icity-demand-and-price-forecasting

November 27, 2024

[ ]:
```
#Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/DATASETS/complete_dataset (1).csv')
df.head()
```

[ ]:

	date	demand	RRP	demand_pos_RRP	RRP_positive	demand_neg_RRP	RRP_negative	frac_at_neg_RRP	min_temperature	max_temperature	solar_exposure	rainfall	school_day	holiday
0	2015-01-01	99635.030	25.633696	97319.240	26.415953	2315.790	-7.240000	0.020833	13.3	26.9	23.6	0.0	N	Y
1	2015-01-02	129606.010	33.138988	121082.015	38.837661	8523.995	-47.809777	0.062500	15.4	38.8	26.8	0.0	N	N
2	2015-01-03	142300.540	34.564855	142300.540	34.564855	0.000	0.000000	0.000000	20.0	38.2	26.5	0.0	N	N
3	2015-01-04	104330.715	25.005560	104330.715	25.005560	0.000	0.000000	0.000000	16.3	21.4	25.2	4.2	N	N
4	2015-01-05	118132.200	26.724176	118132.200	26.724176	0.000	0.000000	0.000000	15.0	22.0	30.7	0.0	N	N

[ ]:
```
#Convert date to date time format
df['date'] = pd.to_datetime(df['date'])
print(df)
```

	date	demand	RRP	demand_pos_RRP	RRP_positive	\
0	2015-01-01	99635.030	25.633696	97319.240	26.415953	
1	2015-01-02	129606.010	33.138988	121082.015	38.837661	

```
2    2015-01-03  142300.540  34.564855      142300.540     34.564855
3    2015-01-04  104330.715  25.005560      104330.715     25.005560
4    2015-01-05  118132.200  26.724176      118132.200     26.724176
…           …           …          …               …            …
2101 2020-10-02   99585.835  -6.076028       41988.240     26.980251
2102 2020-10-03   92277.025  -1.983471       44133.510     32.438156
2103 2020-10-04   94081.565  25.008614       88580.995     26.571687
2104 2020-10-05  113610.030  36.764701      106587.375     39.616015
2105 2020-10-06  122607.560  75.771059      122607.560     75.771059

      demand_neg_RRP  RRP_negative  frac_at_neg_RRP  min_temperature  \
0            2315.790     -7.240000         0.020833             13.3
1            8523.995    -47.809777         0.062500             15.4
2               0.000      0.000000         0.000000             20.0
3               0.000      0.000000         0.000000             16.3
4               0.000      0.000000         0.000000             15.0
…                 …            …                …                …
2101        57597.595    -30.173823         0.625000             12.8
2102        48143.515    -33.538025         0.583333             17.4
2103         5500.570     -0.163066         0.062500             13.5
2104         7022.655     -6.511550         0.083333              9.1
2105            0.000      0.000000         0.000000              8.9

      max_temperature  solar_exposure  rainfall school_day holiday
0                26.9            23.6       0.0          N       Y
1                38.8            26.8       0.0          N       N
2                38.2            26.5       0.0          N       N
3                21.4            25.2       4.2          N       N
4                22.0            30.7       0.0          N       N
…                 …               …         …          …       …
2101             26.0            22.0       0.0          N       N
2102             29.4            19.8       0.0          N       N
2103             29.5             8.4       0.0          N       N
2104             12.7             7.3      12.8          N       N
2105             12.6             5.8       1.0          N       N

[2106 rows x 14 columns]
```

[ ]:
```
# Basic information about the dataset
print(df.info())
print("\nBasic statistics:")
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2106 entries, 0 to 2105
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
```

```
 0   date             2106 non-null   datetime64[ns]
 1   demand           2106 non-null   float64
 2   RRP              2106 non-null   float64
 3   demand_pos_RRP   2106 non-null   float64
 4   RRP_positive     2106 non-null   float64
 5   demand_neg_RRP   2106 non-null   float64
 6   RRP_negative     2106 non-null   float64
 7   frac_at_neg_RRP  2106 non-null   float64
 8   min_temperature  2106 non-null   float64
 9   max_temperature  2106 non-null   float64
 10  solar_exposure   2105 non-null   float64
 11  rainfall         2103 non-null   float64
 12  school_day       2106 non-null   object
 13  holiday          2106 non-null   object
dtypes: datetime64[ns](1), float64(11), object(2)
memory usage: 230.5+ KB
None

Basic statistics:
                      date         demand           RRP  demand_pos_RRP  \
count                 2106    2106.000000  2106.000000     2106.000000
mean   2017-11-18 12:00:00  120035.476503    76.079554   119252.305055
min    2015-01-01 00:00:00   85094.375000    -6.076028    41988.240000
25%    2016-06-10 06:00:00  109963.650000    38.707040   109246.250000
50%    2017-11-18 12:00:00  119585.912500    66.596738   119148.082500
75%    2019-04-28 18:00:00  130436.006250    95.075012   130119.477500
max    2020-10-06 00:00:00  170653.840000  4549.645105   170653.840000
std                    NaN   13747.993761   130.246805    14818.631319

       RRP_positive  demand_neg_RRP  RRP_negative  frac_at_neg_RRP  \
count   2106.000000     2106.000000   2106.000000      2106.000000
mean      76.553847      783.171448     -2.686052         0.008547
min       13.568986        0.000000   -342.220000         0.000000
25%       39.117361        0.000000      0.000000         0.000000
50%       66.869058        0.000000      0.000000         0.000000
75%       95.130181        0.000000      0.000000         0.000000
max     4549.645105    57597.595000      0.000000         0.625000
std      130.114184     3578.920686     19.485432         0.039963

       min_temperature  max_temperature  solar_exposure     rainfall
count      2106.000000      2106.000000     2105.000000  2103.000000
mean         11.582289        20.413200       14.743373     1.505944
min           0.600000         9.000000        0.700000     0.000000
25%           8.500000        15.525000        8.200000     0.000000
50%          11.300000        19.100000       12.700000     0.000000
75%          14.600000        23.900000       20.700000     0.800000
max          28.000000        43.500000       33.300000    54.600000
std           4.313711         6.288693        7.945527     4.307897
```

[ ]:
```
# Handle missing values
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
print(df)
```

```
           date      demand        RRP  demand_pos_RRP  RRP_positive  \
0    2015-01-01   99635.030  25.633696       97319.240     26.415953
1    2015-01-02  129606.010  33.138988      121082.015     38.837661
2    2015-01-03  142300.540  34.564855      142300.540     34.564855
3    2015-01-04  104330.715  25.005560      104330.715     25.005560
4    2015-01-05  118132.200  26.724176      118132.200     26.724176
…           …         …         …               …             …
2101 2020-10-02   99585.835  -6.076028       41988.240     26.980251
2102 2020-10-03   92277.025  -1.983471       44133.510     32.438156
2103 2020-10-04   94081.565  25.008614       88580.995     26.571687
2104 2020-10-05  113610.030  36.764701      106587.375     39.616015
2105 2020-10-06  122607.560  75.771059      122607.560     75.771059

      demand_neg_RRP  RRP_negative  frac_at_neg_RRP  min_temperature  \
0            2315.790     -7.240000         0.020833             13.3
1            8523.995    -47.809777         0.062500             15.4
2               0.000      0.000000         0.000000             20.0
3               0.000      0.000000         0.000000             16.3
4               0.000      0.000000         0.000000             15.0
…               …            …                …                …
2101        57597.595    -30.173823         0.625000             12.8
2102        48143.515    -33.538025         0.583333             17.4
2103         5500.570     -0.163066         0.062500             13.5
2104         7022.655     -6.511550         0.083333              9.1
2105            0.000      0.000000         0.000000              8.9

      max_temperature  solar_exposure  rainfall school_day holiday
0                26.9            23.6       0.0          N       Y
1                38.8            26.8       0.0          N       N
2                38.2            26.5       0.0          N       N
3                21.4            25.2       4.2          N       N
4                22.0            30.7       0.0          N       N
…                …               …          …            …       …
2101             26.0            22.0       0.0          N       N
2102             29.4            19.8       0.0          N       N
2103             29.5             8.4       0.0          N       N
2104             12.7             7.3      12.8          N       N
2105             12.6             5.8       1.0          N       N

[2106 rows x 14 columns]
```

[ ]:
```
# Remove duplicates
df.drop_duplicates(inplace=True)
```

[ ]:
```
# Scale all numeric columns
scaler = MinMaxScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
print(df)
```

```
           date    demand       RRP  demand_pos_RRP  RRP_positive  \
0    2015-01-01  0.169948  0.006960        0.430037      0.002832
1    2015-01-02  0.520242  0.008608        0.614724      0.005571
2    2015-01-03  0.668613  0.008921        0.779636      0.004629
3    2015-01-04  0.224830  0.006823        0.484531      0.002521
4    2015-01-05  0.386139  0.007200        0.591797      0.002900
…           …         …         …               …             …
2101 2020-10-02  0.169373  0.000000        0.000000      0.002957
2102 2020-10-03  0.083949  0.000898        0.016673      0.004160
2103 2020-10-04  0.105040  0.006823        0.362123      0.002867
2104 2020-10-05  0.333285  0.009404        0.502070      0.005742
2105 2020-10-06  0.438446  0.017966        0.626580      0.013713

      demand_neg_RRP  RRP_negative  frac_at_neg_RRP  min_temperature  \
0           0.040206      0.978844         0.033333         0.463504
1           0.147992      0.860295         0.100000         0.540146
2           0.000000      1.000000         0.000000         0.708029
3           0.000000      1.000000         0.000000         0.572993
4           0.000000      1.000000         0.000000         0.525547
…                …             …                …                …
2101        1.000000      0.911829         1.000000         0.445255
2102        0.835860      0.901999         0.933333         0.613139
2103        0.095500      0.999524         0.100000         0.470803
2104        0.121926      0.980973         0.133333         0.310219
2105        0.000000      1.000000         0.000000         0.302920

      max_temperature  solar_exposure  rainfall school_day holiday
0            0.518841        0.702454  0.000000          N       Y
1            0.863768        0.800613  0.000000          N       N
2            0.846377        0.791411  0.000000          N       N
3            0.359420        0.751534  0.076923          N       N
4            0.376812        0.920245  0.000000          N       N
…               …              …         …          …       …
2101         0.492754        0.653374  0.000000          N       N
2102         0.591304        0.585890  0.000000          N       N
2103         0.594203        0.236196  0.000000          N       N
2104         0.107246        0.202454  0.234432          N       N
2105         0.104348        0.156442  0.018315          N       N
```

[2106 rows x 14 columns]

[ ]:
```
#Encode categorical variables
le = LabelEncoder()
df['school_day'] = le.fit_transform(df['school_day'])
df['holiday'] = le.fit_transform(df['holiday'])
print(df)
```

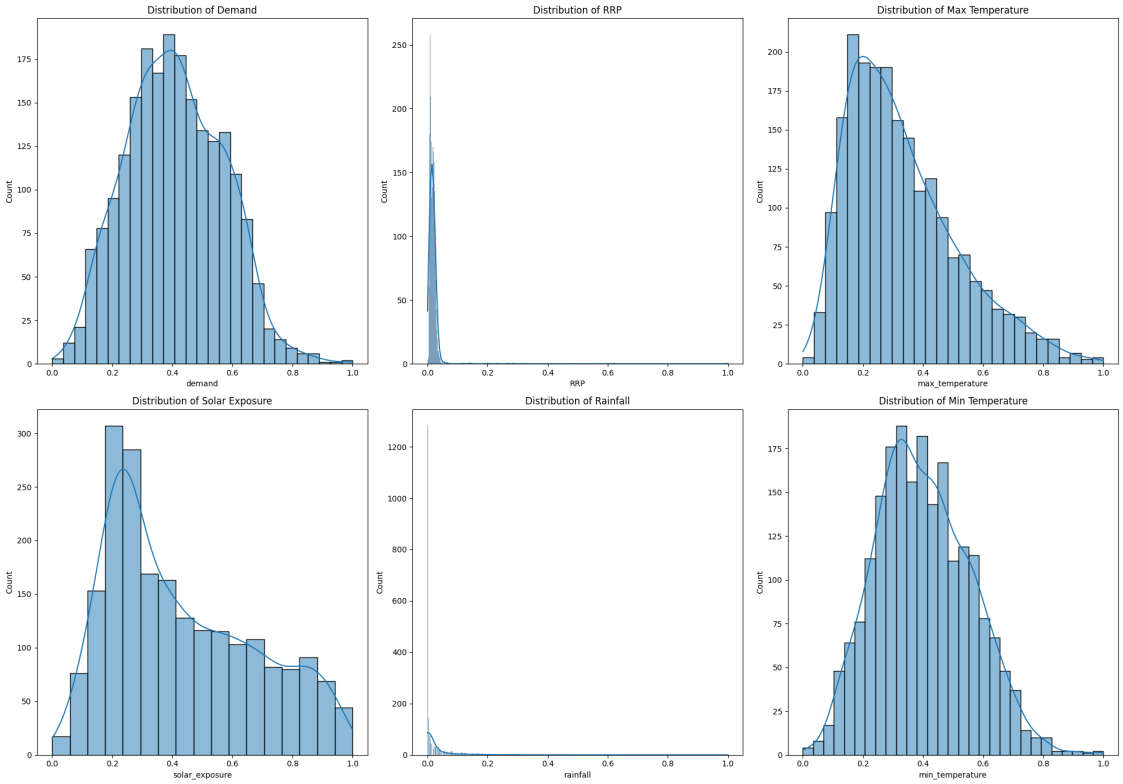
```
           date    demand       RRP  demand_pos_RRP  RRP_positive  \
0    2015-01-01  0.169948  0.006960        0.430037      0.002832
1    2015-01-02  0.520242  0.008608        0.614724      0.005571
2    2015-01-03  0.668613  0.008921        0.779636      0.004629
3    2015-01-04  0.224830  0.006823        0.484531      0.002521
4    2015-01-05  0.386139  0.007200        0.591797      0.002900
…           …         …         …               …             …
2101 2020-10-02  0.169373  0.000000        0.000000      0.002957
2102 2020-10-03  0.083949  0.000898        0.016673      0.004160
2103 2020-10-04  0.105040  0.006823        0.362123      0.002867
2104 2020-10-05  0.333285  0.009404        0.502070      0.005742
2105 2020-10-06  0.438446  0.017966        0.626580      0.013713

      demand_neg_RRP  RRP_negative  frac_at_neg_RRP  min_temperature  \
0           0.040206      0.978844         0.033333         0.463504
1           0.147992      0.860295         0.100000         0.540146
2           0.000000      1.000000         0.000000         0.708029
3           0.000000      1.000000         0.000000         0.572993
4           0.000000      1.000000         0.000000         0.525547
…                …             …                …                …
2101        1.000000      0.911829         1.000000         0.445255
2102        0.835860      0.901999         0.933333         0.613139
2103        0.095500      0.999524         0.100000         0.470803
2104        0.121926      0.980973         0.133333         0.310219
2105        0.000000      1.000000         0.000000         0.302920

      max_temperature  solar_exposure  rainfall  school_day  holiday
0            0.518841        0.702454  0.000000           0        1
1            0.863768        0.800613  0.000000           0        0
2            0.846377        0.791411  0.000000           0        0
3            0.359420        0.751534  0.076923           0        0
4            0.376812        0.920245  0.000000           0        0
…                …               …         …           …        …
2101         0.492754        0.653374  0.000000           0        0
2102         0.591304        0.585890  0.000000           0        0
2103         0.594203        0.236196  0.000000           0        0
2104         0.107246        0.202454  0.234432           0        0
2105         0.104348        0.156442  0.018315           0        0

[2106 rows x 14 columns]
```

[ ]:
```
# Extract additional date features
df['day'] = df['date'].dt.day
df['month'] = df['date'].dt.month
df['year'] = df['date'].dt.year
print(df)
```

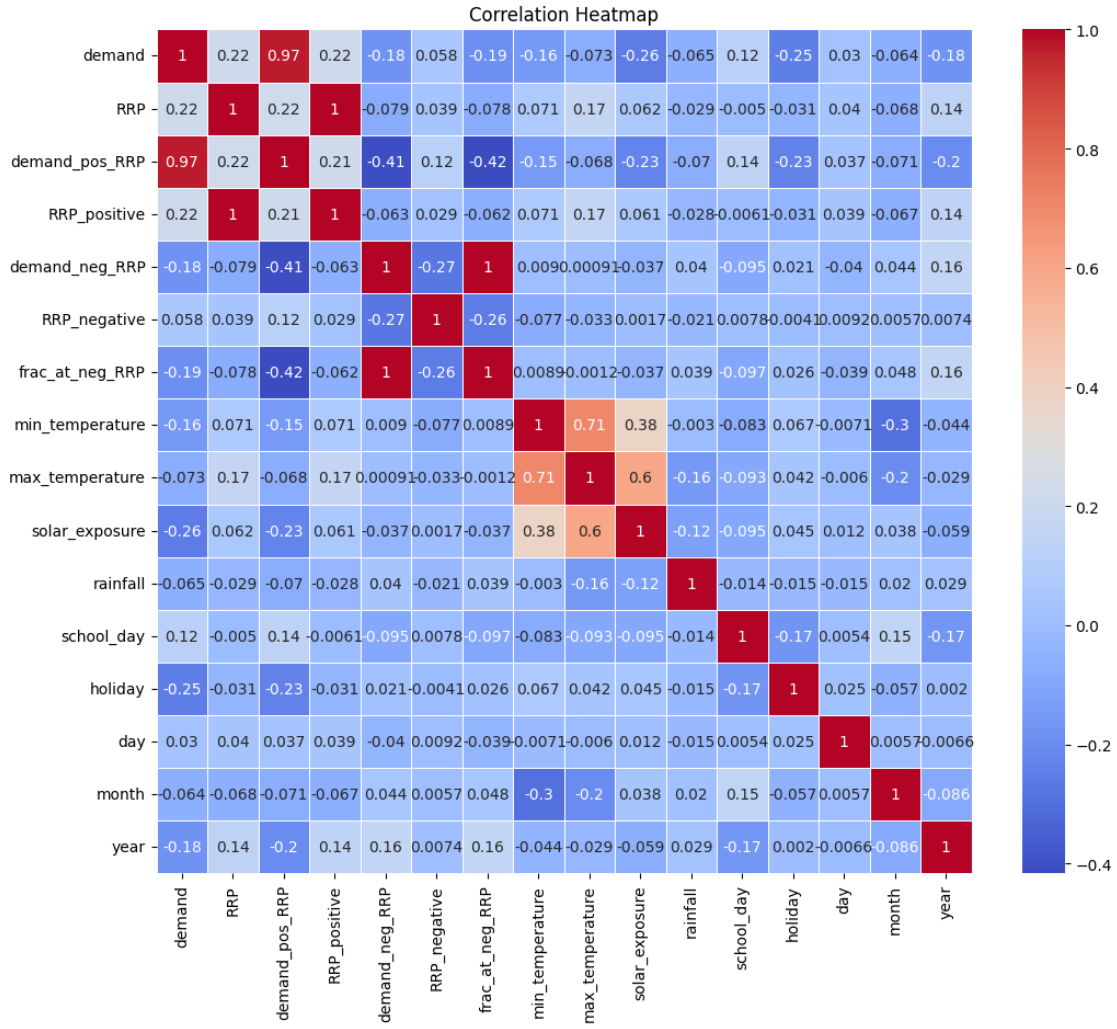
```
           date    demand       RRP  demand_pos_RRP  RRP_positive  \
0    2015-01-01  0.169948  0.006960        0.430037      0.002832
1    2015-01-02  0.520242  0.008608        0.614724      0.005571
2    2015-01-03  0.668613  0.008921        0.779636      0.004629
3    2015-01-04  0.224830  0.006823        0.484531      0.002521
4    2015-01-05  0.386139  0.007200        0.591797      0.002900
…           …         …         …               …             …
2101 2020-10-02  0.169373  0.000000        0.000000      0.002957
2102 2020-10-03  0.083949  0.000898        0.016673      0.004160
2103 2020-10-04  0.105040  0.006823        0.362123      0.002867
2104 2020-10-05  0.333285  0.009404        0.502070      0.005742
2105 2020-10-06  0.438446  0.017966        0.626580      0.013713

      demand_neg_RRP  RRP_negative  frac_at_neg_RRP  min_temperature  \
0           0.040206      0.978844         0.033333         0.463504
1           0.147992      0.860295         0.100000         0.540146
2           0.000000      1.000000         0.000000         0.708029
3           0.000000      1.000000         0.000000         0.572993
4           0.000000      1.000000         0.000000         0.525547
…                …             …                …                …
2101        1.000000      0.911829         1.000000         0.445255
2102        0.835860      0.901999         0.933333         0.613139
2103        0.095500      0.999524         0.100000         0.470803
2104        0.121926      0.980973         0.133333         0.310219
2105        0.000000      1.000000         0.000000         0.302920

      max_temperature  solar_exposure  rainfall  school_day  holiday  day  \
0            0.518841        0.702454  0.000000           0        1    1
1            0.863768        0.800613  0.000000           0        0    2
2            0.846377        0.791411  0.000000           0        0    3
3            0.359420        0.751534  0.076923           0        0    4
4            0.376812        0.920245  0.000000           0        0    5
…                 …               …         …           …        …  …
2101         0.492754        0.653374  0.000000           0        0    2
2102         0.591304        0.585890  0.000000           0        0    3
2103         0.594203        0.236196  0.000000           0        0    4
2104         0.107246        0.202454  0.234432           0        0    5
2105         0.104348        0.156442  0.018315           0        0    6

      month  year
0         1  2015
```

```
1          1  2015
2          1  2015
3          1  2015
4          1  2015
…      …   …
2101      10  2020
2102      10  2020
2103      10  2020
2104      10  2020
2105      10  2020

[2106 rows x 17 columns]
```

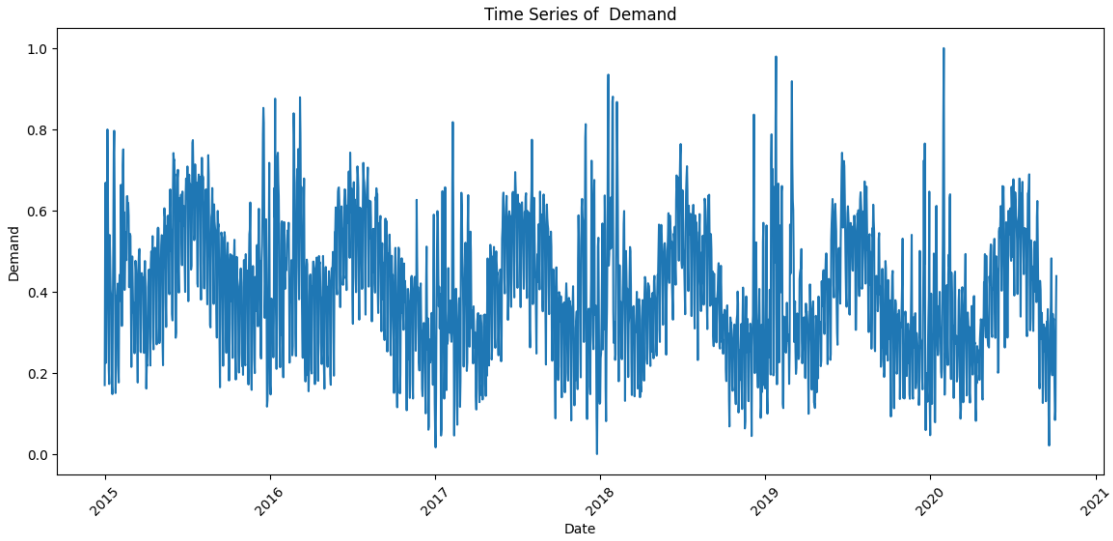
[ ]:
```python
# 1. Visualize distributions of key variables
fig, axes = plt.subplots(2, 3, figsize=(20, 14))
sns.histplot(df['demand'], kde=True, ax=axes[0, 0])
axes[0, 0].set_title('Distribution of Demand')
sns.histplot(df['RRP'], kde=True, ax=axes[0, 1])
axes[0, 1].set_title('Distribution of RRP')
sns.histplot(df['max_temperature'], kde=True, ax=axes[0, 2])
axes[0, 2].set_title('Distribution of Max Temperature')
sns.histplot(df['solar_exposure'], kde=True, ax=axes[1, 0])
axes[1, 0].set_title('Distribution of Solar Exposure')
sns.histplot(df['rainfall'], kde=True, ax=axes[1, 1])
axes[1, 1].set_title('Distribution of Rainfall')
sns.histplot(df['min_temperature'], kde=True, ax=axes[1, 2])
axes[1, 2].set_title('Distribution of Min Temperature')
plt.tight_layout()
plt.show()
```



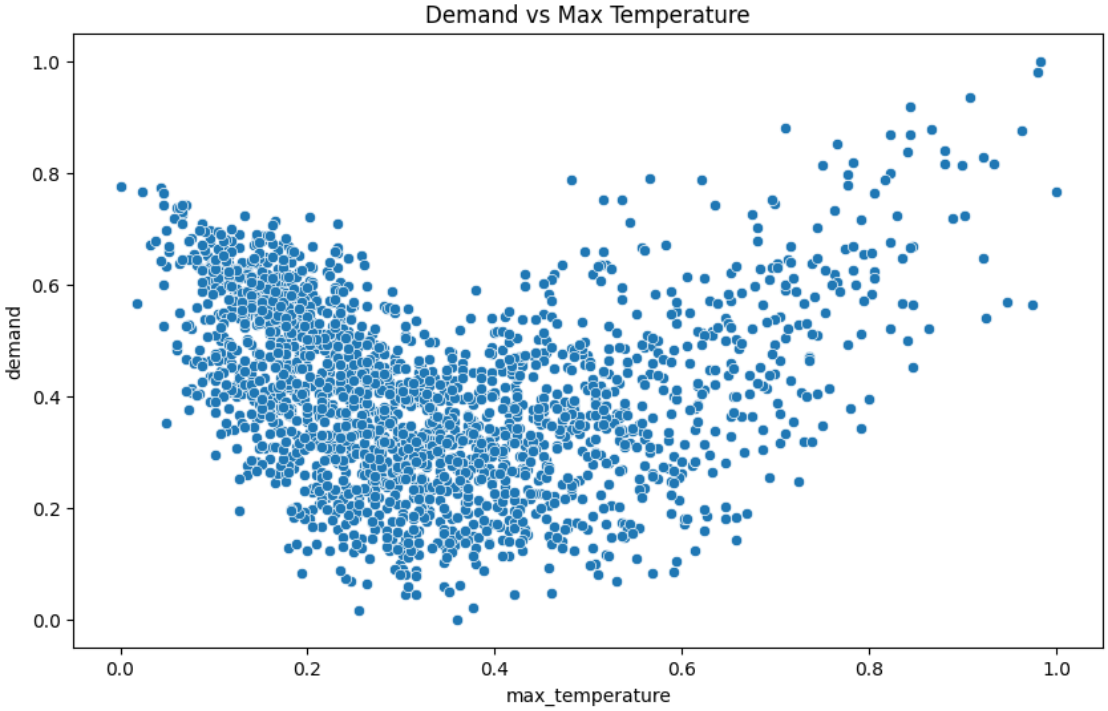
[ ]:
```
# Correlation heatmap
plt.figure(figsize=(12, 10))
corr = df.select_dtypes(include=[np.number]).corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



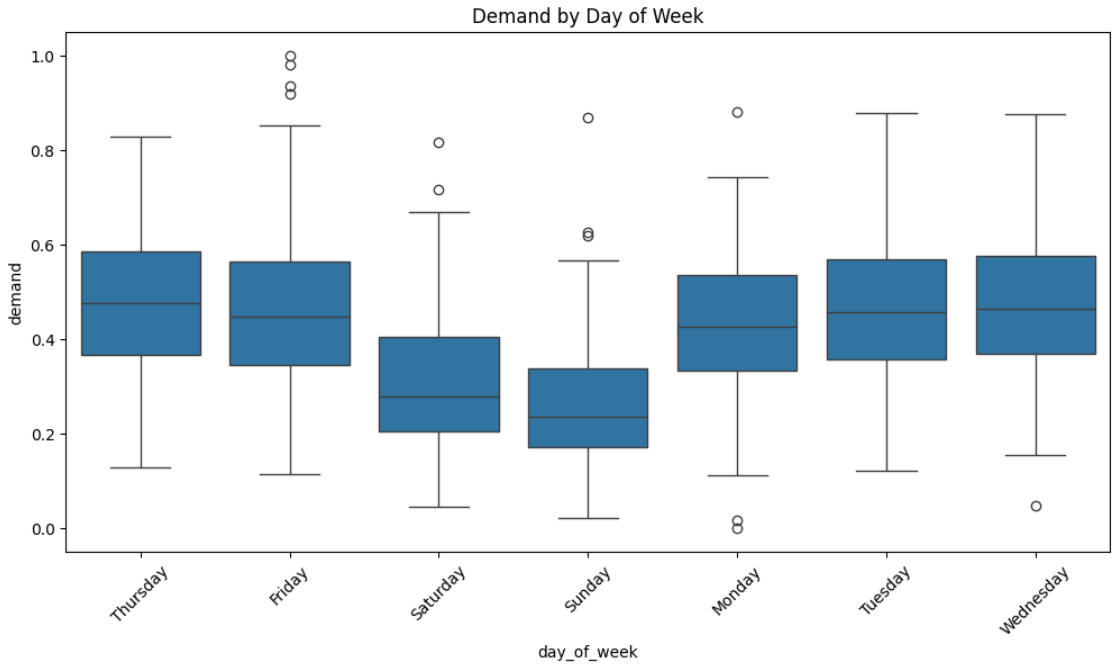
[ ]:
```
# Time series plot Electricity of demand
plt.figure(figsize=(14, 6))
plt.plot(df['date'], df['demand'])
plt.title('Time Series of  Demand')
plt.xlabel('Date')
plt.ylabel('Demand')
plt.xticks(rotation=45)
plt.show()
```



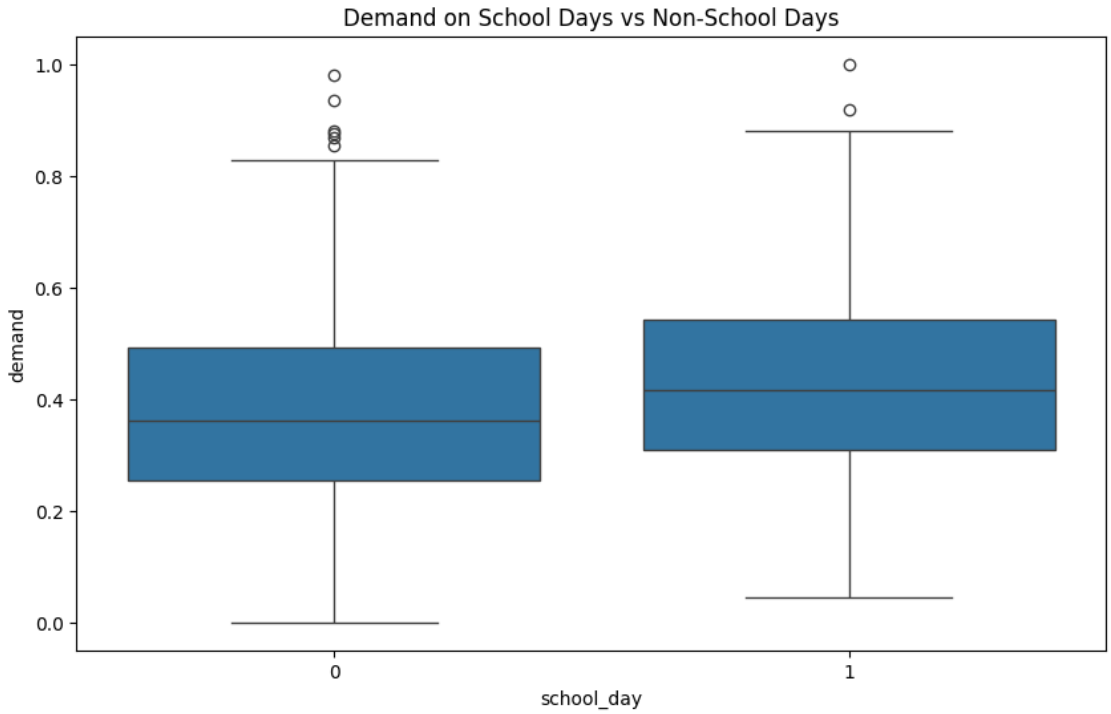
[ ]:
```
#Scatter plot: Demand vs Max Temperature
plt.figure(figsize=(10, 6))
sns.scatterplot(x='max_temperature', y='demand', data=df)
plt.title('Demand vs Max Temperature')
plt.show()
```



[ ]:
```
# Box plot: Demand by Day of Week
df['day_of_week'] = df['date'].dt.day_name()
plt.figure(figsize=(12, 6))
sns.boxplot(x='day_of_week', y='demand', data=df)
plt.title('Demand by Day of Week')
plt.xticks(rotation=45)
plt.show()
```



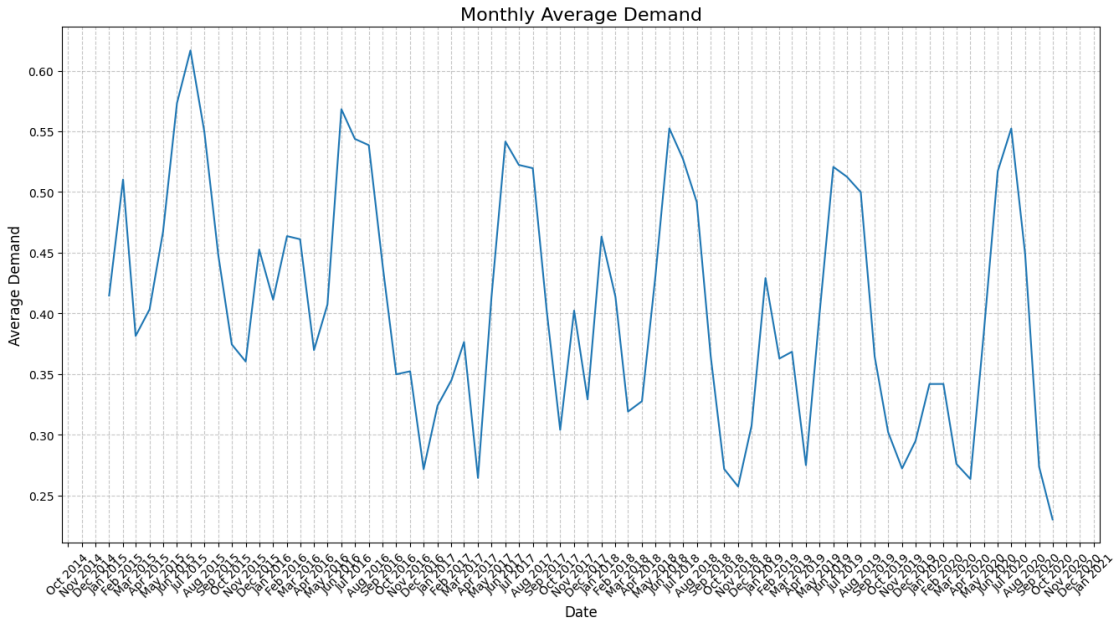
[ ]:
```
# Demand on School Days vs Holidays
plt.figure(figsize=(10, 6))
sns.boxplot(x='school_day', y='demand', data=df)
plt.title('Demand on School Days vs Non-School Days')
plt.show()

print("\nAverage demand on holidays vs non-holidays:")
print(df.groupby('holiday')['demand'].mean())
```



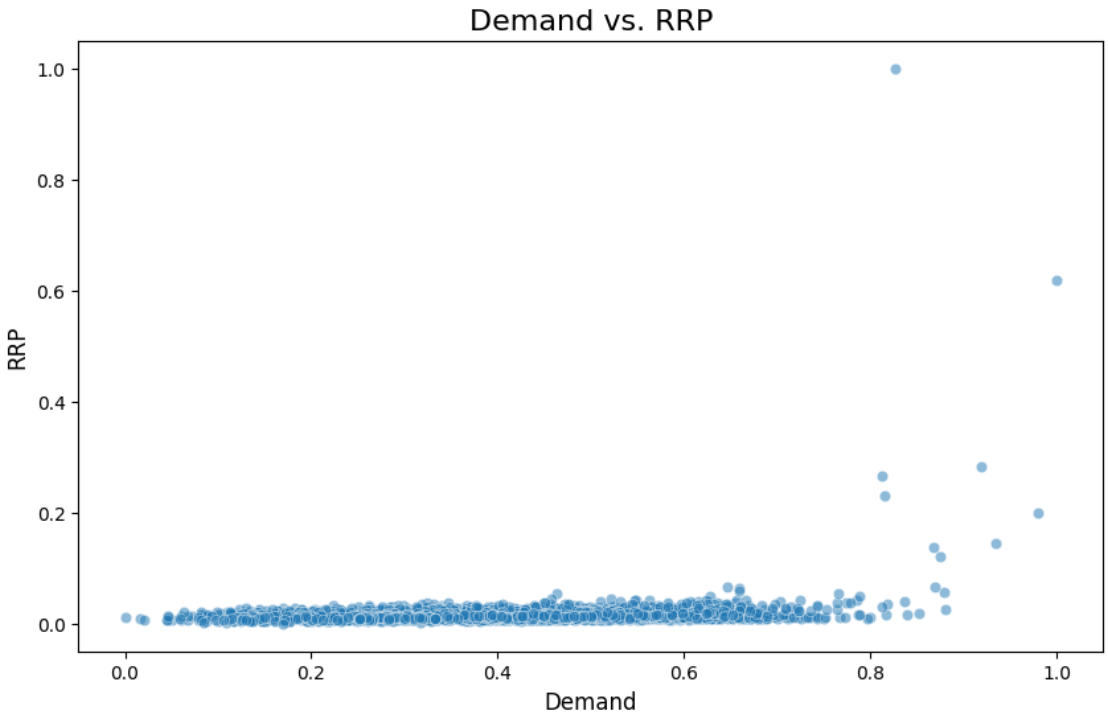

```
Average demand on holidays vs non-holidays:
holiday
0    0.416135
1    0.204135
Name: demand, dtype: float64
```

[ ]:
```python
import matplotlib.dates as mdates

# 7. Monthly average demand
monthly_demand = df.groupby(df['date'].dt.to_period('M'))['demand'].mean().
  ↪reset_index()
monthly_demand['date'] = monthly_demand['date'].dt.to_timestamp()

plt.figure(figsize=(16, 8))
plt.plot(monthly_demand['date'], monthly_demand['demand'])
plt.title('Monthly Average Demand', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Average Demand', fontsize=12)

# Set x-axis major locator to show only months
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
```

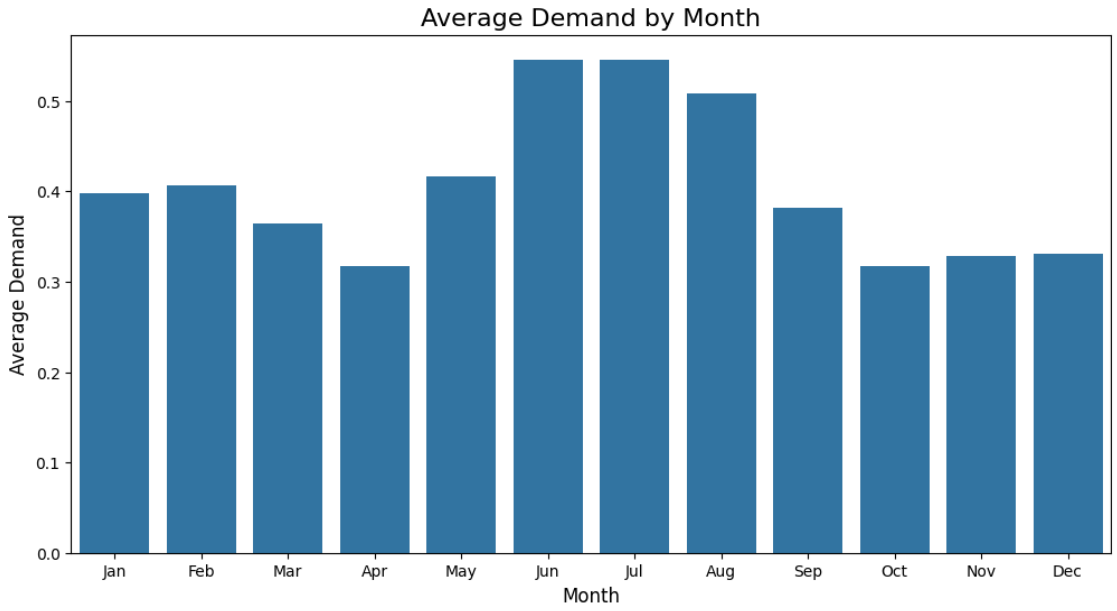
```
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```



[ ]:
```
# Demand vs. RRP scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='demand', y='RRP', data=df, alpha=0.5)
plt.title('Demand vs. RRP', fontsize=16)
plt.xlabel('Demand', fontsize=12)
plt.ylabel('RRP', fontsize=12)
plt.show()
```

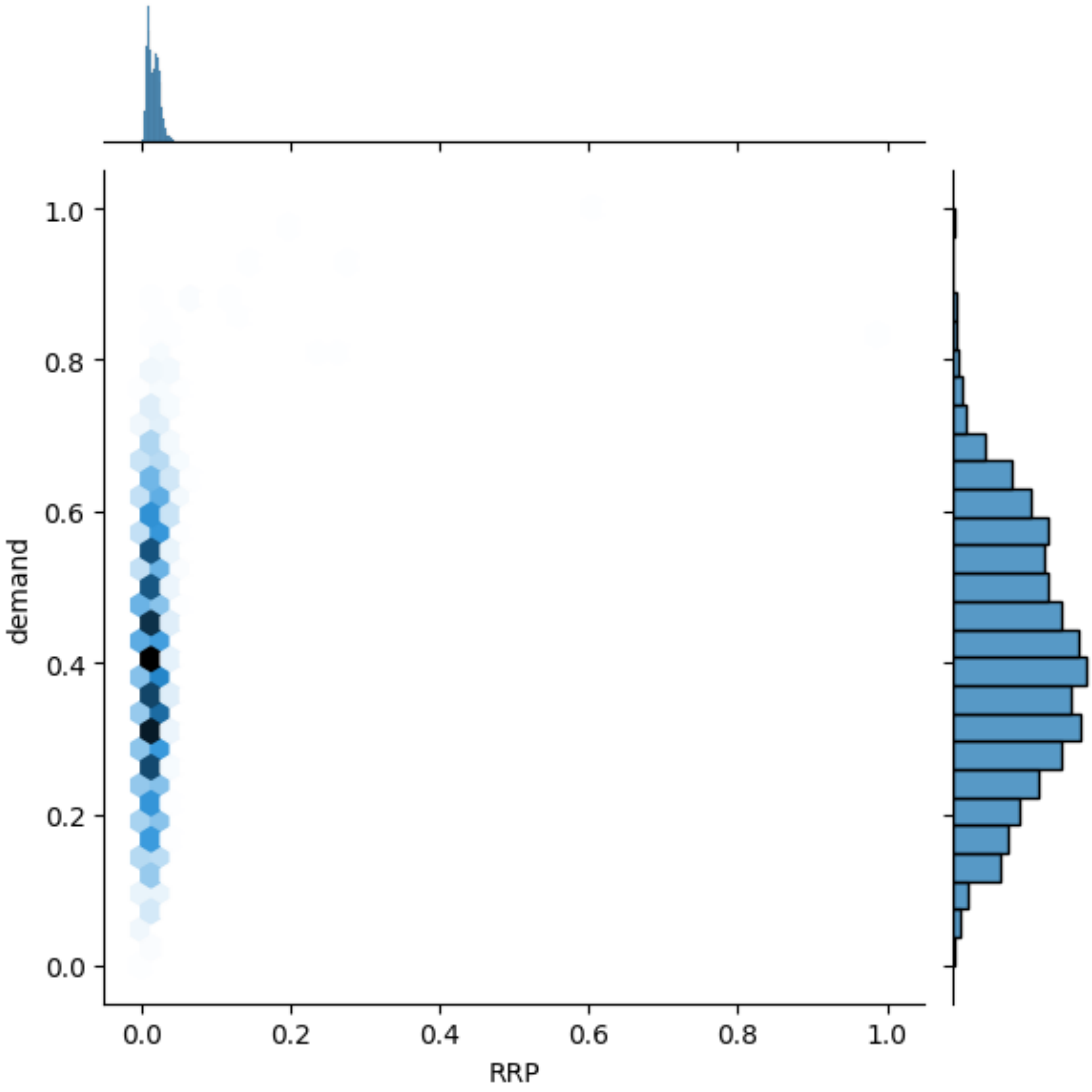


[ ]:
```
# Additional analysis: Demand by month
monthly_demand = df.groupby(df['date'].dt.month)['demand'].mean().reset_index()
monthly_demand['month'] = monthly_demand['date'].map({1:'Jan', 2:'Feb', 3:
 ↪'Mar', 4:'Apr', 5:'May', 6:'Jun',
                                                      7:'Jul', 8:'Aug', 9:
 ↪'Sep', 10:'Oct', 11:'Nov', 12:'Dec'})

plt.figure(figsize=(12, 6))
sns.barplot(x='month', y='demand', data=monthly_demand)
plt.title('Average Demand by Month', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Average Demand', fontsize=12)
plt.show()
```

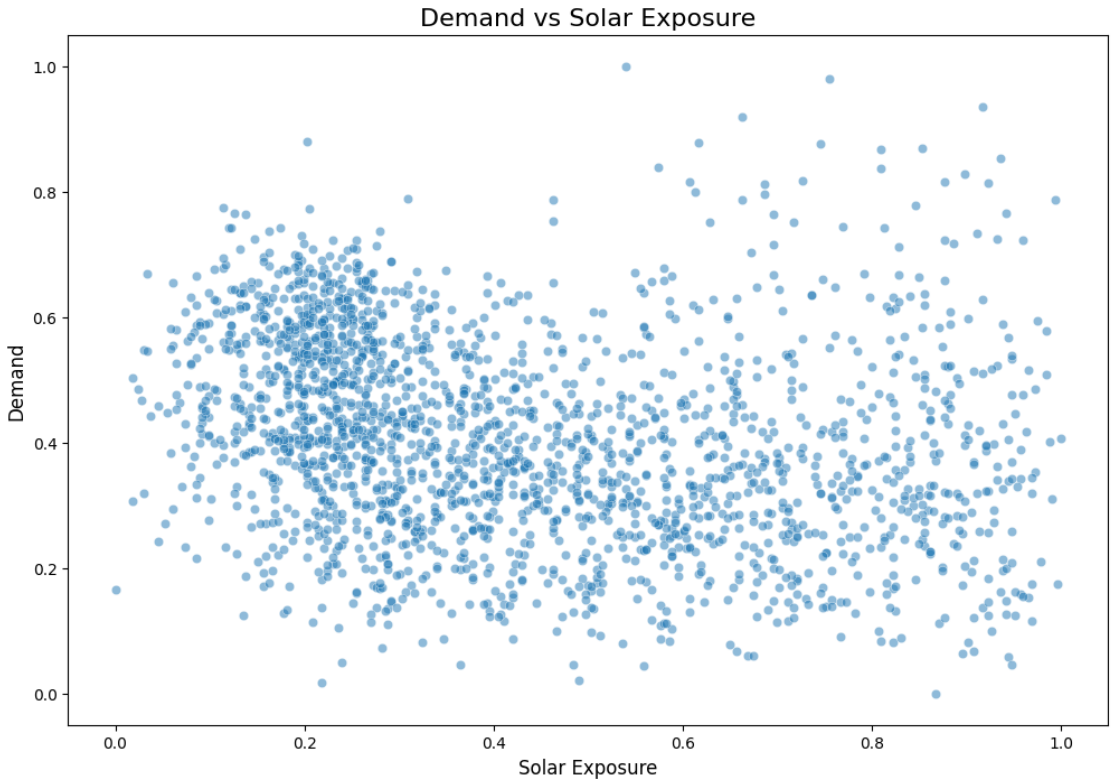


[ ]:
```
#Demand vs. RRP Joint Plot
plt.figure(figsize=(12, 10))
sns.jointplot(x='RRP', y='demand', data=df, kind='hex')
plt.suptitle('Demand vs RRP', y=1.5, fontsize=16)
plt.show()
```

```
<Figure size 1200x1000 with 0 Axes>
```


Demand vs RRP
demand
RRP
0.0
0.2
0.4
0.6
0.8
1.0

[ ]:
```
#Demand vs Solar Exposure scatter plot
plt.figure(figsize=(12, 8))
sns.scatterplot(x='solar_exposure', y='demand', data=df, alpha=0.5)
plt.title('Demand vs Solar Exposure', fontsize=16)
plt.xlabel('Solar Exposure', fontsize=12)
plt.ylabel('Demand', fontsize=12)
plt.show()
plt.show()
```



[ ]:
```
duplicate_rows = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_rows}")
```

```
Number of duplicate rows: 0
```

[ ]:
```
duplicate_columns = df.duplicated().sum()
print(f"Number of duplicate columns: {duplicate_columns}")
```

```
Number of duplicate columns: 0
```

[ ]:
```
null_values = df.isnull().sum()
print("Number of null values of each column:\n",null_values)
```

```
Number of null values of each column:
 date                0
demand              0
RRP                 0
demand_pos_RRP      0
RRP_positive        0
demand_neg_RRP      0
RRP_negative        0
frac_at_neg_RRP     0
min_temperature     0
max_temperature     0
solar_exposure      0
rainfall            0
school_day          0
holiday             0
day                 0
month               0
year                0
day_of_week         0
dtype: int64
```

[ ]:
```
# Final DataFrame shape
print(f"\nDataFrame shape after preprocessing: {df.shape}")
display(df)
```

```
DataFrame shape after preprocessing: (2106, 18)
```

	date	demand	RRP	demand_pos_RRP	RRP_positive	\
0	2015-01-01	0.169948	0.006960	0.430037	0.002832	
1	2015-01-02	0.520242	0.008608	0.614724	0.005571	
2	2015-01-03	0.668613	0.008921	0.779636	0.004629	
3	2015-01-04	0.224830	0.006823	0.484531	0.002521	
4	2015-01-05	0.386139	0.007200	0.591797	0.002900	
…	…	…	…	…	…	
2101	2020-10-02	0.169373	0.000000	0.000000	0.002957	
2102	2020-10-03	0.083949	0.000898	0.016673	0.004160	
2103	2020-10-04	0.105040	0.006823	0.362123	0.002867	
2104	2020-10-05	0.333285	0.009404	0.502070	0.005742	
2105	2020-10-06	0.438446	0.017966	0.626580	0.013713	

	demand_neg_RRP	RRP_negative	frac_at_neg_RRP	min_temperature	\
0	0.040206	0.978844	0.033333	0.463504	
1	0.147992	0.860295	0.100000	0.540146	
2	0.000000	1.000000	0.000000	0.708029	

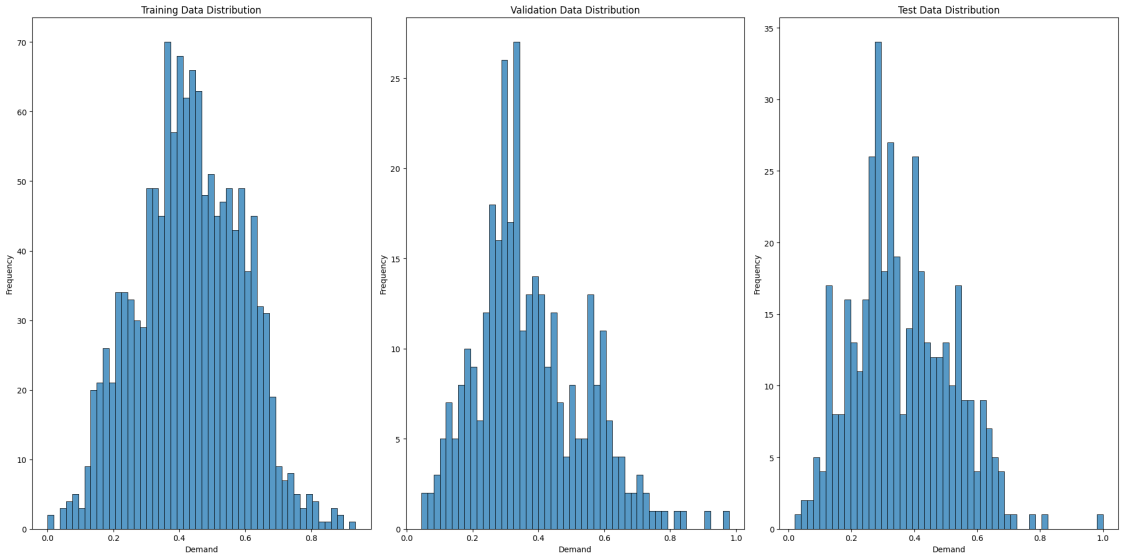
```
3           0.000000      1.000000          0.000000          0.572993
4           0.000000      1.000000          0.000000          0.525547
…                 …             …                 …                 …
2101        1.000000      0.911829          1.000000          0.445255
2102        0.835860      0.901999          0.933333          0.613139
2103        0.095500      0.999524          0.100000          0.470803
2104        0.121926      0.980973          0.133333          0.310219
2105        0.000000      1.000000          0.000000          0.302920

      max_temperature  solar_exposure  rainfall  school_day  holiday  day  \
0            0.518841        0.702454  0.000000           0        1    1
1            0.863768        0.800613  0.000000           0        0    2
2            0.846377        0.791411  0.000000           0        0    3
3            0.359420        0.751534  0.076923           0        0    4
4            0.376812        0.920245  0.000000           0        0    5
…                 …               …         …           …        …  …
2101         0.492754        0.653374  0.000000           0        0    2
2102         0.591304        0.585890  0.000000           0        0    3
2103         0.594203        0.236196  0.000000           0        0    4
2104         0.107246        0.202454  0.234432           0        0    5
2105         0.104348        0.156442  0.018315           0        0    6

      month  year day_of_week
0         1  2015    Thursday
1         1  2015      Friday
2         1  2015    Saturday
3         1  2015      Sunday
4         1  2015      Monday
…     …   …          …
2101     10  2020      Friday
2102     10  2020    Saturday
2103     10  2020      Sunday
2104     10  2020      Monday
2105     10  2020     Tuesday

[2106 rows x 18 columns]
```

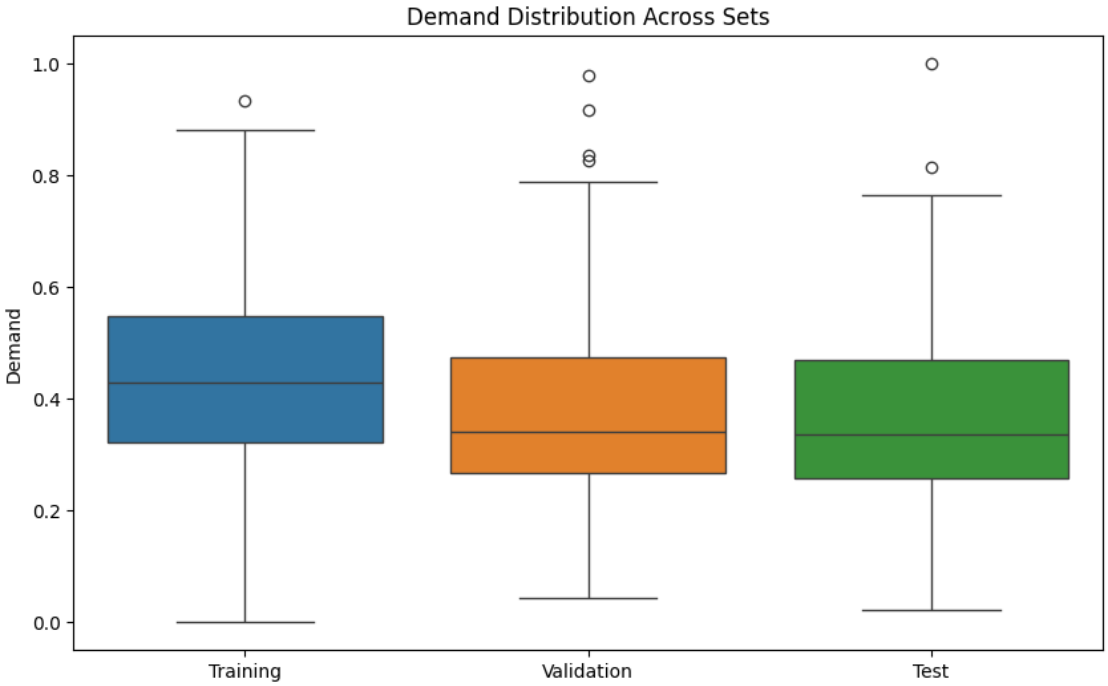
[ ]:
```
#Splitting the dataset into training,validation,and testing dataset
import pandas as pd
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import seaborn as sns
```

```
pd.set_option('future.no_silent_downcasting', True)

features = [
    'RRP', 'demand_pos_RRP', 'RRP_positive', 'demand_neg_RRP',
    'RRP_negative', 'frac_at_neg_RRP', 'min_temperature',
    'max_temperature', 'solar_exposure', 'rainfall',
    'school_day', 'holiday', 'day', 'month', 'year', 'day_of_week'
]

target = 'demand'

X = df[features].copy()
y = df[target].copy()
X = X.fillna(0)
X = pd.get_dummies(X, drop_first=True)
# 3. Data Splitting
train_size = int(len(df) * 0.8)
val_size = int(train_size * 0.2)
X_train = X[:train_size-val_size].copy()
X_val = X[train_size-val_size:train_size].copy()
X_test = X[train_size:].copy()

y_train = y[:train_size-val_size].copy()
y_val = y[train_size-val_size:train_size].copy()
y_test = y[train_size:].copy()

scaler = StandardScaler()
scaler.fit(X_train)

imputer = SimpleImputer(strategy='mean')
X_train_scaled = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.
 ↪columns)
X_val_scaled = pd.DataFrame(imputer.transform(X_val), columns=X_val.columns)
X_test_scaled = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)

print("\nDataset Shapes:")
print(f"Training set   : {X_train_scaled.shape}")
print(f"Validation set : {X_val_scaled.shape}")
print(f"Test set       : {X_test_scaled.shape}")

# 6. Visualization of Data Distribution
plt.figure(figsize=(20, 10))
plt.subplot(1, 3, 1)
sns.histplot(y_train, bins=50)
plt.title('Training Data Distribution')
plt.xlabel('Demand')
plt.ylabel('Frequency')
```

```
plt.subplot(1, 3, 2)
sns.histplot(y_val, bins=50)
plt.title('Validation Data Distribution')
plt.xlabel('Demand')
plt.ylabel('Frequency')
plt.subplot(1, 3, 3)
sns.histplot(y_test, bins=50)
plt.title('Test Data Distribution')
plt.xlabel('Demand')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
plt.figure(figsize=(10, 6))
combined_data = pd.DataFrame({
    'Training': y_train,
    'Validation': y_val,
    'Test': y_test
})

sns.boxplot(data=combined_data)
plt.title('Demand Distribution Across Sets')
plt.ylabel('Demand')
plt.show()
print("\nSummary Statistics:")
print("\nTraining Set:")
print(y_train.describe())
print("\nValidation Set:")
print(y_val.describe())
print("\nTest Set:")
print(y_test.describe())
print("\nData Types of Features:")
print(X.dtypes)

print("\nData preprocessing and splitting completed successfully!")
```

```
Dataset Shapes:
Training set   : (1348, 21)
Validation set : (336, 21)
Test set       : (422, 21)
```






```
Summary Statistics:

Training Set:
count    1348.000000
mean        0.431626
```

```
std         0.157574
min         0.000000
25%         0.322182
50%         0.430283
75%         0.548017
max         0.934744
Name: demand, dtype: float64

Validation Set:
count    336.000000
mean       0.374641
std        0.163176
min        0.044515
25%        0.266407
50%        0.341604
75%        0.475261
max        0.979441
Name: demand, dtype: float64

Test Set:
count    422.000000
mean       0.361007
std        0.153550
min        0.021001
25%        0.258401
50%        0.337463
75%        0.469220
max        1.000000
Name: demand, dtype: float64

Data Types of Features:
RRP                      float64
demand_pos_RRP           float64
RRP_positive             float64
demand_neg_RRP           float64
RRP_negative             float64
frac_at_neg_RRP          float64
min_temperature          float64
max_temperature          float64
solar_exposure           float64
rainfall                 float64
school_day                 int64
holiday                    int64
day                        int32
month                      int32
year                       int32
day_of_week_Monday          bool
day_of_week_Saturday        bool
```

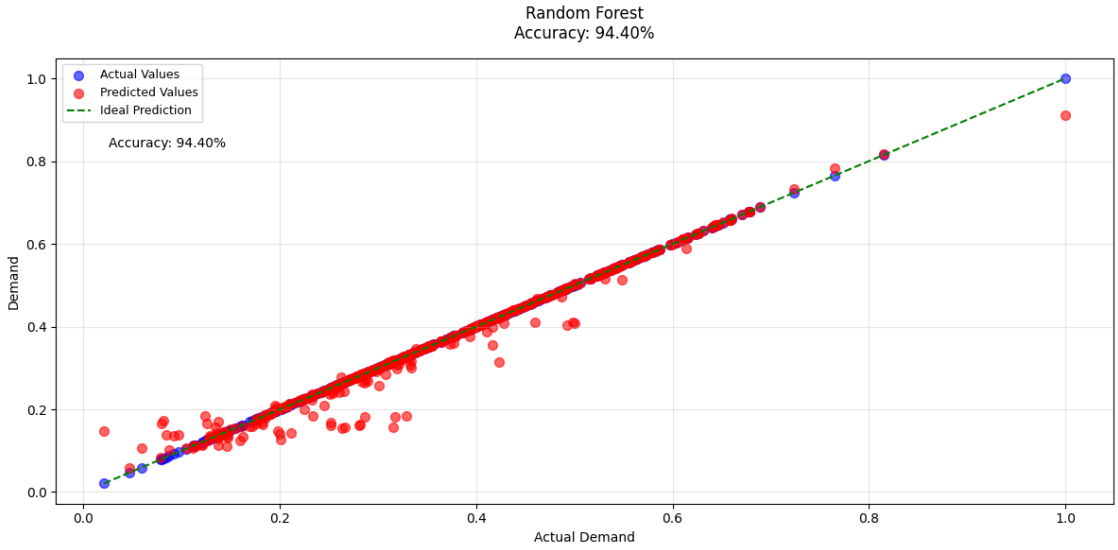
```
day_of_week_Sunday          bool
day_of_week_Thursday        bool
day_of_week_Tuesday         bool
day_of_week_Wednesday       bool
dtype: object

Data preprocessing and splitting completed successfully!
```

[ ]:
```python
# Random forest regression
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler

np.random.seed(42)
random.seed(42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

rf_model = RandomForestRegressor(
    n_estimators=100,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)

rf_model.fit(X_train_scaled, y_train)

rf_predictions = rf_model.predict(X_test_scaled)

rf_r2 = r2_score(y_test, rf_predictions)
rf_mae = mean_absolute_error(y_test, rf_predictions)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_predictions))
rf_mape = np.mean(np.abs((y_test - rf_predictions) / y_test)) * 100
rf_accuracy = 100 - rf_mape

print("Random Forest R-squared:", rf_r2)
print("Random Forest MAE:", rf_mae)
print("Random Forest RMSE:", rf_rmse)
print("Random Forest MAPE:", rf_mape)
print("Random Forest Accuracy:", rf_accuracy)
```

```
Random Forest R-squared: 0.9717205921752488
Random Forest MAE: 0.009243488141475421
Random Forest RMSE: 0.025791035532222922
```

```
Random Forest MAPE: 5.597700575007415
Random Forest Accuracy: 94.40229942499259
```

[ ]:
```python
#Plotting demand vs actual demand for Random Forest
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import r2_score, mean_absolute_error

rf_r2 = r2_score(y_test, rf_predictions) * 100
rf_accuracy = 100 - (np.mean(np.abs((y_test - rf_predictions) / y_test)) * 100)

plt.figure(figsize=(12, 6))

plt.subplot(1, 1, 1)
plt.scatter(y_test, y_test, color='blue', alpha=0.6, label='Actual Values',␣
 ↪s=50)
plt.scatter(y_test, rf_predictions, color='red', alpha=0.6, label='Predicted␣
 ↪Values', s=50)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
 ↪color='green', linestyle='--', label='Ideal Prediction')
plt.title(f'Random Forest\nAccuracy: {rf_accuracy:.2f}%', fontsize=12, pad=15)
plt.xlabel('Actual Demand', fontsize=10)
plt.ylabel('Demand', fontsize=10)
plt.legend(fontsize=9)
plt.grid(True, alpha=0.3)

plt.text(0.05, 0.80, f'Accuracy: {rf_accuracy:.2f}%', transform=plt.gca().
 ↪transAxes, bbox=dict(facecolor='white', alpha=0.8, edgecolor='none'),␣
 ↪fontsize=10)

plt.tight_layout()
plt.show()

print("\nDetailed Accuracy Metrics:")
print(f"Random Forest Accuracy: {rf_accuracy:.2f}%")
```

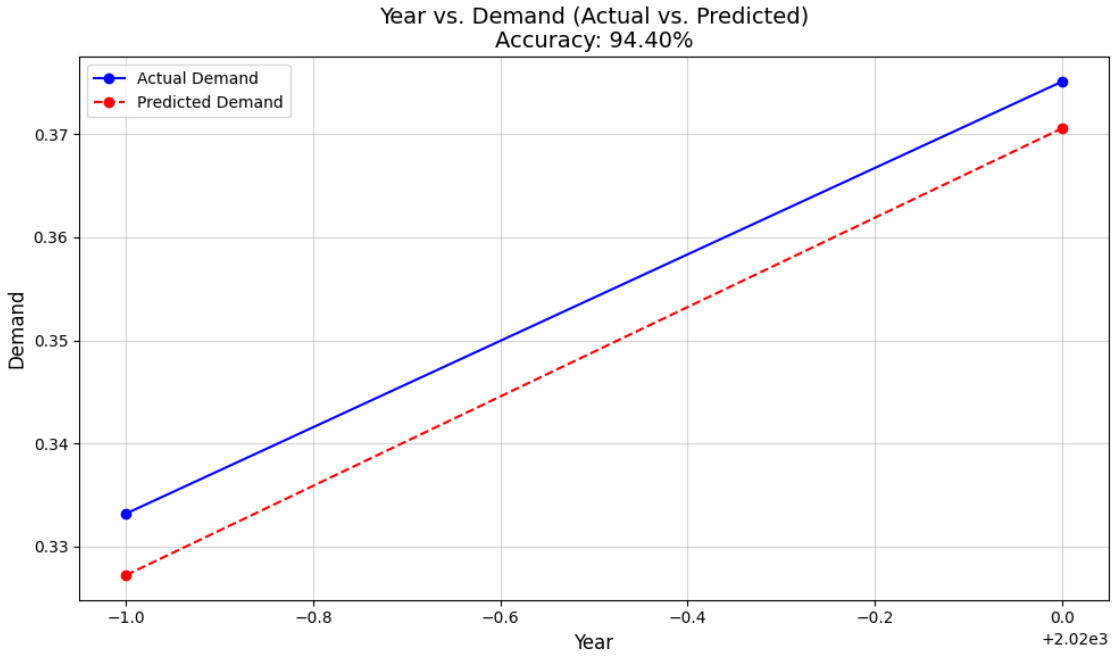



```
Detailed Accuracy Metrics:
Random Forest Accuracy: 94.40%
```

[ ]:
```python
# Plotting year vs demand for Random Forest
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

df_test = df.loc[y_test.index].copy()
df_test['predicted_demand'] = rf_predictions
yearly_actual_demand = df_test.groupby('year')['demand'].mean()
yearly_predicted_demand = df_test.groupby('year')['predicted_demand'].mean()

accuracy = 100 - (np.mean(np.abs((y_test - rf_predictions) / y_test)) * 100)
r2 = r2_score(y_test, rf_predictions) * 100
mae = mean_absolute_error(y_test, rf_predictions)
mape = np.mean(np.abs((y_test - rf_predictions) / y_test)) * 100
rmse = np.sqrt(mean_squared_error(y_test, rf_predictions))

print("Accuracy:", accuracy, "%")
print("R^2 Score:", r2, "%")
print("MAE:", mae)
print("MAPE:", mape, "%")
print("RMSE:", rmse)

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

```
plt.plot(yearly_actual_demand.index, yearly_actual_demand.values, marker='o',␣
 ↪linestyle='-', color='blue', label='Actual Demand')

plt.plot(yearly_predicted_demand.index, yearly_predicted_demand.values,␣
 ↪marker='o', linestyle='--', color='red', label='Predicted Demand')

plt.title('Year vs. Demand (Actual vs. Predicted)\nAccuracy: {:.2f}%'.
 ↪format(accuracy, r2), fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Demand', fontsize=12)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.5)
plt.tight_layout()

plt.show()
```

```
Accuracy: 94.40229942499259 %
R^2 Score: 97.17205921752488 %
MAE: 0.009243488141475421
MAPE: 5.597700575007415 %
RMSE: 0.025791035532222922
```



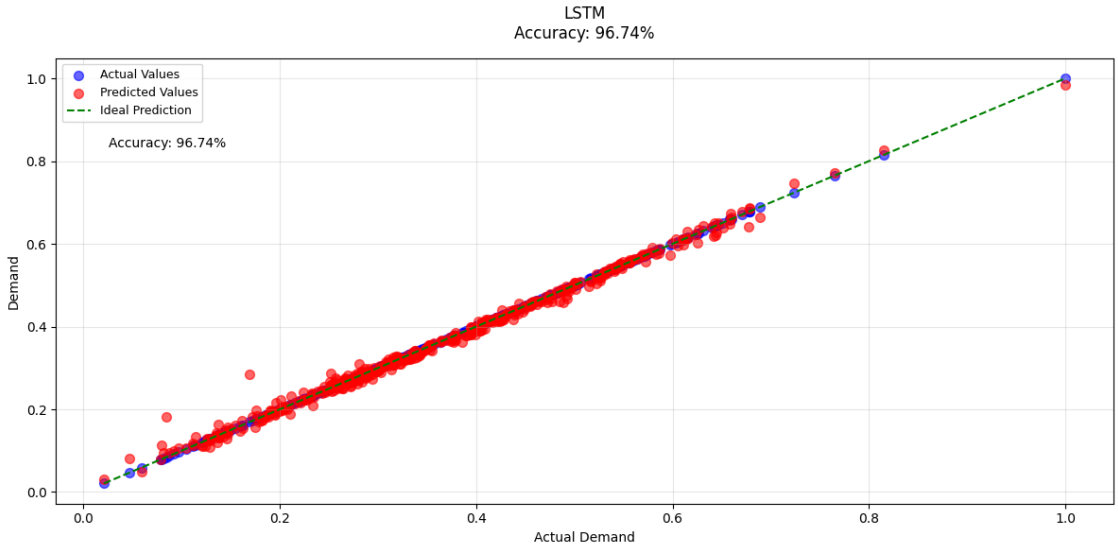
[ ]:
```
#Fitting LSTM model
import numpy as np
import tensorflow as tf
```

```
import random
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input
from sklearn.preprocessing import MinMaxScaler
import os

np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
os.environ['TF_DETERMINISTIC_OPS'] = '1'
X_train_array = np.array(X_train)
X_test_array = np.array(X_test)
y_train_array = np.array(y_train)
y_test_array = np.array(y_test)

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train_array)
X_test_scaled = scaler.transform(X_test_array)

X_train_reshaped = X_train_scaled.reshape((X_train_scaled.shape[0], 1,␣
 ↪X_train_scaled.shape[1]))
X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], 1,␣
 ↪X_test_scaled.shape[1]))
model = Sequential()
model.add(Input(shape=(1, X_train_scaled.shape[1])))
model.add(LSTM(50, activation='relu', kernel_initializer='glorot_uniform',␣
 ↪recurrent_initializer='glorot_uniform'))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

model.fit(X_train_reshaped, y_train_array, epochs=50, batch_size=32,␣
 ↪validation_split=0.1, verbose=0)

lstm_predictions = model.predict(X_test_reshaped, verbose=0).flatten()

lstm_r2 = r2_score(y_test_array, lstm_predictions)
lstm_mae = mean_absolute_error(y_test_array, lstm_predictions)
lstm_rmse = np.sqrt(mean_squared_error(y_test_array, lstm_predictions))
lstm_mape = np.mean(np.abs((y_test_array - lstm_predictions) / y_test_array)) *␣
 ↪100
lstm_accuracy = 100 - lstm_mape

print("LSTM R-squared:", lstm_r2)
print("LSTM MAE:", lstm_mae)
print("LSTM RMSE:", lstm_rmse)
```

```
print("LSTM MAPE:", lstm_mape)
print("LSTM Accuracy:", lstm_accuracy)
```

```
LSTM R-squared: 0.9936641363756519
LSTM MAE: 0.007798684427441561
LSTM RMSE: 0.01220777551700799
LSTM MAPE: 3.258611007363971
LSTM Accuracy: 96.74138899263603
```

[ ]:
```
#Plotting demand vs actual demand for LSTM
import matplotlib.pyplot as plt
import numpy as np

from sklearn.metrics import r2_score, mean_absolute_error

lstm_r2 = r2_score(y_test, lstm_predictions) * 100
lstm_accuracy = 100 - (np.mean(np.abs((y_test - lstm_predictions) / y_test)) *␣
 ↪100)

plt.figure(figsize=(12, 6))
plt.subplot(1, 1, 1)
plt.scatter(y_test, y_test, color='blue', alpha=0.6, label='Actual Values',␣
 ↪s=50)
plt.scatter(y_test, lstm_predictions, color='red', alpha=0.6, label='Predicted␣
 ↪Values', s=50)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
 ↪color='green', linestyle='--', label='Ideal Prediction')
plt.title(f'LSTM\nAccuracy: {lstm_accuracy:.2f}%', fontsize=12, pad=15)
plt.xlabel('Actual Demand', fontsize=10)
plt.ylabel('Demand', fontsize=10)
plt.legend(fontsize=9)
plt.grid(True, alpha=0.3)
plt.text(0.05, 0.80, f'Accuracy: {lstm_accuracy:.2f}%', transform=plt.gca().
 ↪transAxes, bbox=dict(facecolor='white', alpha=0.8, edgecolor='none'),␣
 ↪fontsize=10)
plt.tight_layout()
plt.show()
print("\nDetailed Accuracy Metrics:")
print(f"LSTM Accuracy: {lstm_accuracy:.2f}%")
```

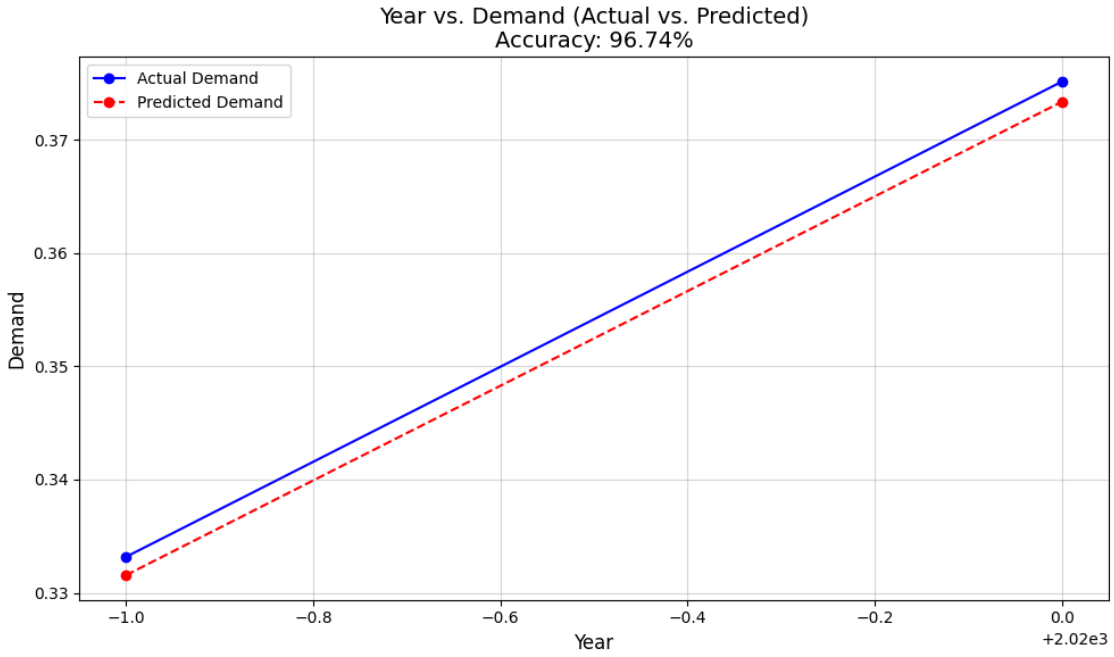



```
Detailed Accuracy Metrics:
LSTM Accuracy: 96.74%
```

[ ]:
```
# Plotting year vs demand for LSTM
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
df_test = df.loc[y_test.index].copy()
df_test['predicted_demand'] = lstm_predictions
yearly_actual_demand = df_test.groupby('year')['demand'].mean()
yearly_predicted_demand = df_test.groupby('year')['predicted_demand'].mean()
accuracy = 100 - (np.mean(np.abs((y_test - lstm_predictions) / y_test)) * 100)
r2 = r2_score(y_test, lstm_predictions) * 100
mae = mean_absolute_error(y_test, lstm_predictions)
mape = np.mean(np.abs((y_test - lstm_predictions) / y_test)) * 100
rmse = np.sqrt(mean_squared_error(y_test, lstm_predictions))
print("Accuracy:", accuracy, "%")
print("R^2 Score:", r2, "%")
print("MAE:", mae)
print("MAPE:", mape, "%")
print("RMSE:", rmse)
plt.figure(figsize=(10, 6))
plt.plot(yearly_actual_demand.index, yearly_actual_demand.values, marker='o',␣
 ↪linestyle='-', color='blue', label='Actual Demand')
plt.plot(yearly_predicted_demand.index, yearly_predicted_demand.values,␣
 ↪marker='o', linestyle='--', color='red', label='Predicted Demand')
plt.title('Year vs. Demand (Actual vs. Predicted)\nAccuracy: {:.2f}%'.
 ↪format(accuracy, r2), fontsize=14)
```

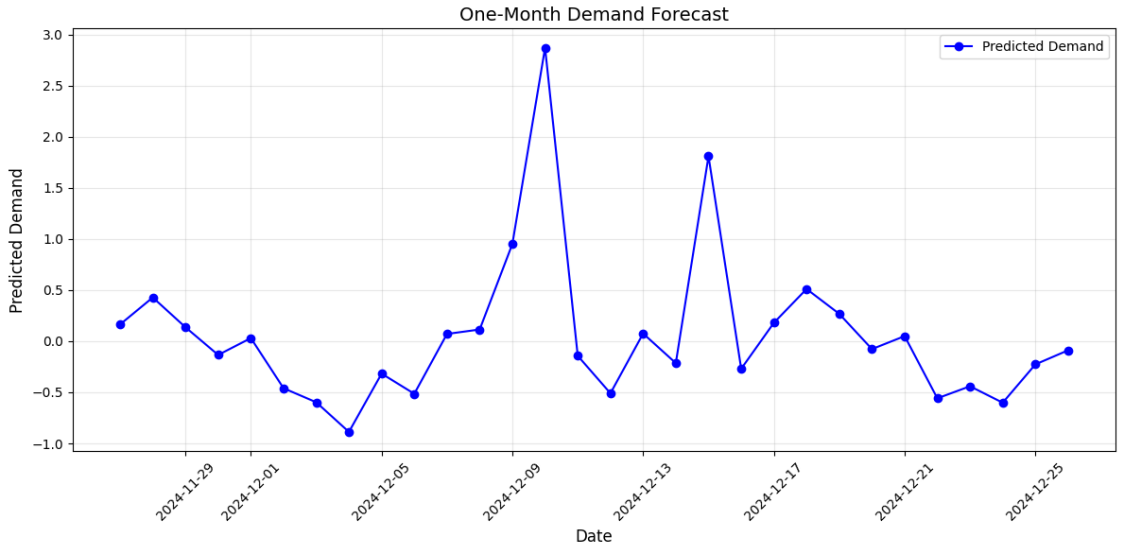
```
plt.xlabel('Year', fontsize=12)
plt.ylabel('Demand', fontsize=12)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.5)
plt.tight_layout()
plt.show()
```

```
Accuracy: 96.74138899263603 %
R^2 Score: 99.36641363756519 %
MAE: 0.007798684427441561
MAPE: 3.258611007363971 %
RMSE: 0.01220777551700799
```



[ ]:
```
#Predicting One month demand
import numpy as np
forecast_days = 30
input_data = X_test_scaled[-1].reshape((1, 1, X_test_scaled.shape[1]))
future_predictions = []

for _ in range(forecast_days):
    next_pred = model.predict(input_data, verbose=0).flatten()[0]
    future_predictions.append(next_pred)
    next_input = np.append(input_data[:, :, 1:], next_pred).reshape((1, 1,␣
 ↪X_test_scaled.shape[1]))
    input_data = next_input
```

```
future_predictions_unscaled = future_predictions
import pandas as pd
start_date = pd.to_datetime('today').normalize()
future_dates = [start_date + pd.Timedelta(days=i) for i in range(forecast_days)]
forecast_df = pd.DataFrame({'Date': future_dates, 'Predicted Demand':␣
 ↪future_predictions_unscaled})
plt.figure(figsize=(12, 6))
plt.plot(forecast_df['Date'], forecast_df['Predicted Demand'], marker='o',␣
 ↪label='Predicted Demand', color='blue')
plt.title('One-Month Demand Forecast', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Predicted Demand', fontsize=12)
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.legend(fontsize=10)
plt.tight_layout()
plt.show()
print(forecast_df)
```



	Date	Predicted Demand
0	2024-11-27	0.163668
1	2024-11-28	0.427155
2	2024-11-29	0.135978
3	2024-11-30	-0.134831
4	2024-12-01	0.029899
5	2024-12-02	-0.460260
6	2024-12-03	-0.597640
7	2024-12-04	-0.887626
8	2024-12-05	-0.317628

9	2024-12-06	-0.514772
10	2024-12-07	0.070582
11	2024-12-08	0.113921
12	2024-12-09	0.950186
13	2024-12-10	2.869552
14	2024-12-11	-0.143442
15	2024-12-12	-0.509984
16	2024-12-13	0.074615
17	2024-12-14	-0.214810
18	2024-12-15	1.809117
19	2024-12-16	-0.270221
20	2024-12-17	0.179522
21	2024-12-18	0.510049
22	2024-12-19	0.266998
23	2024-12-20	-0.077648
24	2024-12-21	0.049431
25	2024-12-22	-0.558887
26	2024-12-23	-0.441116
27	2024-12-24	-0.601957
28	2024-12-25	-0.227032
29	2024-12-26	-0.089490