In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```python
df=pd.read_csv("PDB_Load_History.csv")
df.head()
```

Out[2]:

	date	year	month	day	weekday	hour	demand	temperature
0	3/1/2003	2003	3	1	7	1	12863.0	29
1	3/1/2003	2003	3	1	7	2	12389.0	28
2	3/1/2003	2003	3	1	7	3	12155.0	29
3	3/1/2003	2003	3	1	7	4	12072.0	27
4	3/1/2003	2003	3	1	7	5	12160.0	29

In [3]:
```python
df.describe()
```

Out[3]:

	year	month	day	weekday	hour	demand	temperature
count	103776.000000	103776.000000	103776.000000	103776.000000	103776.00000	103776.000000	103776.000000
mean	2008.574699	6.591813	15.735661	3.999306	12.50000	14674.947493	50.576097
std	3.414726	3.420534	8.802989	2.000183	6.92222	2894.544130	18.300454
min	2003.000000	1.000000	1.000000	1.000000	1.00000	7794.000000	-7.000000
25%	2006.000000	4.000000	8.000000	2.000000	6.75000	12514.000000	36.000000
50%	2009.000000	7.000000	16.000000	4.000000	12.50000	14773.000000	51.000000
75%	2012.000000	10.000000	23.000000	6.000000	18.25000	16443.000000	65.000000
max	2014.000000	12.000000	31.000000	7.000000	24.00000	27622.000000	100.000000

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

Out[4]: (103776, 8)

In [5]:
```python
# Summarize the number of null values in each column
null_counts = df.isnull().sum()

# Display the counts of null values per column
print(null_counts)
```

```
date           0
year           0
month          0
day            0
weekday        0
hour           0
demand         0
temperature    0
dtype: int64
```

In [6]:
```python
#uniquely identifies the no of years in the dataset
df["year"].unique()
```

Out[6]:
```
array([2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013,
       2014])
```

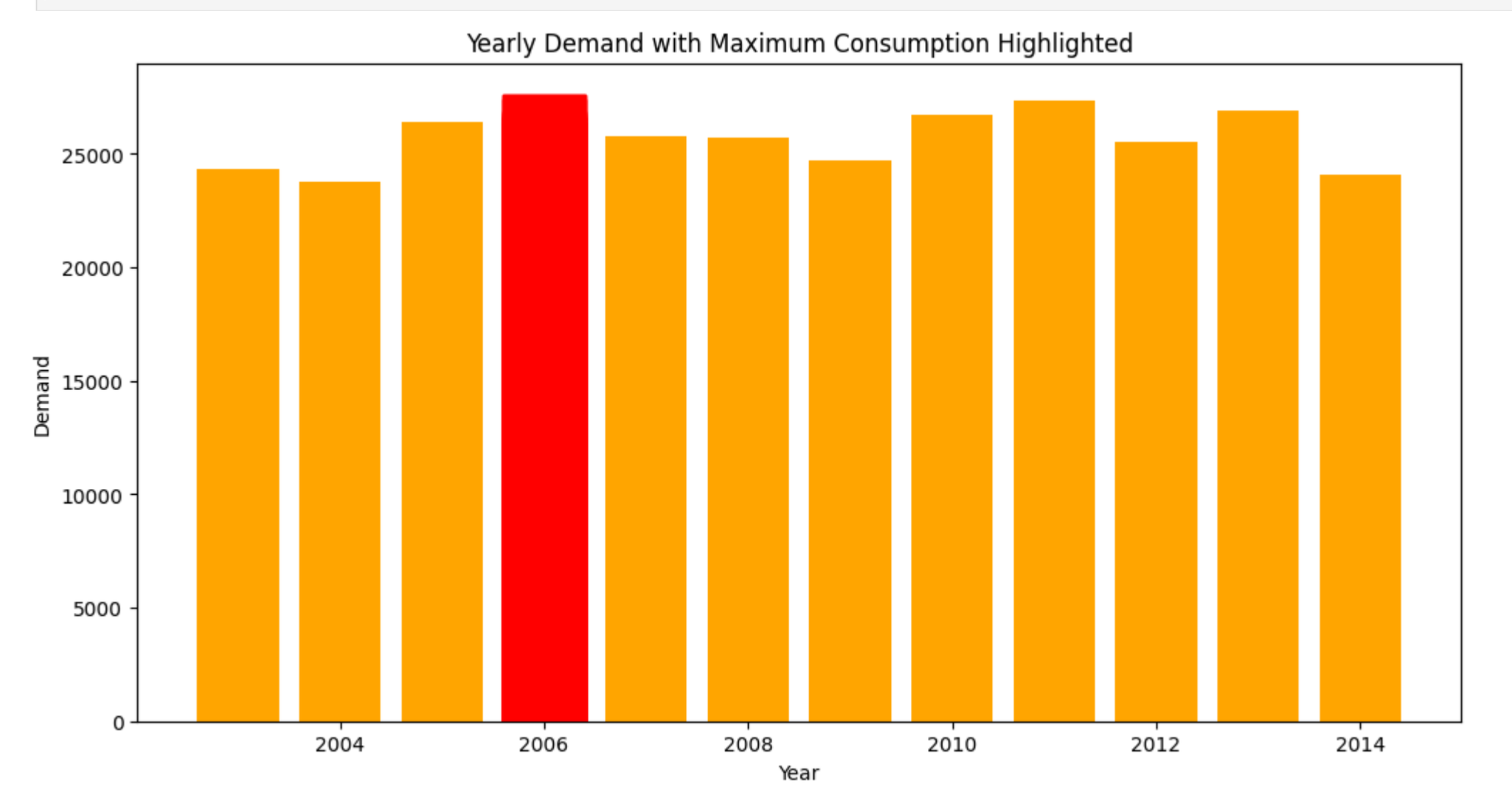
In [7]:
```python
#counting the total no of unqiue years
df["year"].nunique()
```

Out[7]: 12

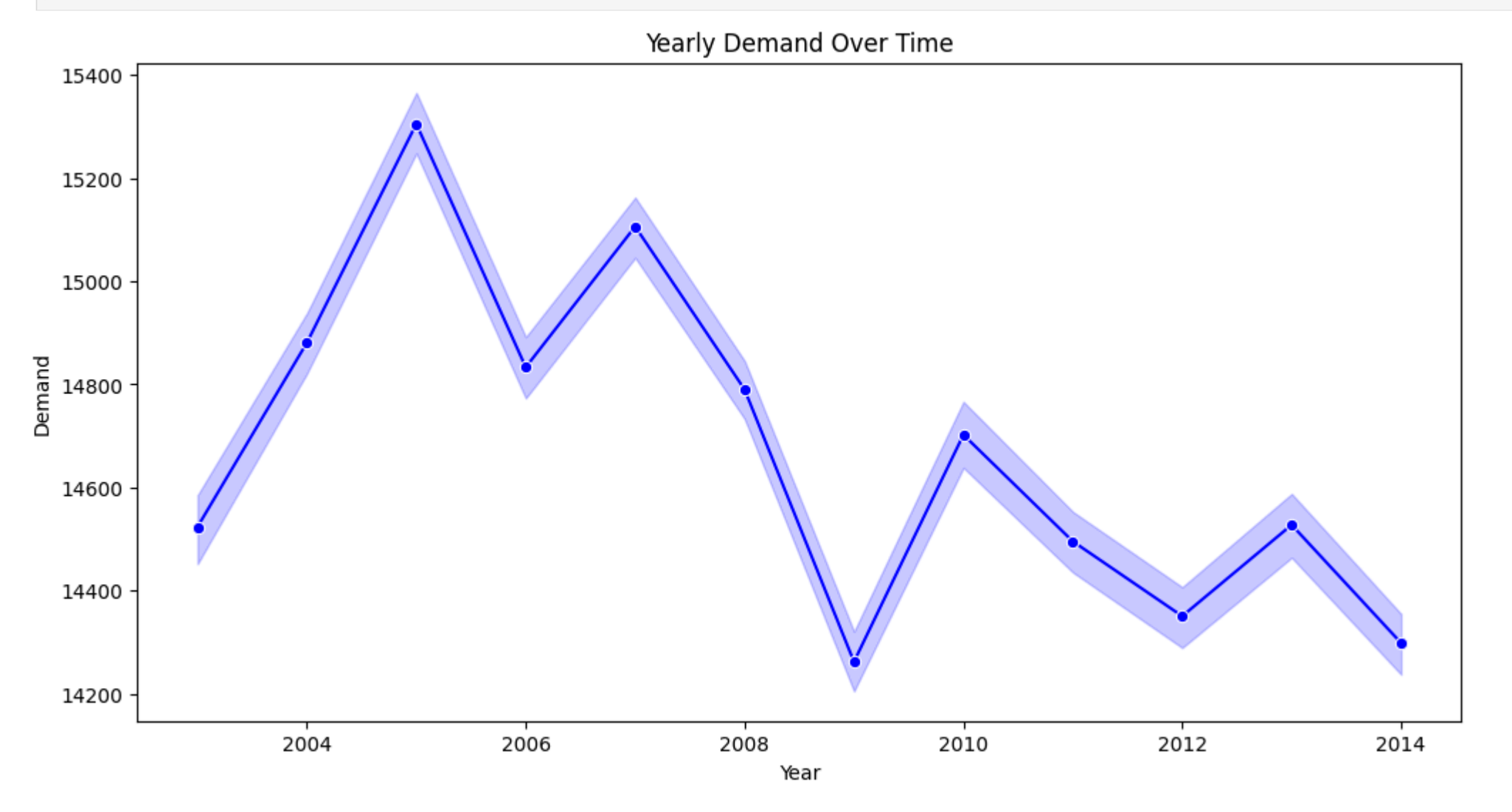
In [8]:
```python
#maximum consumption of electrcity in different years
max_consumption_2003 = df[df["year"] == 2003]["demand"].max()
max_consumption_2004 = df[df["year"] == 2004]["demand"].max()
max_consumption_2005 = df[df["year"] == 2005]["demand"].max()
max_consumption_2006 = df[df["year"] == 2006]["demand"].max()
max_consumption_2007 = df[df["year"] == 2007]["demand"].max()
max_consumption_2008 = df[df["year"] == 2008]["demand"].max()
max_consumption_2009 = df[df["year"] == 2009]["demand"].max()
max_consumption_2010 = df[df["year"] == 2010]["demand"].max()
max_consumption_2011 = df[df["year"] == 2011]["demand"].max()
max_consumption_2012 = df[df["year"] == 2012]["demand"].max()
max_consumption_2013 = df[df["year"] == 2013]["demand"].max()
max_consumption_2014 = df[df["year"] == 2014]["demand"].max()

print("Maximum electricity consumption in 2003:", max_consumption_2003)
print("Maximum electricity consumption in 2004:", max_consumption_2004)
print("Maximum electricity consumption in 2005:", max_consumption_2005)
print("Maximum electricity consumption in 2006:", max_consumption_2006)
print("Maximum electricity consumption in 2007:", max_consumption_2007)
print("Maximum electricity consumption in 2008:", max_consumption_2008)
print("Maximum electricity consumption in 2009:", max_consumption_2009)
print("Maximum electricity consumption in 2010:", max_consumption_2010)
print("Maximum electricity consumption in 2011:", max_consumption_2011)
print("Maximum electricity consumption in 2012:", max_consumption_2012)
print("Maximum electricity consumption in 2013:", max_consumption_2013)
print("Maximum electricity consumption in 2014:", max_consumption_2014)
```

```
Maximum electricity consumption in 2003: 24330.0
Maximum electricity consumption in 2004: 23750.0
Maximum electricity consumption in 2005: 26416.0
Maximum electricity consumption in 2006: 27622.0
Maximum electricity consumption in 2007: 25785.0
Maximum electricity consumption in 2008: 25696.0
Maximum electricity consumption in 2009: 24723.0
Maximum electricity consumption in 2010: 26705.0
Maximum electricity consumption in 2011: 27333.0
Maximum electricity consumption in 2012: 25553.0
Maximum electricity consumption in 2013: 26919.0
Maximum electricity consumption in 2014: 24089.0
```

In [9]:
```python
#This code snippet performs two main tasks: identifying the year with the maximum demand from a DataFrame df and then creating a bar plot to visualize the yearly demand, with the year of maximum demand highlighted in red.
max_demand = df['demand'].max()
max_year = df[df['demand'] == max_demand]['year'].values[0]

# Plotting
plt.figure(figsize=(12, 6))
bars = plt.bar(df['year'], df['demand'], color='orange')

#Highlight the bar with the maximum demand
for bar in bars:
    if bar.get_x() == max_year - 0.4:
        bar.set_color('red')

plt.xlabel('Year')
plt.ylabel('Demand')
plt.title('Yearly Demand with Maximum Consumption Highlighted')
plt.show()
```



In [23]:
```python
plt.figure(figsize=(12, 6))
sns.lineplot(x=df["year"], y=df["demand"], data=df, marker='o', linestyle='-', color='blue')

# Adding labels and title
plt.xlabel('Year')
plt.ylabel('Demand')
plt.title('Yearly Demand Over Time')
plt.show()
```

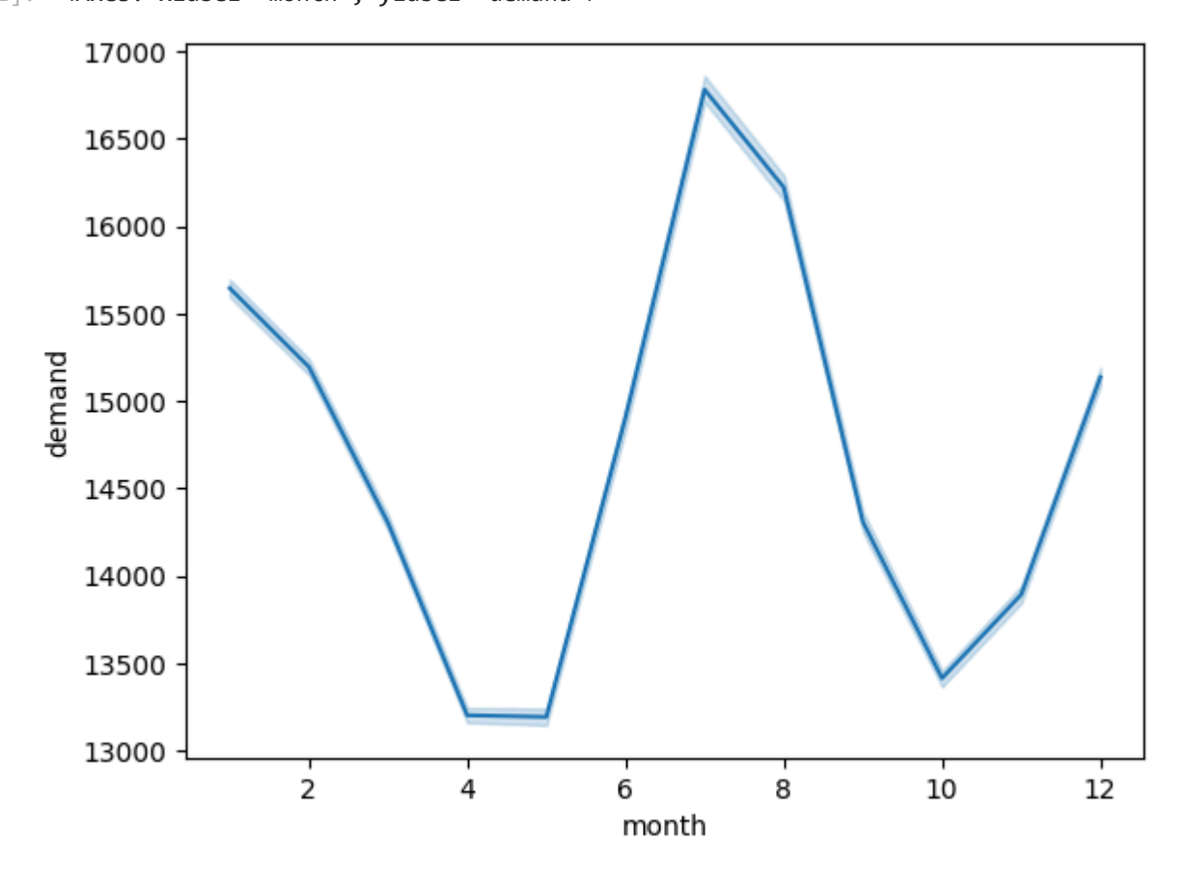


In [11]:
```python
mean_demand_2005 = df[df["year"] == 2005]["demand"].mean()

print("Mean demand in 2005:", mean_demand_2005)
```

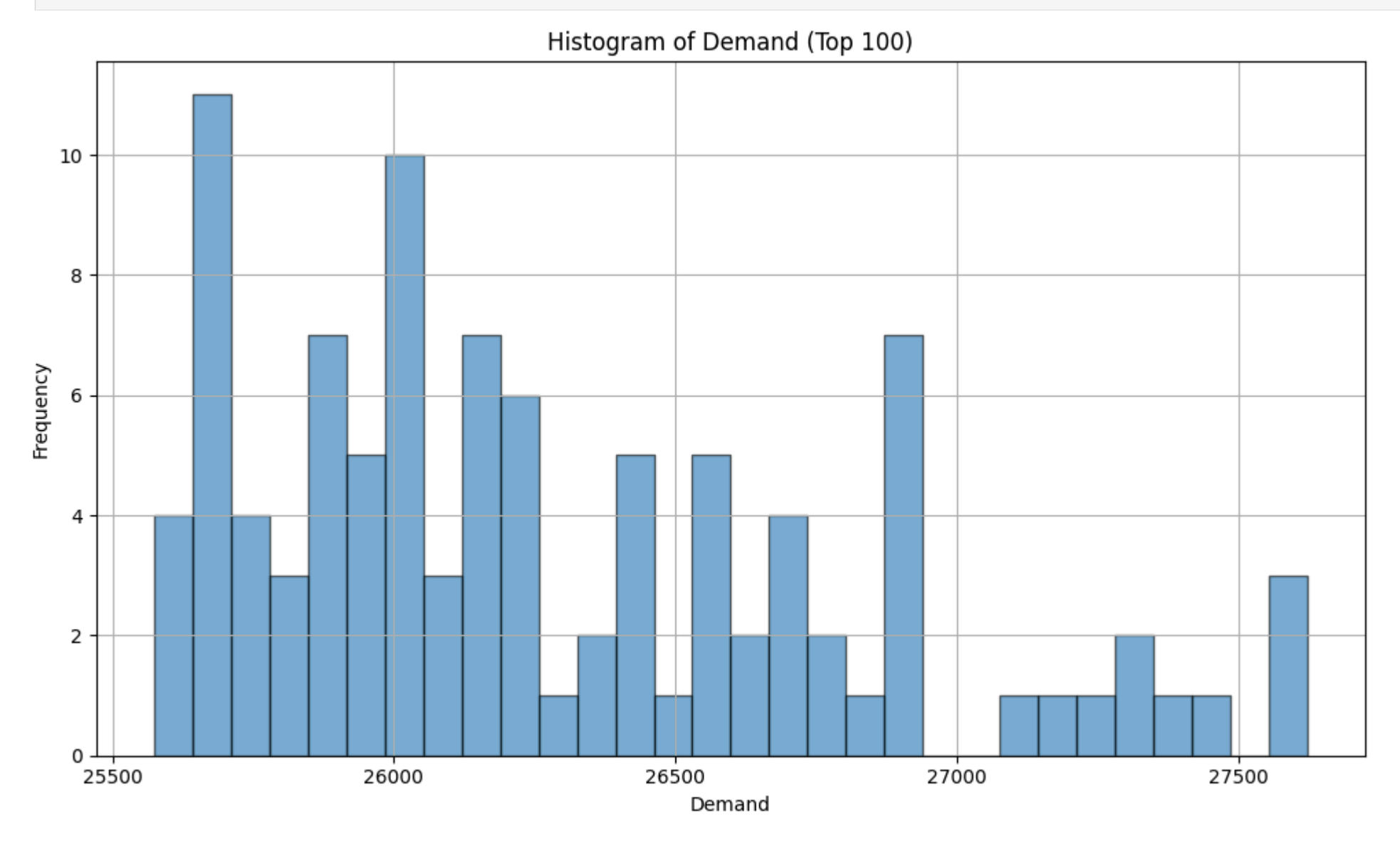
```
Mean demand in 2005: 15305.089726027398
```

In [12]:
```python
#relationship between demand and month
sns.lineplot(x=df["month"],y=df["demand"],data=df)
```

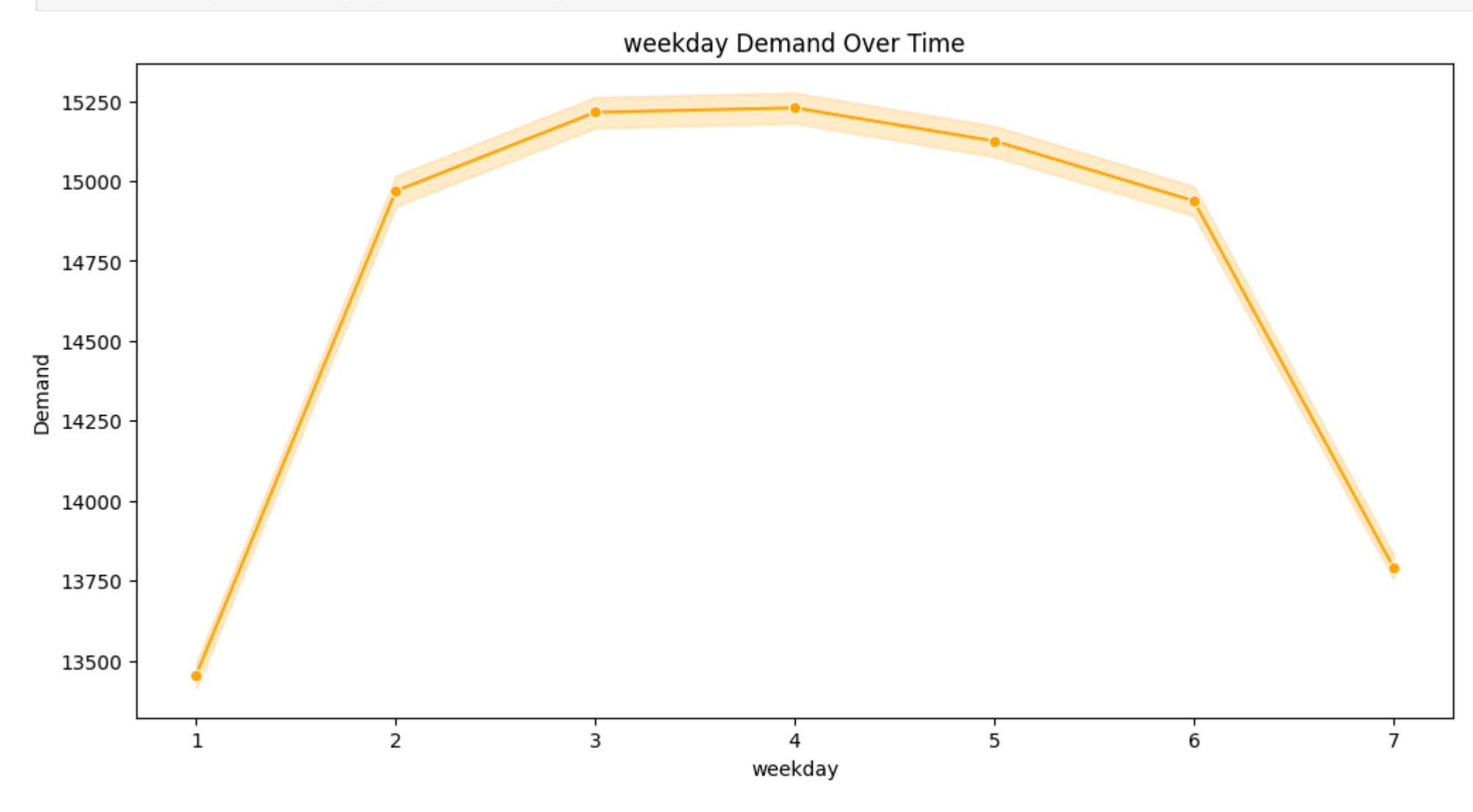
Out[12]: <Axes: xlabel='month', ylabel='demand'>



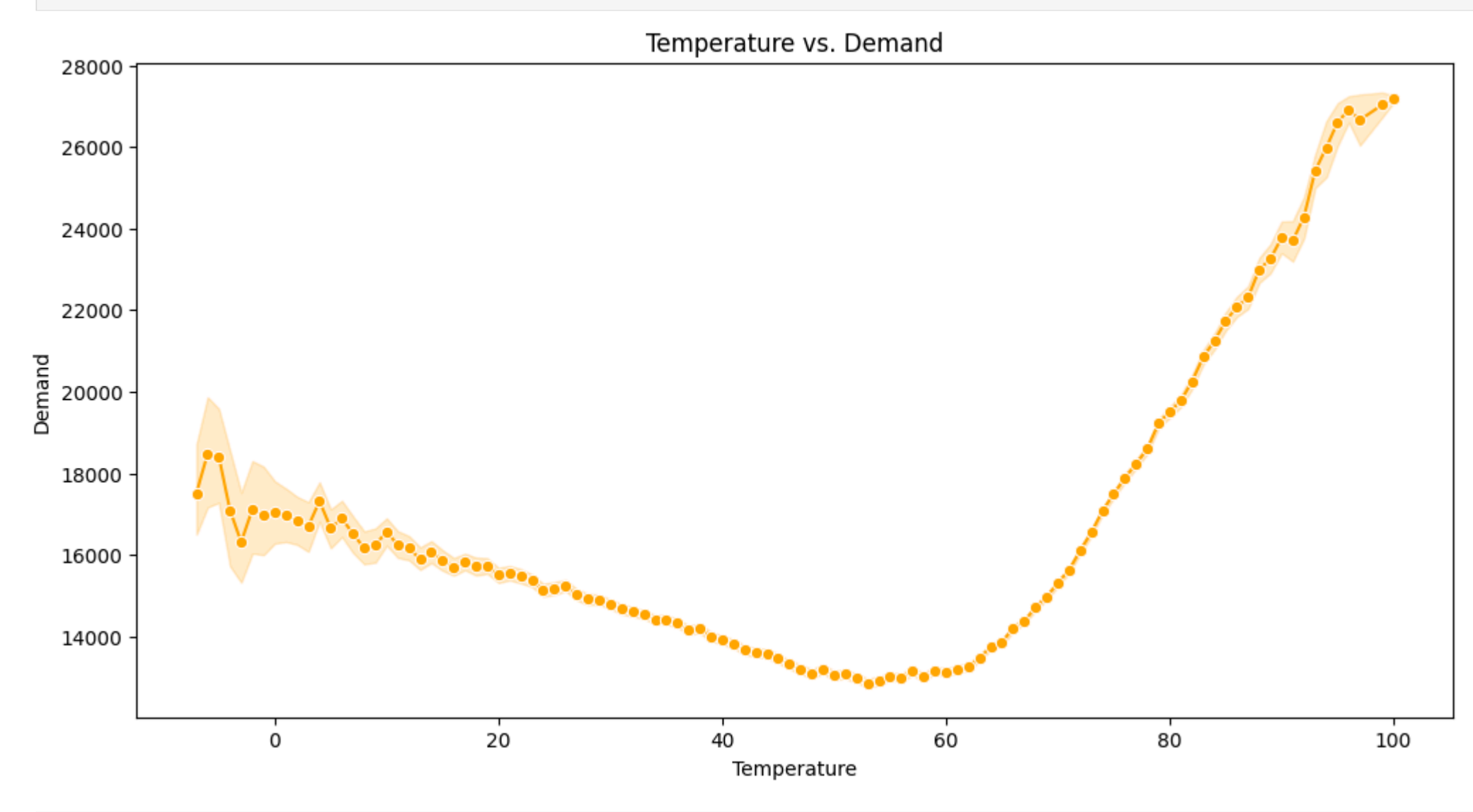
In [13]:
```python
top_100 = df.nlargest(100, 'demand')

plt.figure(figsize=(10, 6))
plt.hist(top_100['demand'], bins=30, edgecolor='black', alpha=0.6)

plt.title('Histogram of Demand (Top 100)')
plt.xlabel('Demand')
plt.ylabel('Frequency')
plt.grid(True)
plt.tight_layout()
plt.show()
```



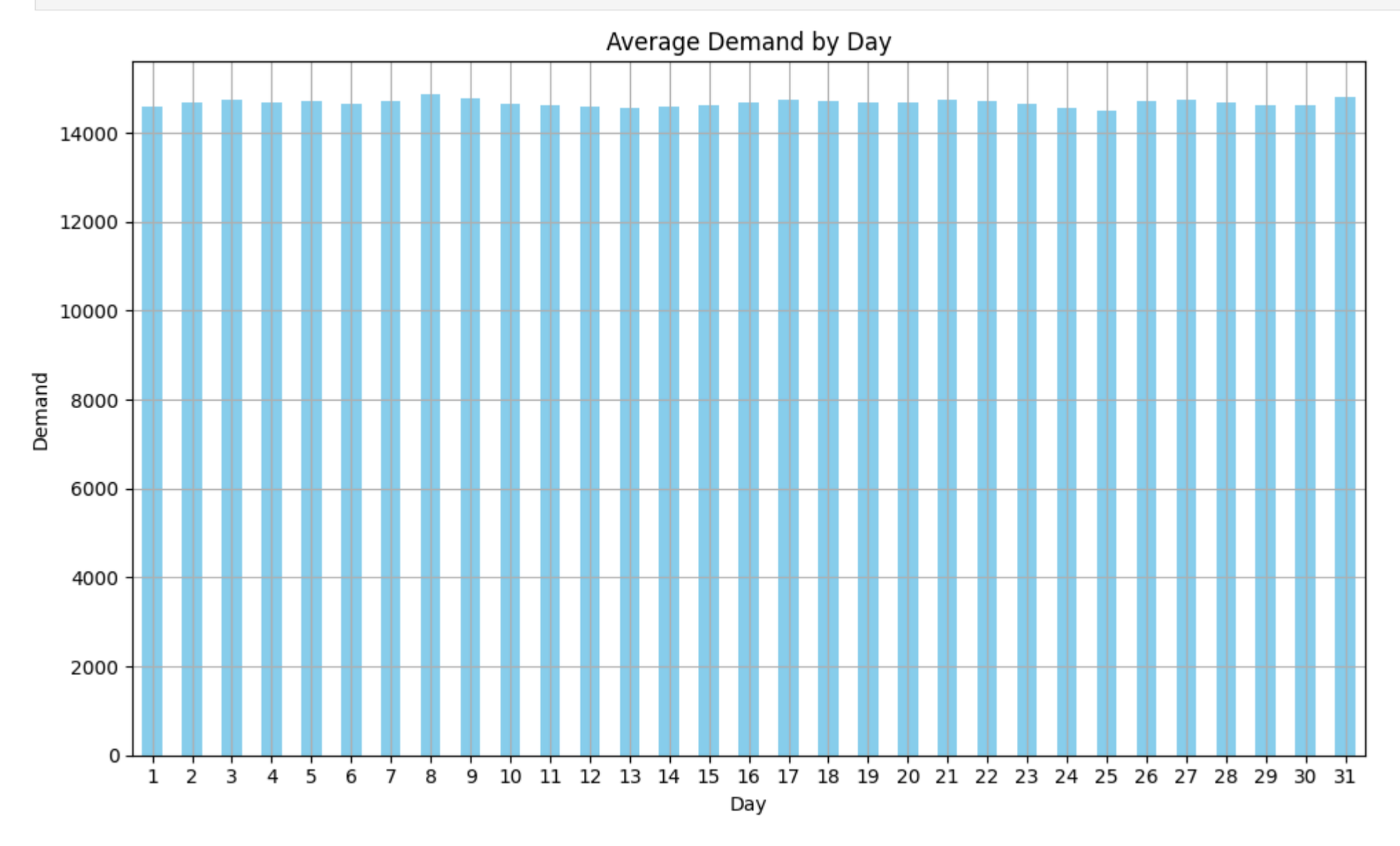
In [24]:
```python
plt.figure(figsize=(12, 6))
sns.lineplot(x=df["weekday"], y=df["demand"], data=df, marker='o', linestyle='-', color='orange')

# Adding labels and title
plt.xlabel('weekday')
plt.ylabel('Demand')
plt.title('weekday Demand Over Time')
plt.show()
#sns.lineplot(x=df["weekday"],y=df["demand"],data=df)
```



In [29]:
```python
plt.figure(figsize=(12, 6))
sns.lineplot(x=df["temperature"], y=df["demand"], data=df, marker='o', linestyle='-', color='orange')

# Adding labels and title
plt.xlabel('Temperature')
plt.ylabel('Demand')
plt.title('Temperature vs. Demand')
plt.show()
```



In [32]:
```python
mean_demand_by_day = df.groupby('day')['demand'].mean()

# Plotting the bar plot
plt.figure(figsize=(10, 6))
mean_demand_by_day.plot(kind='bar', color='skyblue')
plt.title('Average Demand by Day')
plt.xlabel('Day')
plt.ylabel('Demand')
plt.xticks(rotation=0)  # Rotate x-axis labels if needed
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [15]:
```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import pandas as pd
import numpy as np

# Define features and target
FEATURES = ['day', 'hour', 'weekday', 'temperature', 'month', 'year']
TARGET = 'demand'

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df[FEATURES], df[TARGET], test_size=0.2, random_state=42)

# Initialize Random Forest Regressor
reg = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=0)

# Fit the model
reg.fit(X_train, y_train)

# Predict on the test set
y_pred = reg.predict(X_test)

# Evaluate the model
r2 = reg.score(X_test, y_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)

print(f"R^2 Score: {r2}")
print(f"RMSE: {rmse}")
print(f"Mean Absolute Error: {mae}")

# Print feature importances
if hasattr(reg, 'feature_importances_'):
    fi = pd.DataFrame(data=reg.feature_importances_, index=X_train.columns, columns=['importance'])
    fi_sorted = fi.sort_values(by='importance', ascending=False)
    print("\nFeature Importances:")
    #print(fi_sorted)

# Optional: Store feature importances in a DataFrame
df1 = pd.DataFrame(fi_sorted)
df1
```

```
R^2 Score: 0.9323197056425113
RMSE: 755.8857107984361
Mean Absolute Error: 551.2352830332908

Feature Importances:
```

Out[15]:

	importance
hour	0.566942
temperature	0.335678
weekday	0.071012
month	0.016065
year	0.008540
day	0.001762

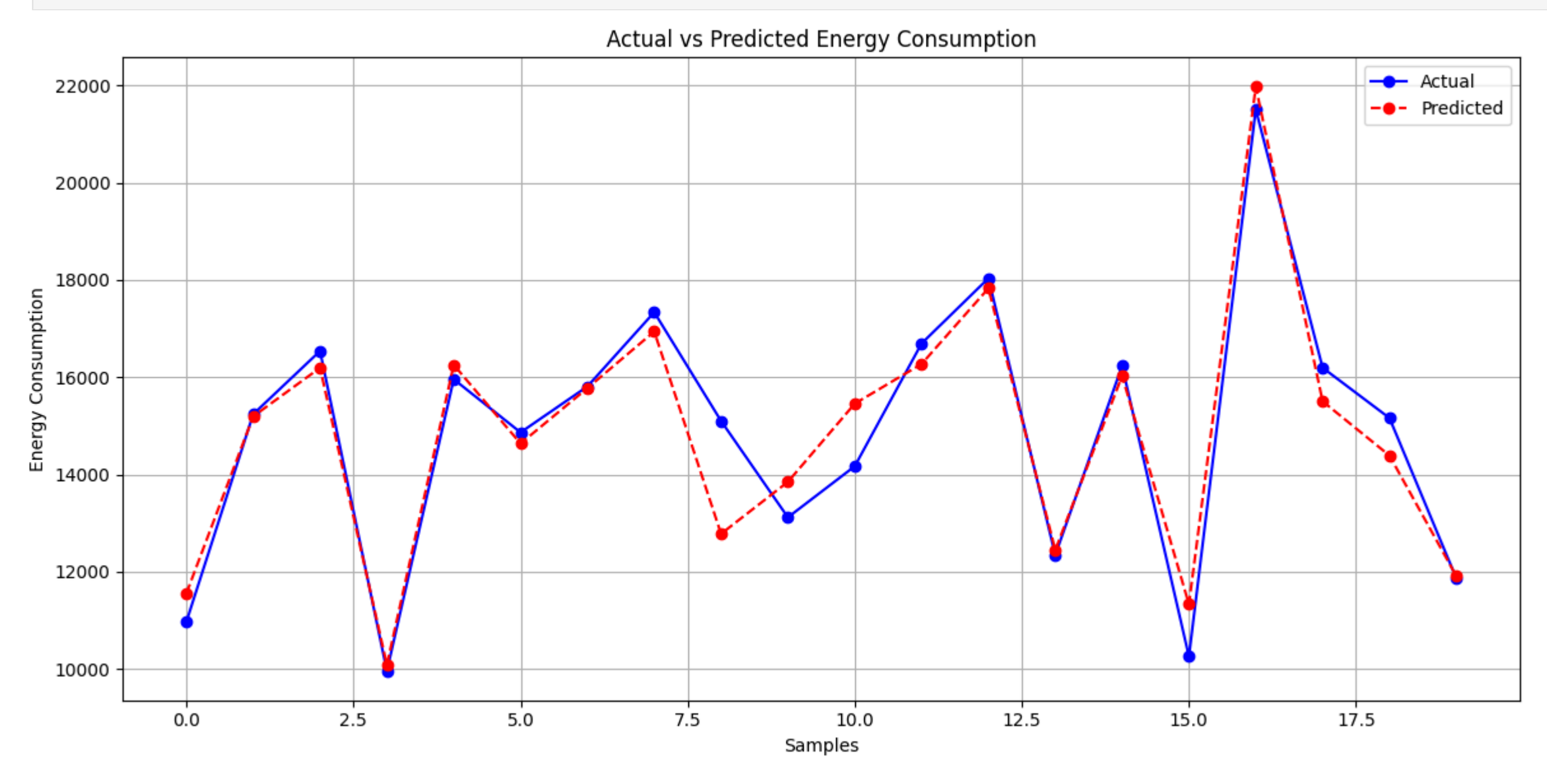
In [17]:
```python
# Suggestions based on feature importances
print("\nSuggestions:")
if fi_sorted.loc['hour', 'importance'] > 0.5:
    print("- 'hour' is the most important feature. Consider analyzing hourly patterns.")
if fi_sorted.loc['temperature', 'importance'] > 0.3:
    print("- 'temperature' has significant importance. Temperature fluctuations may strongly affect demand like more energy consumption in summer.")
if fi_sorted.loc['weekday', 'importance'] > 0.05:
    print("- 'weekday' is moderately important. Check for weekday-specific demand trends.")
if fi_sorted.loc['month', 'importance'] > 0.01:
    print("- 'month' shows some importance. Seasonal variations could impact demand.")
if fi_sorted.loc['year', 'importance'] > 0.01:
    print("- 'year' has minor importance. Long-term trends might influence demand.")
if fi_sorted.loc['day', 'importance'] < 0.01:
    print("- 'day' has negligible importance. Consider excluding it from future models.")
```

```
Suggestions:
- 'hour' is the most important feature. Consider analyzing hourly patterns.
- 'temperature' has significant importance. Temperature fluctuations may strongly affect demand like more energy consumption in summer.
- 'weekday' is moderately important. Check for weekday-specific demand trends.
- 'month' shows some importance. Seasonal variations could impact demand.
- 'day' has negligible importance. Consider excluding it from future models.
```

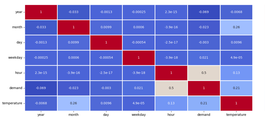
In [18]:
```python
from sklearn.neighbors import KNeighborsRegressor

# Define features and target
FEATURES = ['day', 'hour', 'weekday', 'temperature', 'month', 'year']
TARGET = 'demand'

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df[FEATURES], df[TARGET], test_size=0.2, random_state=42)

# Initialize KNN Regressor
reg = KNeighborsRegressor(n_neighbors=5)

# Fit the model
reg.fit(X_train, y_train)

# Predict on the test set
y_pred = reg.predict(X_test)

# Evaluate the model
r2 = reg.score(X_test, y_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)

print(f"R² Score: {r2}")
print(f"RMSE: {rmse}")
print(f"Mean Absolute Error: {mae}")

# KNN does not have a 'feature_importances_' attribute, so no feature importance analysis is possible with KNN.

predictions_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print("\nPredictions vs Actual:")
print(predictions_df.head(20))
```

```
R² Score: 0.924419375214687
RMSE: 798.7856512533248
Mean Absolute Error: 586.4006889574099

Predictions vs Actual:
        Actual  Predicted
55418  10973.0    11555.4
61591  15241.0    15195.4
43238  16540.0    16194.6
72650   9961.0    10088.2
43045  15960.0    16254.8
18439  14866.0    14637.8
58624  15811.0    15776.2
2483   17337.0    16940.6
33150  15101.0    12781.0
85198  13120.0    13850.0
69622  14171.0    15457.6
51992  16695.0    16272.8
60249  18035.0    17828.8
17113  12345.0    12448.0
27116  16235.0    16029.4
55806  10260.0    11334.0
73507  21507.0    21974.2
33248  16196.0    15505.4
63284  15162.0    14390.2
12053  11868.0    11913.8
```

In [19]:
```python
# Plotting actual vs predicted
plt.figure(figsize=(12, 6))

plt.plot(y_test.values[:20], marker='o', linestyle='-', color='b', label='Actual')
plt.plot(y_pred[:20], marker='o', linestyle='--', color='r', label='Predicted')

plt.title('Actual vs Predicted Energy Consumption')
plt.xlabel('Samples')
plt.ylabel('Energy Consumption')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [20]:
```python
#heatmap analysis
import numpy as np
df['date'] = pd.to_datetime(df['date'], format='%m/%d/%Y')

# Select only numeric columns
numeric_df = df.select_dtypes(include=[np.number])

# Plot the heatmap
plt.figure(figsize=(16, 6))
sns.heatmap(numeric_df.corr(), annot=True, linewidths=1, fmt=".2g", cmap="coolwarm")
plt.xticks(rotation=0)  # Rotate x-axis labels to be horizontal
plt.show()
```


year
month
day
weekday
hour
demand
temperature
1
-0.033
-0.0013
-0.00025
2.3e-15
-0.063
0.0068
-0.033
1
0.0098
0.0094
-3.5e-16
-0.023
0.26
-0.0013
0.0098
1
0.00054
-2.7e-17
-0.002
0.0096
-0.00025
0.0094
-0.00054
1
-3.3e-18
0.021
-4.5e-05
2.3e-15
-3.5e-16
-2.7e-17
-3.3e-18
1
0.5
0.11
-0.063
-0.023
-0.002
0.021
0.5
1
0.31
0.0068
0.26
0.0096
-4.5e-05
0.11
0.31
1
year
month
day
weekday
hour
demand
temperature
1.0
0.8
0.6
0.4
0.2
0.0

In [ ]: