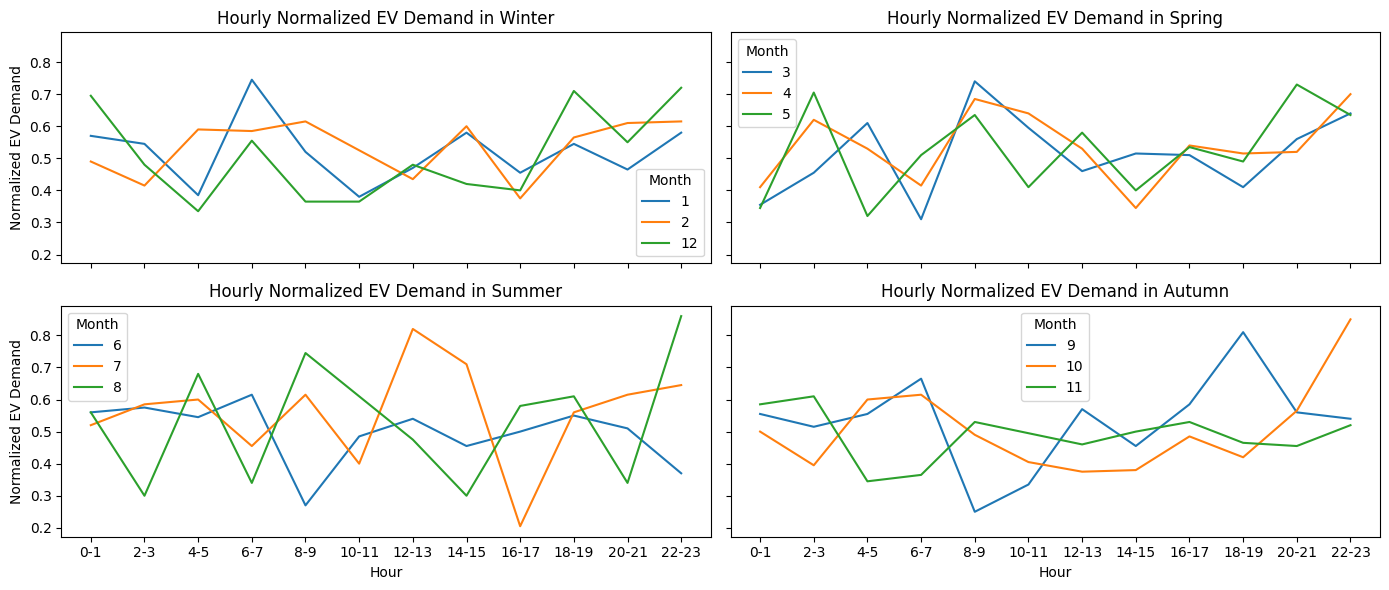
* Individual titles for each subplot indicating the season
* Consistent axis labels:
  + X-label: "Hour"
  + Y-label: "Normalized EV Demand"
* Legend from the first subplot used for all plots
* tight\_layout() applied to prevent label overlapping

### 0.1.3 Purpose:

This visualization helps compare how EV charging demand patterns vary:

1. Across different hours of the day
2. Between different months
3. Among the four seasons (Winter, Spring, Summer, Autumn)

import seaborn as sns  
import matplotlib.pyplot as plt  
  
fig, axes = plt.subplots(2, 2, figsize=(14, 6), sharex=True, sharey=True)  
season = ['Winter', 'Spring', 'Summer', 'Autumn']  
  
for idx, s in enumerate(season):  
 ax = axes[idx // 2, idx % 2]  
 df\_condition = df\_demand\_group[df\_demand\_group['Season'] == s]  
 sns.lineplot(  
 data=df\_condition,  
 x='Hour Label',  
 y='EV Charging Demand (kW) - Normalized',  
 hue='Month',  
 palette='tab10',  
 ax=ax,  
 ci=None # Disable confidence intervals  
 )  
 ax.set\_title(f'Hourly Normalized EV Demand in {s}')  
 ax.set\_xlabel('Hour')  
 ax.set\_ylabel('Normalized EV Demand')  
  
# Add a single legend outside the plot  
handles, labels = axes[0, 0].get\_legend\_handles\_labels()  
plt.tight\_layout()  
plt.show()



df\_demand\_group.head()

Hour Month Season EV Charging Demand (kW) \  
0 0 1 Winter 0.142705   
1 0 2 Winter 0.152314   
2 0 3 Spring 0.150169   
3 0 4 Spring 0.140316   
4 0 5 Spring 0.146437   
  
 EV Charging Demand (kW) - Standardized \  
0 -0.85   
1 0.29   
2 0.03   
3 -1.13   
4 -0.41   
  
 EV Charging Demand (kW) - Normalized Hour Label   
0 0.37 0-1   
1 0.57 0-1   
2 0.53 0-1   
3 0.32 0-1   
4 0.45 0-1

## 0.2 Hourly Normalized EV Charging Demand (Aggregated)

This visualization displays the overall trend of normalized EV charging demand across different hours of the day, aggregating all available data.

### 0.2.1 Visualization Details:

* **Plot Type**: Single line plot
* **Figure Size**: 10×5 inches
* **Data Representation**:
  + **X-axis**: Hour of day (Hour Label, likely 0–23)
  + **Y-axis**: Normalized EV charging demand (EV Charging Demand (kW) - Normalized)
* **Confidence Intervals**: Disabled (ci=None) for a clean trend line

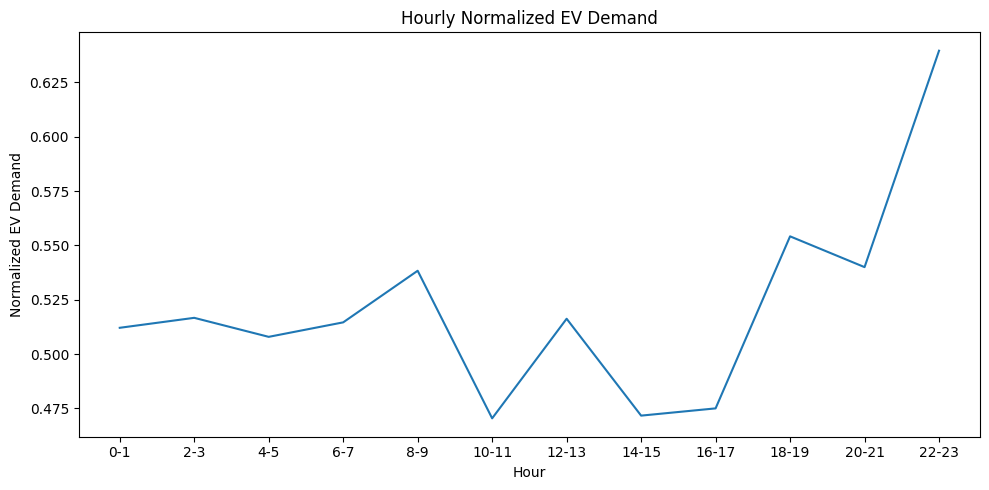
### 0.2.2 Customizations:

* **Axis Labels**:
  + X-label: Hour
  + Y-label: Normalized EV Demand
* **Title**: Hourly Normalized EV Demand (describes the plot's focus)
* **Layout**: tight\_layout() ensures no overlapping elements

### 0.2.3 Purpose:

This plot provides a high-level overview of how EV charging demand fluctuates throughout the day, ignoring seasonal or monthly variations. Useful for identifying peak demand hours and general usage patterns.

import seaborn as sns  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(10, 5))  
sns.lineplot(  
 data=df\_demand\_group,  
 x='Hour Label',  
 y='EV Charging Demand (kW) - Normalized',  
 ci=None  
 )  
plt.xlabel('Hour')  
plt.ylabel('Normalized EV Demand')  
plt.title('Hourly Normalized EV Demand')  
plt.tight\_layout()  
plt.show()



df.columns

Index(['Date', 'Time', 'EV Charging Demand (kW)',  
 'Solar Energy Production (kW)', 'Wind Energy Production (kW)',  
 'Electricity Price ($/kWh)', 'Grid Availability', 'Weather Conditions',  
 'Battery Storage (kWh)', 'Charging Station Capacity (kW)',  
 'EV Charging Efficiency (%)', 'Number of EVs Charging',  
 'Peak Demand (kW)', 'Renewable Energy Usage (%)',  
 'Grid Stability Index', 'Carbon Emissions (kgCO2/kWh)',  
 'Power Outages (hours)', 'Energy Savings ($)',  
 'Total Renewable Energy Production (kW)',  
 'Effective Charging Capacity (kW)', 'Adjusted Charging Demand (kW)',  
 'Net Energy Cost ($)', 'Carbon Footprint Reduction (kgCO2)',  
 'Renewable Energy Efficiency',  
 'EV Charging Efficiency (%) - Normalized', 'Hour', 'Day Period',  
 'EV Charging Demand (kW) - Normalized',  
 'Total Renewable Energy Production (kW) - Normalized', 'Season',  
 'Month'],  
 dtype='object')

# 1 SHAP Analysis for EV Charging Demand Prediction

This code performs a machine learning pipeline to analyze feature importance for EV charging demand prediction using SHAP (SHapley Additive exPlanations).

## 1.1 Data Preprocessing

### 1.1.1 Features & Target

* **Numerical Features (21)**: ```python ['Solar Energy Production (kW)', 'Wind Energy Production (kW)', ... , 'Hour', 'Month']

# 2 EV Charging Demand Prediction with SHAP Analysis

## 2.1 Data Preparation and Feature Engineering

This section defines the feature sets and target variable for our EV charging demand prediction model. We establish three categories of variables:

* **Numerical features**: 21 continuous variables including energy production metrics, pricing, battery specifications, charging infrastructure details, and environmental factors
* **Categorical features**: 3 categorical variables (Season, Day Period, Weather Conditions) that will be one-hot encoded
* **Target variable**: EV Charging Demand (kW) - our prediction objective

The feature selection encompasses both direct charging-related variables and contextual factors that influence charging patterns.

## 2.2 Data Preprocessing Pipeline

The preprocessing pipeline prepares the dataset for machine learning by:

1. **Categorical Encoding**: Converting categorical variables to numerical format using one-hot encoding with drop\_first=True to avoid multicollinearity
2. **Missing Value Handling**: Removing rows with missing values to ensure data quality
3. **Feature Matrix Construction**: Building the feature matrix X by combining numerical features with encoded categorical variables
4. **Target Variable Extraction**: Isolating the EV Charging Demand as our target variable y

This approach ensures all features are in a suitable format for linear regression modeling.

## 2.3 Feature Standardization and Data Splitting

To optimize model performance and SHAP interpretability:

1. **Standardization**: Apply StandardScaler to normalize all features, ensuring they have zero mean and unit variance
2. **Train-Test Split**: Divide the dataset into 80% training and 20% testing sets with a fixed random state for reproducibility

Standardization is particularly important for linear models as it prevents features with larger scales from dominating the model coefficients.

## 2.4 Model Training and SHAP Explainer Setup

We train a Linear Regression model on the standardized training data and prepare it for interpretability analysis:

1. **Model Training**: Fit a LinearRegression model to learn the relationship between features and EV charging demand
2. **SHAP Explainer**: Initialize a SHAP explainer specifically designed for linear models using the training data as background
3. **SHAP Values Calculation**: Compute SHAP values for the test set to understand feature contributions to individual predictions

SHAP (SHapley Additive exPlanations) provides unified framework for interpreting machine learning models.

## 2.5 SHAP Visualization Analysis

Generate comprehensive visualizations to understand model behavior:

1. **Bar Plot**: shap.plots.bar() shows the average absolute impact of each feature across all test samples
2. **Beeswarm Plot**: shap.plots.beeswarm() displays the distribution of SHAP values for each feature, revealing both magnitude and direction of influence
3. **Waterfall Plot**: shap.plots.waterfall() demonstrates how individual features contribute to a single prediction, starting from the expected value

These visualizations provide different perspectives on feature importance and model decision-making processes.

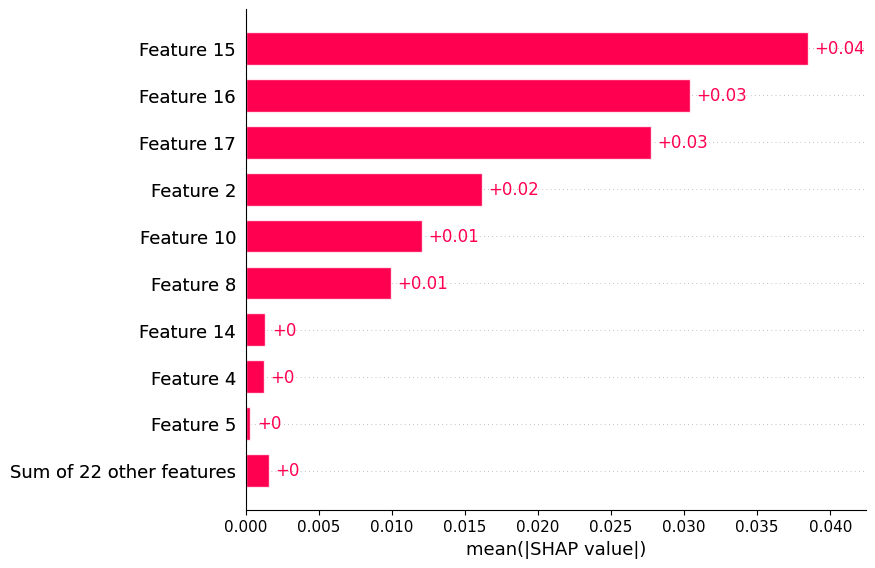
## 2.6 SHAP Values Data Frame Creation

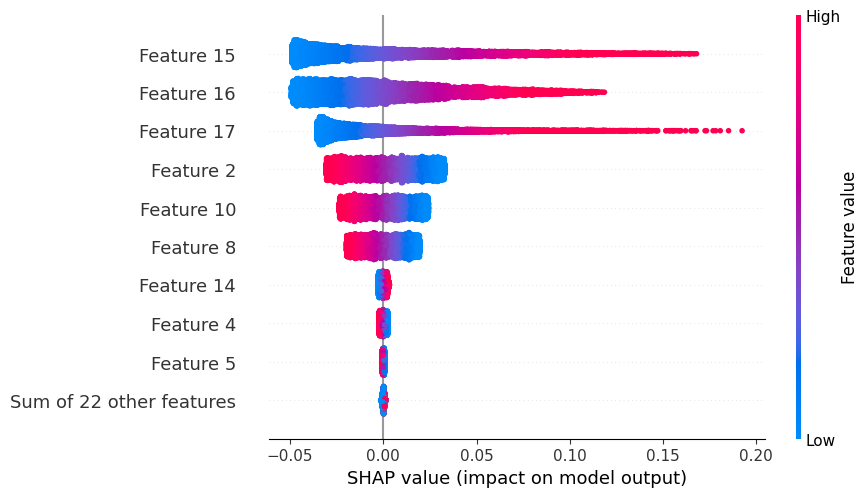
Convert SHAP values into a pandas DataFrame for further analysis:

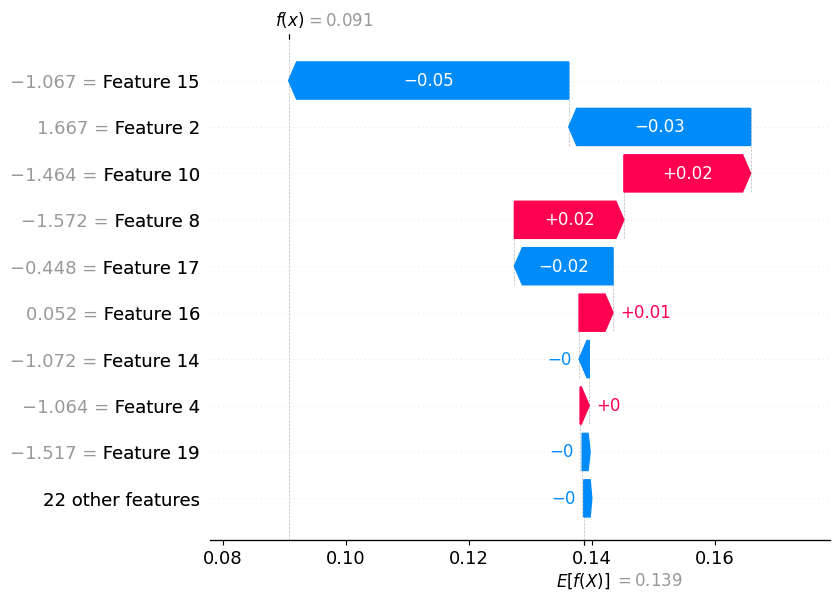
* **Structure**: Each row represents a test sample, each column represents a feature
* **Values**: SHAP values indicating how much each feature contributes to the prediction for that specific sample
* **Purpose**: Enables quantitative analysis of feature contributions and supports downstream analytical tasks

The DataFrame format facilitates statistical analysis and custom visualizations of the SHAP values.

import pandas as pd  
import numpy as np  
import shap  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
  
features = [  
 'Solar Energy Production (kW)',   
 'Wind Energy Production (kW)',   
 'Electricity Price ($/kWh)',  
 'Battery Storage (kWh)',  
 'Charging Station Capacity (kW)',  
 'EV Charging Efficiency (%)',  
 'Number of EVs Charging',  
 'Peak Demand (kW)',  
 'Renewable Energy Usage (%)',  
 'Grid Stability Index',   
 'Carbon Emissions (kgCO2/kWh)',  
 'Power Outages (hours)',   
 'Energy Savings ($)',  
 'Total Renewable Energy Production (kW)',  
 'Effective Charging Capacity (kW)',   
 'Adjusted Charging Demand (kW)',  
 'Net Energy Cost ($)',   
 'Carbon Footprint Reduction (kgCO2)',  
 'Renewable Energy Efficiency',  
 'Hour',   
 'Month'  
]  
  
categorical = ['Season', 'Day Period', 'Weather Conditions']  
target = 'EV Charging Demand (kW)'  
  
# --- Step 2: Preprocessing ---  
df\_model = df.copy()  
  
# One-hot encode categoricals  
df\_model = pd.get\_dummies(df\_model, columns=categorical, drop\_first=True)  
  
# Fill missing values if any  
df\_model = df\_model.dropna()  
  
X = df\_model[features + [col for col in df\_model.columns if col.startswith(('Season\_', 'Day Period\_', 'Weather Conditions\_'))]]  
y = df\_model[target]  
  
# Standardize features (optional but helps SHAP with linear models)  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
# Train-test split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
explainer = shap.Explainer(model, X\_train) # uses LinearExplainer  
shap\_values = explainer(X\_test)  
  
shap.plots.bar(shap\_values)  
  
shap.plots.beeswarm(shap\_values)  
  
shap.plots.waterfall(shap\_values[0])  
  
shap\_df = pd.DataFrame(shap\_values.values, columns=X.columns)  
shap\_df.head()







Solar Energy Production (kW) Wind Energy Production (kW) \  
0 -0.000075 -0.000034   
1 0.000075 -0.000058   
2 -0.000032 -0.000061   
3 -0.000026 -0.000007   
4 0.000023 0.000007   
  
 Electricity Price ($/kWh) Battery Storage (kWh) \  
0 -0.029581 -0.000123   
1 0.001265 0.000111   
2 0.024145 -0.000081   
3 -0.021962 0.000208   
4 -0.026896 -0.000199   
  
 Charging Station Capacity (kW) EV Charging Efficiency (%) \  
0 0.001493 0.000112   
1 -0.002139 0.000035   
2 -0.001049 -0.000596   
3 0.002208 0.000267   
4 -0.001304 0.000158   
  
 Number of EVs Charging Peak Demand (kW) Renewable Energy Usage (%) \  
0 0.000023 -1.181501e-06 0.017817   
1 -0.000068 1.011565e-06 0.012076   
2 -0.000022 -1.410294e-07 -0.003270   
3 0.000023 4.172432e-07 -0.010296   
4 -0.000090 -2.842437e-07 -0.011039   
  
 Grid Stability Index ... Season\_Spring Season\_Summer Season\_Winter \  
0 -0.000004 ... -0.00018 -0.000071 -0.000184   
1 0.000022 ... 0.00042 -0.000071 -0.000184   
2 0.000034 ... 0.00042 -0.000071 -0.000184   
3 -0.000024 ... 0.00042 -0.000071 -0.000184   
4 -0.000035 ... 0.00042 -0.000071 -0.000184   
  
 Day Period\_Evening Day Period\_Morning Day Period\_Night \  
0 -0.000004 1.936035e-07 0.000131   
1 0.000025 1.936035e-07 -0.000046   
2 0.000025 1.936035e-07 -0.000046   
3 -0.000004 -2.786002e-07 -0.000046   
4 0.000025 1.936035e-07 -0.000046   
  
 Weather Conditions\_Cloudy Weather Conditions\_Partly Cloudy \  
0 -0.000017 0.000016   
1 -0.000017 0.000016   
2 -0.000017 0.000016   
3 -0.000017 0.000016   
4 -0.000017 0.000016   
  
 Weather Conditions\_Rainy Weather Conditions\_Sunny   
0 -0.000007 0.000072   
1 -0.000007 -0.000305   
2 0.000038 0.000072   
3 -0.000007 -0.000305   
4 -0.000007 -0.000305   
  
[5 rows x 31 columns]

# 3 SHAP Force Plot Visualization

## 3.1 Interactive SHAP Force Plot Setup

This section creates an interactive force plot visualization to explain individual predictions:

1. **JavaScript Initialization**: shap.initjs() loads the necessary JavaScript libraries for rendering interactive SHAP visualizations in Jupyter notebooks
2. **Force Plot Generation**: Creates a detailed force plot that shows how each feature pushes the prediction above or below the expected baseline value

## 3.2 Understanding the Force Plot

The force plot provides an intuitive visualization of a single prediction by displaying:

* **Baseline (Expected Value)**: The average prediction across all training samples
* **Feature Contributions**: How each feature value pushes the prediction higher (red) or lower (blue) than the baseline
* **Final Prediction**: The sum of the baseline plus all feature contributions
* **Feature Values**: The actual values of features for this specific sample

This visualization helps answer the question: "Why did the model predict this specific value for this particular sample?"

shap.initjs()  
shap.force\_plot(explainer.expected\_value, shap\_values[0].values, X\_test[0], feature\_names=X.columns)

<IPython.core.display.HTML object>

<shap.plots.\_force.AdditiveForceVisualizer at 0x274d6666110>

Based on the SHAP force plot outcome, following actionable suggestions to the end user for efficient EV charging demand can be provided:

1. **Increase Renewable Energy Usage:**  
   The plot shows that higher renewable energy usage (%) significantly reduces the predicted charging demand. Encourage maximizing the integration of renewables (like solar or wind) into the charging infrastructure to lower demand on the grid and improve sustainability.
2. **Monitor Carbon Emissions:**  
   Higher carbon emissions (kgCO2/kWh) are associated with increased charging demand. Reducing reliance on high-emission energy sources can help manage demand and support environmental goals.
3. **Optimize Adjusted Charging Demand:**  
   The adjusted charging demand (kW) also plays a key role. Implementing smart charging strategies (like load shifting or demand response) can help flatten peaks and distribute demand more evenly.
4. **Leverage Electricity Price Signals:**  
   Higher electricity prices are linked to lower charging demand. Consider incentivizing charging during off-peak hours when prices are lower, or using dynamic pricing to influence user behavior.
5. **Focus on Carbon Footprint Reduction:**  
   Efforts to reduce the carbon footprint (kgCO2) are associated with lower demand. Invest in energy efficiency measures and cleaner technologies to further decrease overall demand.

**Summary:**  
To efficiently plan for EV charging demand, prioritize increasing renewable energy usage, reducing carbon emissions, optimizing charging schedules, and leveraging electricity price signals. These actions will help balance demand, reduce costs, and support sustainability objectives.

base\_value = explainer.expected\_value  
shap\_sum = shap\_values[0].values.sum()  
actual\_pred = base\_value + shap\_sum  
  
print(f"Base Value: {base\_value}")  
print(f"SHAP Sum: {shap\_sum}")  
print(f"Actual Prediction: {actual\_pred}")

Base Value: 0.13871081272012317  
SHAP Sum: -0.04802000718084904  
Actual Prediction: 0.09069080553927414

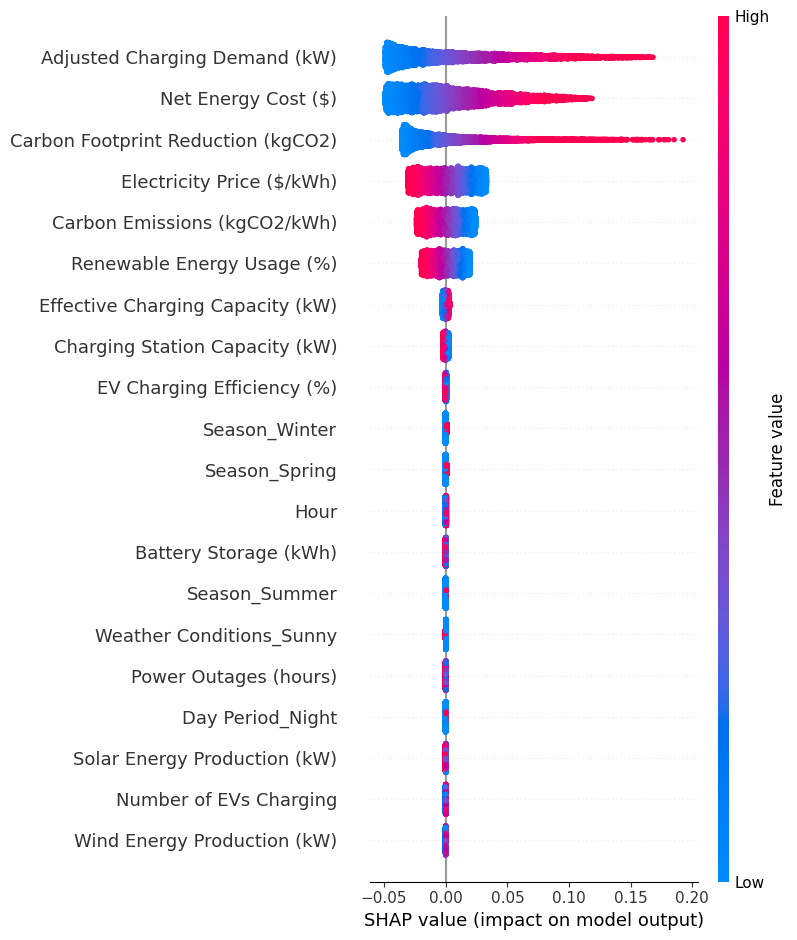
# 4 SHAP Summary Plot Analysis

## 4.1 Interpreting the Summary Plot

This visualization reveals key insights about model behavior:

1. **Feature Importance**: Features at the top have the greatest overall impact on predictions
2. **Impact Patterns**: The spread of dots shows whether a feature has consistent or variable effects
3. **Value-Impact Relationships**: Color patterns reveal how feature values correlate with their impact:
   * Red dots on the right: High feature values increase predictions
   * Blue dots on the left: Low feature values decrease predictions
   * Mixed patterns: Complex non-linear relationships

shap.summary\_plot(shap\_values, X\_test, feature\_names=X.columns)



X.columns

Index(['Solar Energy Production (kW)', 'Wind Energy Production (kW)',  
 'Electricity Price ($/kWh)', 'Battery Storage (kWh)',  
 'Charging Station Capacity (kW)', 'EV Charging Efficiency (%)',  
 'Number of EVs Charging', 'Peak Demand (kW)',  
 'Renewable Energy Usage (%)', 'Grid Stability Index',  
 'Carbon Emissions (kgCO2/kWh)', 'Power Outages (hours)',  
 'Energy Savings ($)', 'Total Renewable Energy Production (kW)',  
 'Effective Charging Capacity (kW)', 'Adjusted Charging Demand (kW)',  
 'Net Energy Cost ($)', 'Carbon Footprint Reduction (kgCO2)',  
 'Renewable Energy Efficiency', 'Hour', 'Month', 'Season\_Spring',  
 'Season\_Summer', 'Season\_Winter', 'Day Period\_Evening',  
 'Day Period\_Morning', 'Day Period\_Night', 'Weather Conditions\_Cloudy',  
 'Weather Conditions\_Partly Cloudy', 'Weather Conditions\_Rainy',  
 'Weather Conditions\_Sunny'],  
 dtype='object')

# 5 Neural Network Model for EV Charging Demand Prediction

## 5.1 Library Imports and Dependencies

This section imports the essential libraries for building and evaluating a neural network model:

* **Data Handling**: pandas and numpy for data manipulation and numerical operations
* **Model Building**: MLPRegressor from scikit-learn for multi-layer perceptron regression
* **Evaluation Metrics**: MAE, MSE, and R² score for comprehensive model assessment
* **Visualization**: matplotlib.pyplot for plotting training curves and prediction comparisons
* **Preprocessing**: StandardScaler for feature normalization

## 5.2 Feature Selection and Target Definition

The model uses a focused set of economic and environmental features:

* **Selected Features**: 5 key variables including energy costs, carbon metrics, electricity pricing, and renewable energy usage
* **Categorical Variables**: Season, Day Period, and Weather Conditions (currently commented out)
* **Target Variable**: EV Charging Demand (kW) as the prediction objective

This streamlined feature set focuses on the most impactful economic and environmental factors affecting charging demand.

## 5.3 Data Preprocessing Pipeline

The preprocessing steps prepare the data for neural network training:

1. **Data Copying**: Create a working copy of the original dataset to preserve data integrity
2. **Feature Matrix Creation**: Extract only the selected numerical features (categorical encoding currently disabled)
3. **Target Extraction**: Isolate the EV Charging Demand as the dependent variable
4. **Feature Standardization**: Apply StandardScaler to normalize all features to zero mean and unit variance

Standardization is crucial for neural networks as it ensures all features contribute equally to the learning process.

## 5.4 Train-Test Split Configuration

The data is split using scikit-learn's best practices:

* **Training Set**: 80% of the data for model learning
* **Test Set**: 20% of the data for unbiased evaluation
* **Random State**: Fixed seed (42) ensures reproducible results across runs

This split ratio provides sufficient training data while maintaining adequate test samples for reliable evaluation.

## 5.5 Neural Network Architecture and Training

The model implements a Multi-Layer Perceptron (MLP) with specific architectural choices:

* **Hidden Layers**: Two hidden layers with 64 and 32 neurons respectively
* **Architecture Pattern**: Decreasing layer sizes (64 → 32) for feature compression
* **Training Parameters**: Maximum 1000 iterations with verbose output for monitoring
* **Random State**: Fixed seed for reproducible model initialization

The two-layer architecture balances model complexity with training efficiency, suitable for the selected feature set.

## 5.6 Model Evaluation and Metrics

Comprehensive evaluation using multiple regression metrics:

* **R² Score**: Coefficient of determination measuring explained variance
* **MAE**: Mean Absolute Error for average prediction deviation
* **MSE**: Mean Squared Error penalizing larger errors more heavily
* **RMSE**: Root Mean Squared Error in original units for interpretability

These metrics provide different perspectives on model performance and prediction accuracy.

## 5.7 Training Progress Visualization

The loss curve visualization monitors training effectiveness:

* **Loss Curve Plot**: Shows how the model's training loss decreases over epochs
* **Convergence Assessment**: Helps identify if the model has converged or needs more iterations
* **Training Diagnostics**: Reveals potential overfitting or underfitting issues

This visualization is essential for understanding the training dynamics and optimization process.

## 5.8 Model Performance Summary

The comprehensive performance report includes:

* **Accuracy Metrics**: All four evaluation metrics with 4-decimal precision
* **Model Configuration**: Number of features, hidden layers, and training epochs
* **Feature List**: Complete list of input variables used in the model
* **Architecture Summary**: Details of the neural network structure

This summary provides stakeholders with complete transparency about model performance and configuration.

## 5.9 Prediction Comparison Visualization

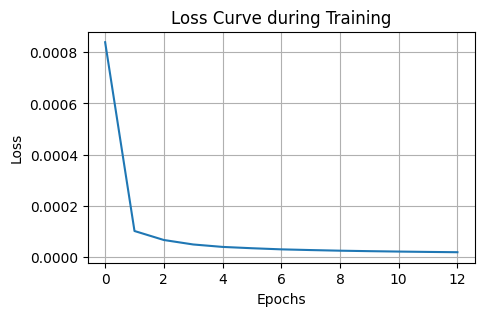
The actual vs predicted plot demonstrates model performance:

* **Sample Visualization**: Shows first 100 test samples for detailed comparison
* **Dual Line Plot**: Actual values (circles) vs predicted values (x-marks)
* **Performance Assessment**: Visual evaluation of prediction accuracy and patterns
* **Error Analysis**: Identifies systematic biases or prediction challenges

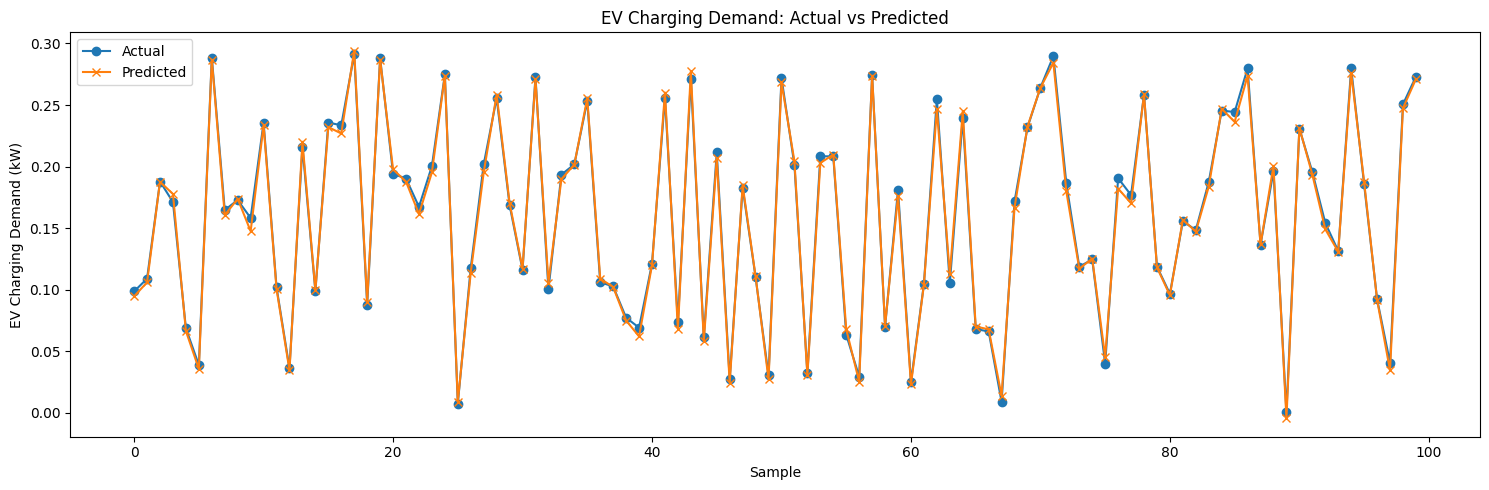
This visualization helps stakeholders understand how well the model captures real-world charging demand patterns.

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neural\_network import MLPRegressor  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score  
import matplotlib.pyplot as plt  
from sklearn.preprocessing import StandardScaler  
  
# --- Step 1: Select Features & Target ---  
features = [  
#'Adjusted Charging Demand (kW)',  
'Net Energy Cost ($)',  
'Carbon Footprint Reduction (kgCO2)',  
'Electricity Price ($/kWh)',  
'Carbon Emissions (kgCO2/kWh)',  
'Renewable Energy Usage (%)'  
]  
  
categorical = ['Season', 'Day Period', 'Weather Conditions']  
target = 'EV Charging Demand (kW)'  
  
# --- Step 2: Preprocessing ---  
df\_model = df.copy()  
  
#One-hot encode categoricals  
#df\_model = pd.get\_dummies(df\_model, columns=categorical, drop\_first=True)  
  
# Fill missing values if any  
#df\_model = df\_model.dropna()  
  
#X = df\_model[features + [col for col in df\_model.columns if col.startswith(('Season\_', 'Day Period\_', 'Weather Conditions\_'))]]  
X = df\_model[features]  
y = df\_model[target]  
  
# Normalize features  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
# --- Step 3: Train/Test Split ---  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
  
# --- Step 4: Build & Train Neural Network ---  
model = MLPRegressor(hidden\_layer\_sizes=(64, 32), max\_iter=1000, random\_state=42,verbose=True)  
model.fit(X\_train, y\_train)  
  
# --- Step 5: Predictions & Evaluation ---  
y\_pred = model.predict(X\_test)  
  
r2 = r2\_score(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse)  
  
if hasattr(model, 'loss\_curve\_'):  
 plt.figure(figsize=(5, 3))  
 plt.plot(model.loss\_curve\_)  
 plt.title('Loss Curve during Training')  
 plt.xlabel('Epochs')  
 plt.ylabel('Loss')  
 plt.grid(True)  
 plt.show()  
  
  
print(f"R² Score: {r2:.4f}")  
print(f"MAE: {mae:.4f}")  
print(f"MSE: {mse:.4f}")  
print(f"RMSE: {rmse:.4f}")  
print(f"Number of Features: {X.shape[1]}")  
print(f"features: {features}")  
print(f"Target: {target}")  
print(f"Number of Hidden Layers: {len(model.hidden\_layer\_sizes)}")  
print(f"Number of Epochs: {model.n\_iter\_}")  
  
  
# --- Step 6: Plot Actual vs Predicted ---  
plt.figure(figsize=(15, 5))  
plt.plot(y\_test.values[:100], label='Actual', marker='o')  
plt.plot(y\_pred[:100], label='Predicted', marker='x')  
plt.title("EV Charging Demand: Actual vs Predicted")  
plt.xlabel("Sample")  
plt.ylabel("EV Charging Demand (kW)")  
plt.legend()  
plt.tight\_layout()  
plt.show()

Iteration 1, loss = 0.00083910  
Iteration 2, loss = 0.00010142  
Iteration 3, loss = 0.00006625  
Iteration 4, loss = 0.00004879  
Iteration 5, loss = 0.00003926  
Iteration 6, loss = 0.00003410  
Iteration 7, loss = 0.00002972  
Iteration 8, loss = 0.00002696  
Iteration 9, loss = 0.00002450  
Iteration 10, loss = 0.00002265  
Iteration 11, loss = 0.00002106  
Iteration 12, loss = 0.00001968  
Iteration 13, loss = 0.00001851  
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.



R² Score: 0.9978  
MAE: 0.0032  
MSE: 0.0000  
RMSE: 0.0040  
Number of Features: 5  
features: ['Net Energy Cost ($)', 'Carbon Footprint Reduction (kgCO2)', 'Electricity Price ($/kWh)', 'Carbon Emissions (kgCO2/kWh)', 'Renewable Energy Usage (%)']  
Target: EV Charging Demand (kW)  
Number of Hidden Layers: 2  
Number of Epochs: 13



print("Length of y\_train",len(y\_train))  
print("Length of y\_test",len(y\_test))  
print("Length of y\_pred",len(y\_pred))

Length of y\_train 47848  
Length of y\_test 11962  
Length of y\_pred 11962

df.head()

Date Time EV Charging Demand (kW) Solar Energy Production (kW) \  
0 2021-01-01 00:00:00 0.112362 0.125388   
1 2021-01-01 01:00:00 0.285214 0.052697   
2 2021-01-01 02:00:00 0.219598 0.105035   
3 2021-01-01 03:00:00 0.179598 0.073839   
4 2021-01-01 04:00:00 0.046806 0.068614   
  
 Wind Energy Production (kW) Electricity Price ($/kWh) Grid Availability \  
0 0.009105 0.137310 Available   
1 0.107589 0.125105 Available   
2 0.043996 0.106661 Available   
3 0.275727 0.072209 Available   
4 0.059824 0.091090 Available   
  
 Weather Conditions Battery Storage (kWh) Charging Station Capacity (kW) \  
0 Partly Cloudy 16.532408 21.763422   
1 Sunny 39.106930 31.215028   
2 Cloudy 6.112691 46.489116   
3 Partly Cloudy 30.041088 49.675029   
4 Partly Cloudy 45.085422 21.166182   
  
 ... Net Energy Cost ($) Carbon Footprint Reduction (kgCO2) \  
0 ... 0.015428 0.023158   
1 ... 0.035682 0.060875   
2 ... 0.023423 0.006425   
3 ... 0.012969 0.083420   
4 ... 0.004264 0.000476   
  
 Renewable Energy Efficiency EV Charging Efficiency (%) - Normalized Hour \  
0 0.006350 0.87 0   
1 0.005799 0.43 1   
2 0.003567 0.49 2   
3 0.007586 0.64 3   
4 0.007242 0.19 4   
  
 Day Period EV Charging Demand (kW) - Normalized \  
0 Night 0.37   
1 Night 0.95   
2 Night 0.73   
3 Night 0.60   
4 Morning 0.16   
  
 Total Renewable Energy Production (kW) - Normalized Season Month   
0 0.22 Winter 1   
1 0.27 Winter 1   
2 0.25 Winter 1   
3 0.58 Winter 1   
4 0.21 Winter 1   
  
[5 rows x 31 columns]

'''  
Checked unique values in both the dataframe  
'''  
  
for i in df.columns:  
 print(f"Unique values in column '{i}':")  
 print(df[i].unique(),"\n")

Unique values in column 'Date':  
<DatetimeArray>  
['2021-01-01 00:00:00', '2021-01-02 00:00:00', '2021-01-03 00:00:00',  
 '2021-01-04 00:00:00', '2021-01-05 00:00:00', '2021-01-06 00:00:00',  
 '2021-01-07 00:00:00', '2021-01-08 00:00:00', '2021-01-09 00:00:00',  
 '2021-01-10 00:00:00',  
 ...  
 '2024-05-22 00:00:00', '2024-05-23 00:00:00', '2024-05-24 00:00:00',  
 '2024-05-25 00:00:00', '2024-05-26 00:00:00', '2024-05-27 00:00:00',  
 '2024-05-28 00:00:00', '2024-05-29 00:00:00', '2024-05-30 00:00:00',  
 '2024-05-31 00:00:00']  
Length: 1247, dtype: datetime64[ns]   
  
Unique values in column 'Time':  
['00:00:00' '01:00:00' '02:00:00' '03:00:00' '04:00:00' '05:00:00'  
 '06:00:00' '07:00:00' '08:00:00' '09:00:00' '10:00:00' '11:00:00'  
 '12:00:00' '13:00:00' '14:00:00' '15:00:00' '16:00:00' '17:00:00'  
 '18:00:00' '19:00:00' '20:00:00' '21:00:00' '22:00:00' '23:00:00']   
  
Unique values in column 'EV Charging Demand (kW)':  
[0.11236204 0.28521429 0.21959818 ... 0.23297838 0.07779743 0.2481701 ]   
  
Unique values in column 'Solar Energy Production (kW)':  
[0.12538804 0.05269671 0.10503542 ... 0.04761178 0.05229399 0.00880412]   
  
Unique values in column 'Wind Energy Production (kW)':  
[0.00910519 0.10758936 0.0439957 ... 0.13834387 0.27850439 0.18543791]   
  
Unique values in column 'Electricity Price ($/kWh)':  
[0.13731049 0.12510459 0.10666085 ... 0.1780538 0.17047302 0.10660294]   
  
Unique values in column 'Grid Availability':  
['Available' 'Unavailable']   
  
Unique values in column 'Weather Conditions':  
['Partly Cloudy' 'Sunny' 'Cloudy' 'Clear' 'Rainy']   
  
Unique values in column 'Battery Storage (kWh)':  
[16.53240777 39.10693019 6.11269143 ... 7.73881941 45.4788398  
 15.01814069]   
  
Unique values in column 'Charging Station Capacity (kW)':  
[21.76342185 31.21502758 46.48911616 ... 17.60406674 8.63806753  
 14.27427145]   
  
Unique values in column 'EV Charging Efficiency (%)':  
[97.3263759 88.54691271 89.87297148 ... 80.31515304 98.46691092  
 99.89052108]   
  
Unique values in column 'Number of EVs Charging':  
[6 7 4 5 9 8 2 1 3]   
  
Unique values in column 'Peak Demand (kW)':  
[0.15168033 0.57343266 0.97848217 ... 0.97122722 0.9229278 0.12162385]   
  
Unique values in column 'Renewable Energy Usage (%)':  
[25.03906598 55.64989943 79.97078295 ... 86.82746215 84.76931863  
 77.48112723]   
  
Unique values in column 'Grid Stability Index':  
[0.73114739 1.494387 1.10929346 ... 1.24461115 0.56112754 1.07690558]   
  
Unique values in column 'Carbon Emissions (kgCO2/kWh)':  
[0.27494407 0.48125091 0.14607863 ... 0.35722855 0.28634342 0.35170394]   
  
Unique values in column 'Power Outages (hours)':  
[1.88920926 0.2773707 0.64264441 ... 1.41103498 1.61914218 0.91412292]   
  
Unique values in column 'Energy Savings ($)':  
[4.56258096 0.21510366 0.02996864 ... 4.40542951 0.76756544 3.03338893]   
  
Unique values in column 'Total Renewable Energy Production (kW)':  
[0.13449323 0.16028607 0.14903113 ... 0.18595565 0.33079839 0.19424203]   
  
Unique values in column 'Effective Charging Capacity (kW)':  
[21.18154976 27.63994322 41.78115011 ... 14.13873314 8.50563826  
 14.25864413]   
  
Unique values in column 'Adjusted Charging Demand (kW)':  
[0.0281344 0.15872147 0.17561439 ... 0.20228921 0.06594835 0.19228499]   
  
Unique values in column 'Net Energy Cost ($)':  
[0.01542849 0.03568162 0.02342253 ... 0.04148269 0.01326236 0.02645566]   
  
Unique values in column 'Carbon Footprint Reduction (kgCO2)':  
[0.02315789 0.06087479 0.00642509 ... 0.01096305 0.00339291 0.01965501]   
  
Unique values in column 'Renewable Energy Efficiency':  
[0.00634955 0.00579907 0.00356695 ... 0.01315221 0.03889166 0.01362276]   
  
Unique values in column 'EV Charging Efficiency (%) - Normalized':  
[0.87 0.43 0.49 0.64 0.19 0.74 0.84 0.28 0.53 0.35 0.07 0.08 0.06 0.02  
 0.65 0.75 0.98 0.18 0.27 0.72 0.45 0.33 0.81 0.91 0.55 0.56 0.83 0.67  
 0.89 1. 0.95 0.05 0.38 0.63 0.34 0.79 0.17 0.39 0.76 0.68 0.62 0.71  
 0.58 0.14 0.86 0.11 0.04 0.6 0.21 0.22 0.01 0.54 0.73 0.15 0.2 0.88  
 0.03 0.1 0.7 0.93 0.24 0.16 0.37 0.13 0.61 0.42 0.52 0.97 0.78 0.66  
 0.41 0.96 0. 0.36 0.51 0.29 0.8 0.85 0.94 0.46 0.3 0.69 0.25 0.5  
 0.77 0.4 0.99 0.44 0.57 0.48 0.23 0.09 0.82 0.47 0.59 0.9 0.32 0.31  
 0.26 0.12 0.92]   
  
Unique values in column 'Hour':  
[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]   
  
Unique values in column 'Day Period':  
['Night' 'Morning' 'Afternoon' 'Evening']   
  
Unique values in column 'EV Charging Demand (kW) - Normalized':  
[0.37 0.95 0.73 0.6 0.16 0.06 0.87 0.71 0.02 0.97 0.83 0.21 0.18 0.3  
 0.52 0.43 0.29 0.61 0.14 0.46 0.79 0.2 0.51 0.59 0.05 0.17 0.07 0.81  
 0.1 0.68 0.44 0.12 0.5 0.03 0.91 0.26 0.66 0.31 0.55 0.78 0.94 0.89  
 0.92 0.09 0.33 0.39 0.27 0.36 0.28 0.54 0.8 0.99 0.77 0.01 0.82 0.86  
 0.62 0.64 0.47 0.76 0.56 0.49 0.11 0.25 0.41 0.23 0.08 0.93 0.63 0.19  
 0.9 0.32 0.42 0.22 0.34 0.7 0.96 0.04 0.24 0.67 0.84 0.65 0.69 0.88  
 0.53 0.35 0.75 0.85 0.57 0.58 0.72 0.74 0.38 0.13 0.98 0.4 0.45 0.15  
 1. 0.48 0. ]   
  
Unique values in column 'Total Renewable Energy Production (kW) - Normalized':  
[0.22 0.27 0.25 0.58 0.21 0.17 0.29 0.43 0.32 0.7 0.45 0.41 0.3 0.31  
 0.57 0.76 0.9 0.44 0.35 0.15 0.87 0.4 0.51 0.36 0.62 0.85 0.39 0.28  
 0.6 0.83 0.91 0.09 0.38 0.72 0.26 0.52 0.64 0.06 0.55 0.54 0.86 0.02  
 0.34 0.18 0.74 0.33 0.46 0.75 0.19 0.59 0.47 0.11 0.63 0.56 0.82 0.2  
 0.81 0.67 0.66 0.37 0.42 0.73 0.5 0.49 0.14 0.48 0.77 0.05 0.79 0.61  
 0.94 0.53 0.8 0.13 0.08 0.23 0.65 0.69 0.24 0.93 0.97 0.84 0.78 0.88  
 0.16 0.71 0.07 0.68 0.04 0.92 0.01 0.03 0.12 0.1 0.99 0.89 0.98 0.96  
 0.95 1. 0. ]   
  
Unique values in column 'Season':  
['Winter' 'Spring' 'Summer' 'Autumn']   
  
Unique values in column 'Month':  
[ 1 2 3 4 5 6 7 8 9 10 11 12]

df\_dt = df[['Month','Hour','Day Period']]  
df\_dt.head()

Month Hour Day Period  
0 1 0 Night  
1 1 1 Night  
2 1 2 Night  
3 1 3 Night  
4 1 4 Morning

# 6 Decision Tree Classification for Day Period Prediction

## 6.1 Library Imports and Dependencies

This section imports the necessary libraries for building and evaluating a decision tree classification model:

* **Data Manipulation**: pandas and numpy for data handling and numerical operations
* **Model Building**: DecisionTreeClassifier from scikit-learn for tree-based classification
* **Tree Visualization**: plot\_tree for visualizing the decision tree structure
* **Evaluation Metrics**: accuracy\_score and confusion\_matrix for classification performance assessment
* **Visualization**: matplotlib.pyplot and ConfusionMatrixDisplay for creating plots and confusion matrix displays

## 6.2 Feature Selection and Target Definition

The model uses a simple temporal feature set to predict day periods:

* **Features**: Month and Hour - two temporal variables that capture seasonal and daily patterns
* **Target Variable**: Day Period - categorical variable representing different time periods of the day
* **Data Source**: Uses df\_dt dataset specifically prepared for this classification task

This minimal feature set tests whether temporal patterns alone can effectively classify day periods.

## 6.3 Train-Test Split Configuration

The data is partitioned using standard machine learning practices:

* **Training Set**: 80% of the data for model learning and tree construction
* **Test Set**: 20% of the data for unbiased performance evaluation
* **Random State**: Fixed seed (42) ensures reproducible results across different runs

This split ratio provides adequate training data while maintaining sufficient test samples for reliable evaluation.

## 6.4 Decision Tree Model Configuration

The decision tree is configured with specific parameters to balance interpretability and performance:

* **Maximum Leaf Nodes**: Limited to 10 nodes to prevent overfitting and maintain interpretability
* **Random State**: Fixed seed for reproducible model initialization and splitting decisions
* **Algorithm**: Uses the default CART (Classification and Regression Trees) algorithm

The leaf node limitation ensures the tree remains simple enough for human interpretation while capturing key decision patterns.

## 6.5 Model Training and Prediction

The model training process involves:

1. **Tree Construction**: The algorithm builds a decision tree by finding optimal splits based on feature values
2. **Learning Process**: The tree learns to classify day periods based on month and hour patterns
3. **Prediction Generation**: Apply the trained model to test data for performance evaluation

The decision tree creates interpretable rules that can be easily understood by domain experts.

## 6.6 Accuracy Assessment

Model performance is evaluated using classification accuracy:

* **Accuracy Metric**: Percentage of correct predictions out of total predictions
* **Display Format**: Results shown with 2 decimal places for clear interpretation
* **Performance Baseline**: Provides a single metric to assess overall model effectiveness

Accuracy gives a quick overview of how well the model performs on unseen data.

## 6.7 Confusion Matrix Analysis

The confusion matrix provides detailed classification performance insights:

* **Matrix Structure**: Shows true labels vs predicted labels for each class
* **Visual Representation**: Uses blue color scheme with ConfusionMatrixDisplay for clear interpretation
* **Performance Breakdown**: Reveals which day periods are correctly classified and which are confused
* **Error Analysis**: Identifies specific misclassification patterns and potential model weaknesses

This analysis helps understand not just overall accuracy but also class-specific performance.

## 6.8 Decision Tree Visualization

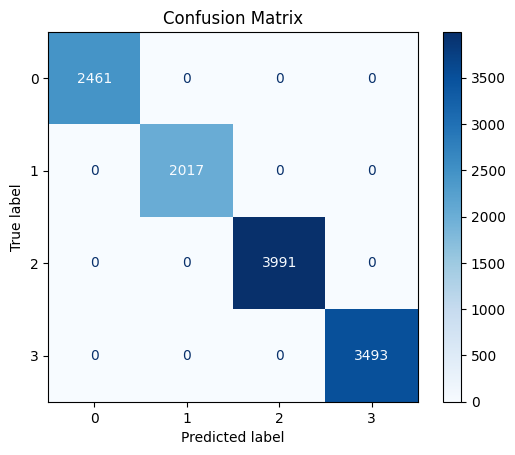
The tree structure visualization reveals the model's decision-making process:

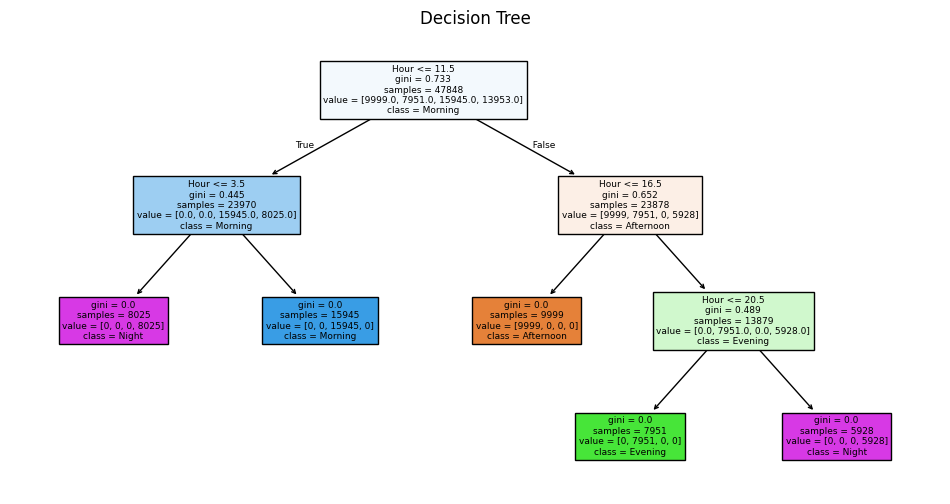
* **Tree Structure**: Shows the complete decision path from root to leaf nodes
* **Feature Names**: Displays actual feature names (Month, Hour) for interpretability
* **Class Labels**: Shows the predicted day period classes at each leaf node
* **Node Coloring**: Filled nodes indicate the dominant class and prediction confidence
* **Decision Rules**: Each split shows the exact threshold values used for classification

This visualization makes the model completely transparent, allowing stakeholders to understand exactly how predictions are made based on temporal patterns.

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier, plot\_tree  
from sklearn.metrics import accuracy\_score, confusion\_matrix, ConfusionMatrixDisplay  
from sklearn.model\_selection import train\_test\_split  
import matplotlib.pyplot as plt  
  
X = df\_dt[['Month','Hour']]  
y = df\_dt['Day Period']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)  
  
model = DecisionTreeClassifier(max\_leaf\_nodes=10,random\_state=42)  
model.fit(X\_train,y\_train)  
  
y\_pred = model.predict(X\_test)  
  
# Accuracy  
acc = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy: {acc:.2f}")  
  
# Confusion Matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm)  
disp.plot(cmap=plt.cm.Blues)  
plt.title("Confusion Matrix")  
plt.show()  
  
# Plot Decision Tree  
plt.figure(figsize=(12,6))  
plot\_tree(model, feature\_names=X.columns, class\_names=[str(c) for c in model.classes\_], filled=True)  
plt.title("Decision Tree")  
plt.show()

Accuracy: 1.00





df\_dt = df[['Month','Hour','EV Charging Demand (kW)','Electricity Price ($/kWh)','Season']]  
df\_dt.head()

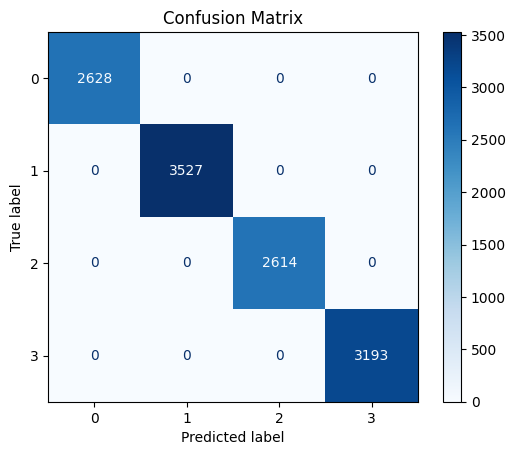
Month Hour EV Charging Demand (kW) Electricity Price ($/kWh) Season  
0 1 0 0.112362 0.137310 Winter  
1 1 1 0.285214 0.125105 Winter  
2 1 2 0.219598 0.106661 Winter  
3 1 3 0.179598 0.072209 Winter  
4 1 4 0.046806 0.091090 Winter

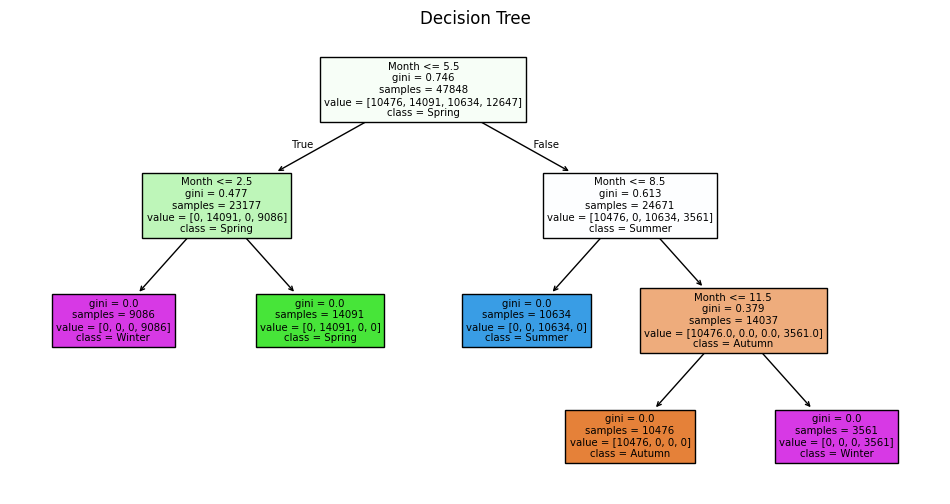
len(df\_dt)\*0.20

11962.0

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier, plot\_tree  
from sklearn.metrics import accuracy\_score, confusion\_matrix, ConfusionMatrixDisplay  
from sklearn.model\_selection import train\_test\_split  
import matplotlib.pyplot as plt  
  
#X = df\_dt[['Month','Hour']]  
#y = df\_dt['Day Period']  
  
X = df\_dt[['Month','Hour','EV Charging Demand (kW)','Electricity Price ($/kWh)']]  
y = df\_dt['Season']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)  
  
model = DecisionTreeClassifier(max\_leaf\_nodes=10,random\_state=42)  
model.fit(X\_train,y\_train)  
  
y\_pred = model.predict(X\_test)  
  
# Accuracy  
acc = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy: {acc:.2f}")  
  
# Confusion Matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm)  
disp.plot(cmap=plt.cm.Blues)  
plt.title("Confusion Matrix")  
plt.show()  
  
# Plot Decision Tree  
plt.figure(figsize=(12,6))  
plot\_tree(model, feature\_names=X.columns, class\_names=[str(c) for c in model.classes\_], filled=True)  
plt.title("Decision Tree")  
plt.show()

Accuracy: 1.00





### 6.8.1 What does this Decision Tree outcome suggest?

* **Main Suggestion:**  
  The Decision Tree is predicting the **Season** using the feature **Month** (and possibly others, but only Month is used in the splits).
* The tree splits the months into four groups, each corresponding to a season: **Winter, Spring, Summer, Autumn**.
* The splits are based on the value of Month (e.g., Month <= 2.5 for Winter, Month <= 5.5 for Spring, etc.).
* This means the model has learned that the month alone is enough to accurately determine the season.

### 6.8.2 What do the different colors in the Decision Tree mean?

* **Colors represent different classes (seasons):**
  + Each color corresponds to a different season (Winter, Spring, Summer, Autumn).
  + For example, all blue nodes might be "Summer", green for "Spring", orange for "Autumn", and purple for "Winter".
* **Helps visualize which class is predicted at each leaf node.**

### 6.8.3 Why is the Gini Index used and what is its purpose?

* **Gini Index** measures how "pure" a node is (how mixed the classes are).
* **Purpose:**
  + A Gini of 0 means all samples in that node belong to one class (perfectly pure).
  + Higher Gini means more mixing of classes (less pure).
* **The tree tries to split the data to make each node as pure as possible.**

### 6.8.4 Cricket Analogy

* **Imagine you are sorting cricket balls by color (red, white, pink, etc.):**
  + You want each box to have only one color.
* **Gini Index:**
  + If a box has only red balls, Gini = 0 (perfect).
  + If a box has half red and half white balls, Gini is higher (not pure).
* **Decision Tree:**
  + At each step, you ask a question (e.g., "Is the ball red?") to split the balls into purer boxes.
  + The tree keeps splitting until each box (leaf node) has only one color (class).

**Summary:**  
The Decision Tree uses the month to perfectly predict the season, with each color showing a different season. The Gini Index helps the tree decide how to split the data to get the purest groups, just like sorting cricket balls by color into the purest boxes.

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 59810 entries, 0 to 59809  
Data columns (total 31 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Date 59810 non-null datetime64[ns]  
 1 Time 59810 non-null object   
 2 EV Charging Demand (kW) 59810 non-null float64   
 3 Solar Energy Production (kW) 59810 non-null float64   
 4 Wind Energy Production (kW) 59810 non-null float64   
 5 Electricity Price ($/kWh) 59810 non-null float64   
 6 Grid Availability 59810 non-null object   
 7 Weather Conditions 59810 non-null object   
 8 Battery Storage (kWh) 59810 non-null float64   
 9 Charging Station Capacity (kW) 59810 non-null float64   
 10 EV Charging Efficiency (%) 59810 non-null float64   
 11 Number of EVs Charging 59810 non-null int64   
 12 Peak Demand (kW) 59810 non-null float64   
 13 Renewable Energy Usage (%) 59810 non-null float64   
 14 Grid Stability Index 59810 non-null float64   
 15 Carbon Emissions (kgCO2/kWh) 59810 non-null float64   
 16 Power Outages (hours) 59810 non-null float64   
 17 Energy Savings ($) 59810 non-null float64   
 18 Total Renewable Energy Production (kW) 59810 non-null float64   
 19 Effective Charging Capacity (kW) 59810 non-null float64   
 20 Adjusted Charging Demand (kW) 59810 non-null float64   
 21 Net Energy Cost ($) 59810 non-null float64   
 22 Carbon Footprint Reduction (kgCO2) 59810 non-null float64   
 23 Renewable Energy Efficiency 59810 non-null float64   
 24 EV Charging Efficiency (%) - Normalized 59810 non-null float64   
 25 Hour 59810 non-null int32   
 26 Day Period 59810 non-null object   
 27 EV Charging Demand (kW) - Normalized 59810 non-null float64   
 28 Total Renewable Energy Production (kW) - Normalized 59810 non-null float64   
 29 Season 59810 non-null object   
 30 Month 59810 non-null int32   
dtypes: datetime64[ns](1), float64(22), int32(2), int64(1), object(5)  
memory usage: 13.7+ MB

df\_dt = df[['Electricity Price ($/kWh)','Hour','Month','Grid Stability Index','Grid Availability']]  
df\_dt.head()

Electricity Price ($/kWh) Hour Month Grid Stability Index \  
0 0.137310 0 1 0.731147   
1 0.125105 1 1 1.494387   
2 0.106661 2 1 1.109293   
3 0.072209 3 1 0.847219   
4 0.091090 4 1 1.452466   
  
 Grid Availability   
0 Available   
1 Available   
2 Available   
3 Available   
4 Available

# 7 Decision Tree Classification for Grid Availability Predictio

## 7.1 Feature Selection and Target Definition

The model uses a comprehensive set of grid-related features to predict grid availability:

* **Features**:
  + Electricity Price ($/kWh): Economic indicator of grid conditions
  + Hour: Temporal pattern capturing daily grid usage cycles
  + Month: Seasonal patterns affecting grid demand and availability
  + Grid Stability Index: Direct measure of grid infrastructure health
* **Target Variable**: Grid Availability - categorical variable indicating whether the grid is available or not
* **Data Source**: Uses df\_dt dataset specifically prepared for grid availability classification

This feature set combines economic, temporal, and technical indicators to predict grid availability status.

## 7.2 Train-Test Split Configuration

The data is partitioned using standard machine learning practices:

* **Training Set**: 80% of the data for model learning and tree construction
* **Test Set**: 20% of the data for unbiased performance evaluation
* **Random State**: Fixed seed (42) ensures reproducible results across different runs

This split ratio provides adequate training data while maintaining sufficient test samples for reliable evaluation of grid availability predictions.

## 7.3 Decision Tree Model Configuration

The decision tree is configured with specific parameters to balance interpretability and performance:

* **Maximum Leaf Nodes**: Limited to 10 nodes to prevent overfitting and maintain interpretability
* **Random State**: Fixed seed for reproducible model initialization and splitting decisions
* **Algorithm**: Uses the default CART (Classification and Regression Trees) algorithm

## 7.4 Confusion Matrix Analysis

The confusion matrix provides detailed classification performance insights for grid availability:

* **Matrix Structure**: Shows true grid availability status vs predicted status for each class

This analysis helps understand not just overall accuracy but also the types of errors the model makes, which is crucial for grid reliability decisions.

## 7.5 Decision Tree Visualization

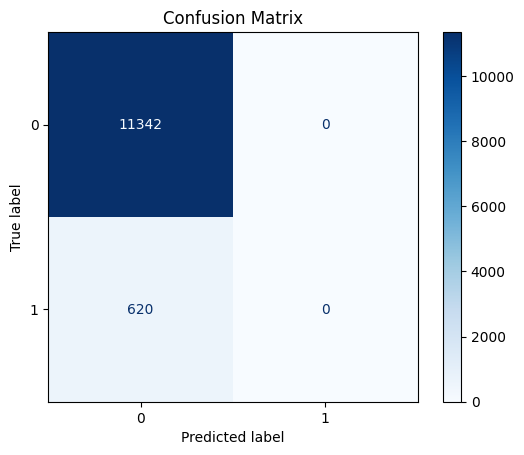
The tree structure visualization reveals the model's decision-making process for grid availability:

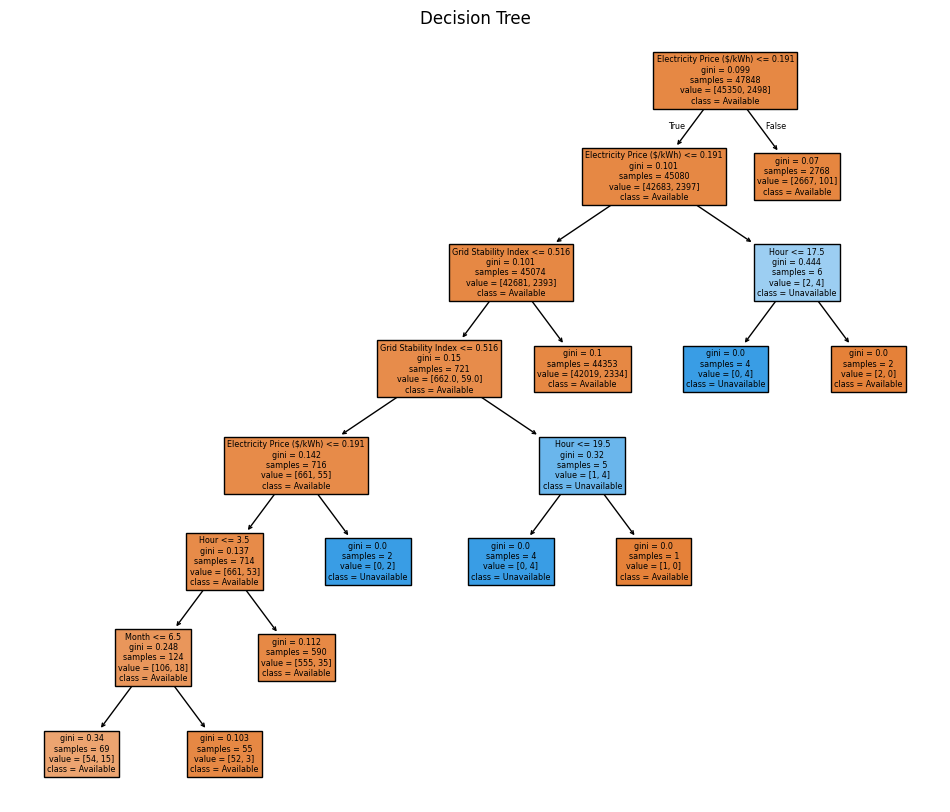
* **Tree Structure**: Shows the complete decision path from root to leaf nodes for grid availability classification
* **Feature Names**: Displays actual feature names (Electricity Price, Hour, Month, Grid Stability Index) for interpretability
* **Class Labels**: Shows the predicted grid availability status at each leaf node
* **Node Coloring**: Filled nodes indicate the dominant class and prediction confidence
* **Decision Rules**: Each split shows the exact threshold values used for classification (e.g., "Grid Stability Index ≤ 0.75")
* **Larger Visualization**: Uses (12,10) figure size to accommodate the complexity of grid availability decision rules

This visualization makes the model completely transparent, allowing grid operators to understand exactly how the system determines grid availability based on economic indicators, temporal patterns, and stability metrics.

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier, plot\_tree  
from sklearn.metrics import accuracy\_score, confusion\_matrix, ConfusionMatrixDisplay  
from sklearn.model\_selection import train\_test\_split  
import matplotlib.pyplot as plt  
  
  
X = df\_dt[['Electricity Price ($/kWh)','Hour','Month','Grid Stability Index']]  
y = df\_dt['Grid Availability']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)  
  
model = DecisionTreeClassifier(max\_leaf\_nodes=10,random\_state=42)  
model.fit(X\_train,y\_train)  
  
y\_pred = model.predict(X\_test)  
  
# Accuracy  
acc = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy: {acc:.2f}")  
  
# Confusion Matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm)  
disp.plot(cmap=plt.cm.Blues)  
plt.title("Confusion Matrix")  
plt.show()  
  
# Plot Decision Tree  
plt.figure(figsize=(12,10))  
plot\_tree(model, feature\_names=X.columns, class\_names=[str(c) for c in model.classes\_], filled=True)  
plt.title("Decision Tree")  
plt.show()

Accuracy: 0.95





**Summary of the Decision Tree Output (Grid Availability Prediction):**

1. **Most Important Feature:**
   * The first and most important split is on **Electricity Price ($/kWh)**. If the price is low (≤ 0.191), the grid is likely to be **Available**.
2. **Secondary Features:**
   * If the price is low, the next important feature is **Grid Stability Index**. Higher stability increases the chance of grid availability.
   * If the price is high, the grid is more likely to be **Unavailable**.
3. **Further Splits:**
   * For cases with borderline stability, the tree checks **Hour** and **Month** to further refine the prediction.
   * Certain hours (e.g., Hour > 17.5 or Hour > 19.5) and months (e.g., Month > 6.5) can indicate a higher risk of grid unavailability.
4. **Class Colors:**
   * **Orange nodes** represent the class **Available**.
   * **Blue nodes** represent the class **Unavailable**.
5. **Gini Index:**
   * Lower Gini values (closer to 0) at the leaves indicate pure predictions (all samples in that node belong to one class).

**Key Takeaways:**

* **Low electricity price and high grid stability** are strong indicators of grid availability.
* **High price, low stability, certain hours, and certain months** increase the risk of grid unavailability.
* The tree uses a combination of price, stability, hour, and month to make its predictions.

**Summary:**

* **Low electricity price (≤ 0.191) and high grid stability (≤ 0.516) with early hours and lower months mostly lead to "Available".**
* **High price, low stability, or certain hours/months increase the chance of "Unavailable".**
* **Each split shows the number of samples and the class distribution at that node.**