## Importing necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from statsmodels.tsa.statespace.sarimax import SARIMAX
import warnings
```

## Data Preprocessing

In [2]:
```
df = pd.read_csv('UrbanEdgeApparel.csv')
```

In [3]:
```
df.head()
```

Out[3]:

| | Order ID | Order Status | Order Date | Order Day of Week | Order Month | Order Year | Customer ID | Company ID |
|---|---|---|---|---|---|---|---|---|
| **0** | 104 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_3161 | Company_87239 |
| **1** | 104 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_3161 | Company_87239 |
| **2** | 107 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_2040 | Company_83024 |
| **3** | 107 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_2040 | Company_83024 |
| **4** | 107 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_2040 | Company_83024 |

5 rows × 21 columns

In [70]:
```
d = df.groupby('Product ID')['Total Selling Price'].agg(['min', 'max'])
d.head(15)
```

Out[70]:

| | min | max |
|---|---|---|
| **Product ID** | | |
| **Prod_100** | 0.00 | 0.0 |
| **Prod_1000** | 3.95 | 2517.5 |
| **Prod_10021** | 1.95 | 250.0 |
| **Prod_1003** | 2.25 | 208.0 |
| **Prod_1005** | 3.95 | 4250.0 |
| **Prod_1007** | 3.50 | 1012.6 |
| **Prod_1009** | 3.00 | 10800.0 |
| **Prod_1012** | 2.25 | 450.0 |
| **Prod_10150** | 3.00 | 3498.0 |
| **Prod_10160** | 3.00 | 18.0 |
| **Prod_10161** | 2.25 | 225.0 |
| **Prod_1017** | 3.50 | 1424.0 |
| **Prod_1018** | 3.50 | 70.0 |
| **Prod_1020** | 4.50 | 80.0 |
| **Prod_10201** | 4.00 | 120.0 |

In [4]:
```
df.shape
```
Out[4]: (89644, 21)

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Order ID                          0
Order Status                      0
Order Date                      151
Order Day of Week               151
Order Month                     151
Order Year                      151
Customer ID                       0
Company ID                     3108
Product ID                        0
Product Variant ID             3444
Product Unit Selling Price        0
Product Quantity                  0
Total Selling Price               0
Payment Status                    0
Shipment ID                       0
Shipment Number                   0
Shipping Address Type             0
Shipping City                    37
Shipping State                   90
Shipping Postal Code             37
Shipping Country                 37
dtype: int64
```

In [6]:
```
df['Order Status']
```

Out[6]:
```
0        Completed
1        Completed
2        Completed
3        Completed
4        Completed
           ...
89639    Completed
89640    Completed
89641    Completed
89642    Completed
89643    Completed
Name: Order Status, Length: 89644, dtype: object
```

In [7]:
```
# Displaying unique values in Payment Status
print(df['Payment Status'].unique())

# Dropping rows where Order Status is 'Completed' and Payment Status is 'Dec
filtered_data = df[~((df['Order Status'] == 'Completed') & (df['Payment Stat

print("Original data shape:", df.shape)
print("Filtered data shape:", filtered_data.shape)
```

```
['Received' 'Pending' 'Canceled' 'Declined']
Original data shape: (89644, 21)
Filtered data shape: (89633, 21)
```

In [8]:
```
filtered_data.head()
```

Out[8]:

| | Order ID | Order Status | Order Date | Order Day of Week | Order Month | Order Year | Customer ID | Company ID |
|---|---|---|---|---|---|---|---|---|
| 0 | 104 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_3161 | Company_87239 |
| 1 | 104 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_3161 | Company_87239 |
| 2 | 107 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_2040 | Company_83024 |
| 3 | 107 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_2040 | Company_83024 |
| 4 | 107 | Completed | 6/6/2013 | Thursday | June | 2013.0 | Cust_2040 | Company_83024 |

5 rows × 21 columns

In [10]:

```
warnings.filterwarnings("ignore")
# Converting date columns to datetime format
filtered_data['Order Date'] = pd.to_datetime(filtered_data['Order Date'])

# Extracting additional features from the date columns
filtered_data['Order Day of Week'] = filtered_data['Order Date'].dt.dayofwee
filtered_data['Order Month'] = filtered_data['Order Date'].dt.month
filtered_data['Order Year'] = filtered_data['Order Date'].dt.year

filtered_data.head()
```

Out[10]:

| | Order ID | Order Status | Order Date | Order Day of Week | Order Month | Order Year | Customer ID | Company ID | Prod |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 104 | Completed | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_3161 | Company_87239 | Prod |
| **1** | 104 | Completed | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_3161 | Company_87239 | Prod_ |
| **2** | 107 | Completed | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_2040 | Company_83024 | Proc |
| **3** | 107 | Completed | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_2040 | Company_83024 | Proc |
| **4** | 107 | Completed | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_2040 | Company_83024 | Proc |

5 rows × 21 columns

In [11]:
```
filtered_data.isnull().sum()
```

Out[11]:
```
Order ID                       0
Order Status                   0
Order Date                   140
Order Day of Week            140
Order Month                  140
Order Year                   140
Customer ID                    0
Company ID                  3097
Product ID                     0
Product Variant ID          3444
Product Unit Selling Price     0
Product Quantity               0
Total Selling Price            0
Payment Status                 0
Shipment ID                    0
Shipment Number                0
Shipping Address Type          0
Shipping City                 37
Shipping State                90
Shipping Postal Code          37
Shipping Country              37
dtype: int64
```

In [12]:
```
# Encoding Order Status column values to have numbers inplace of strings
warnings.filterwarnings("ignore")
label_encoder = LabelEncoder()
```

```
filtered_data['Order Status'] = label_encoder.fit_transform(filtered_data['O
filtered_data['Order Status'].unique()
```

Out[12]:
```
array([1, 0, 3, 2])
```

In [13]:
```
filtered_data = filtered_data.dropna()
```

In [14]:
```
filtered_data.isnull().sum()
```

Out[14]:
```
Order ID                      0
Order Status                  0
Order Date                    0
Order Day of Week             0
Order Month                   0
Order Year                    0
Customer ID                   0
Company ID                    0
Product ID                    0
Product Variant ID            0
Product Unit Selling Price    0
Product Quantity              0
Total Selling Price           0
Payment Status                0
Shipment ID                   0
Shipment Number               0
Shipping Address Type         0
Shipping City                 0
Shipping State                0
Shipping Postal Code          0
Shipping Country              0
dtype: int64
```

In [15]:
```
desc_stats = filtered_data.describe()
desc_stats
```

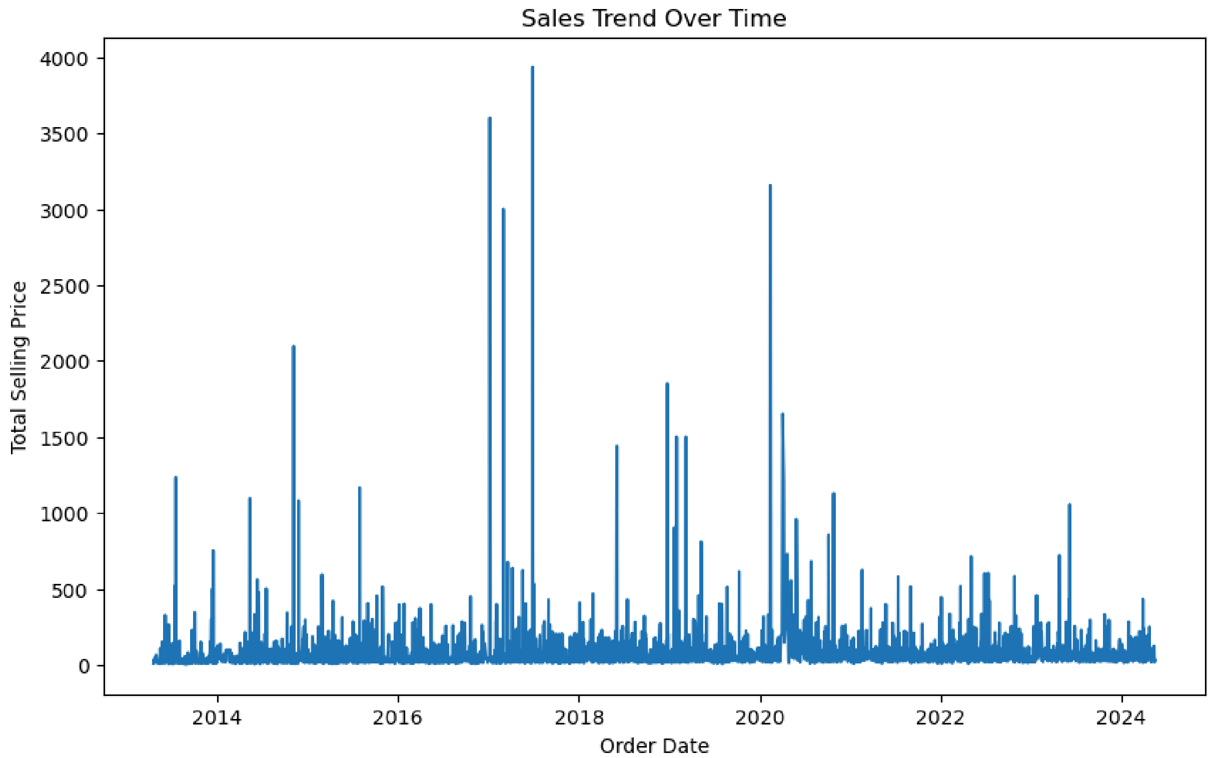
Out[15]:

| | Order ID | Order Status | Order Date | Order Day of Week | Order Month |
|---|---|---|---|---|---|
| count | 83217.000000 | 83217.000000 | 83217 | 83217.000000 | 83217.000000 |
| mean | 5497.746470 | 1.008087 | 2019-01-12 11:11:13.694076928 | 2.294651 | 6.324297 |
| min | 5.000000 | 0.000000 | 2013-04-17 00:00:00 | 0.000000 | 1.000000 |
| 25% | 2258.000000 | 1.000000 | 2016-09-06 00:00:00 | 1.000000 | 4.000000 |
| 50% | 4810.000000 | 1.000000 | 2019-01-23 00:00:00 | 2.000000 | 6.000000 |
| 75% | 8481.000000 | 1.000000 | 2021-07-21 00:00:00 | 4.000000 | 9.000000 |
| max | 13831.000000 | 3.000000 | 2024-05-16 00:00:00 | 6.000000 | 12.000000 |
| std | 3820.420299 | 0.241356 | NaN | 1.751071 | 3.129121 |

## Visualizing the data

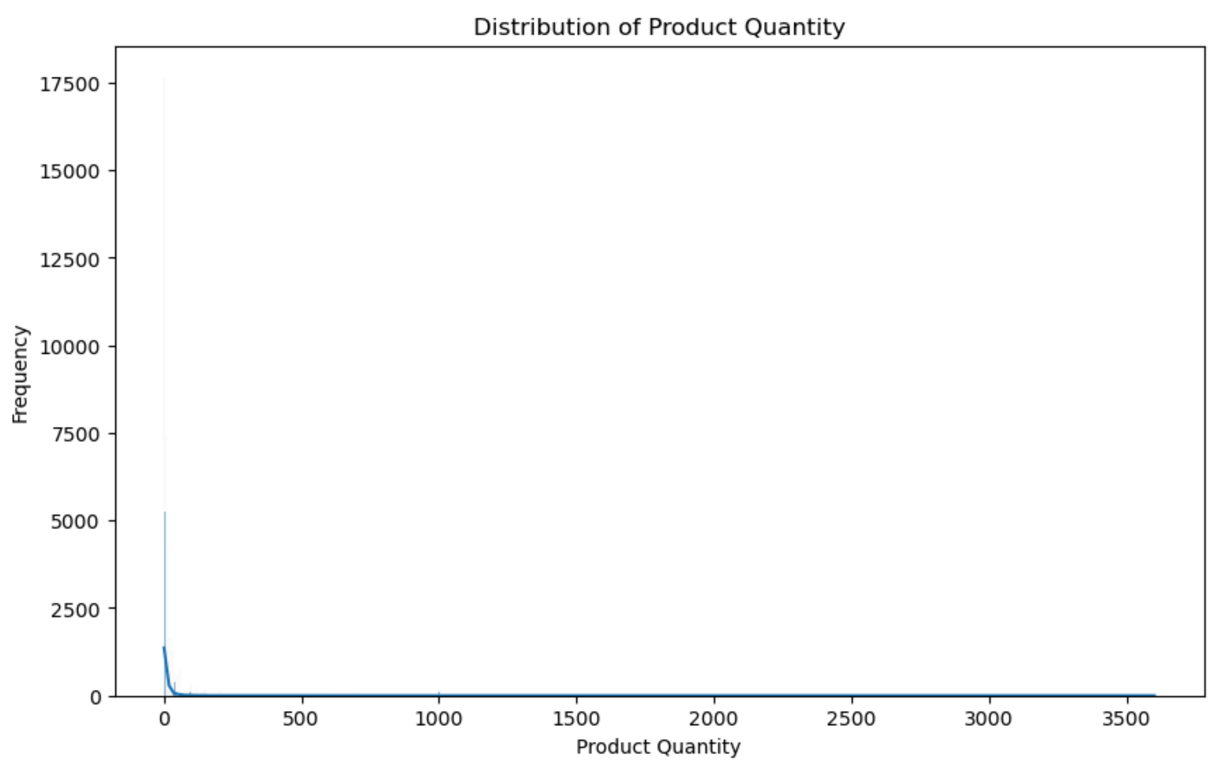
In [16]:

```
# Sales Trend Over Time
warnings.filterwarnings("ignore")
plt.figure(figsize=(10, 6))
sns.lineplot(data=filtered_data, x='Order Date', y='Total Selling Price', ci
plt.title('Sales Trend Over Time')
plt.xlabel('Order Date')
plt.ylabel('Total Selling Price')
plt.show()
```

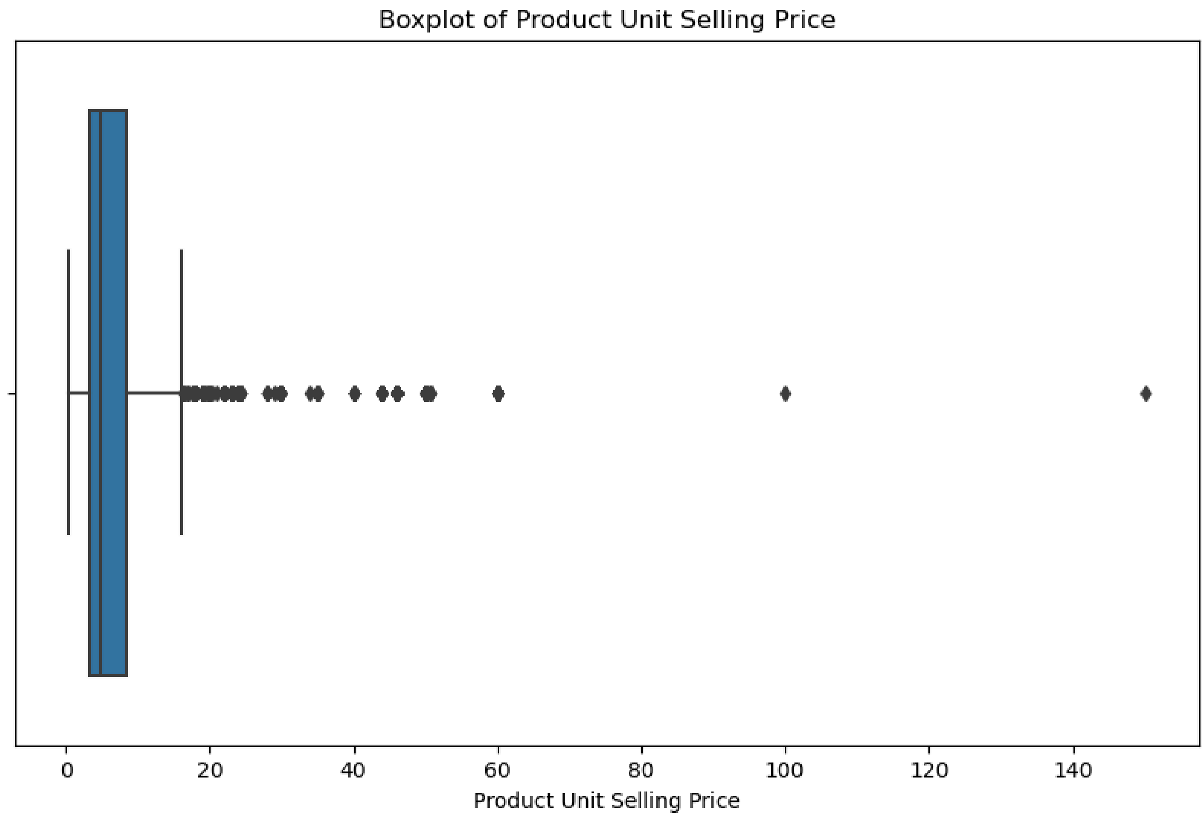



In [17]:

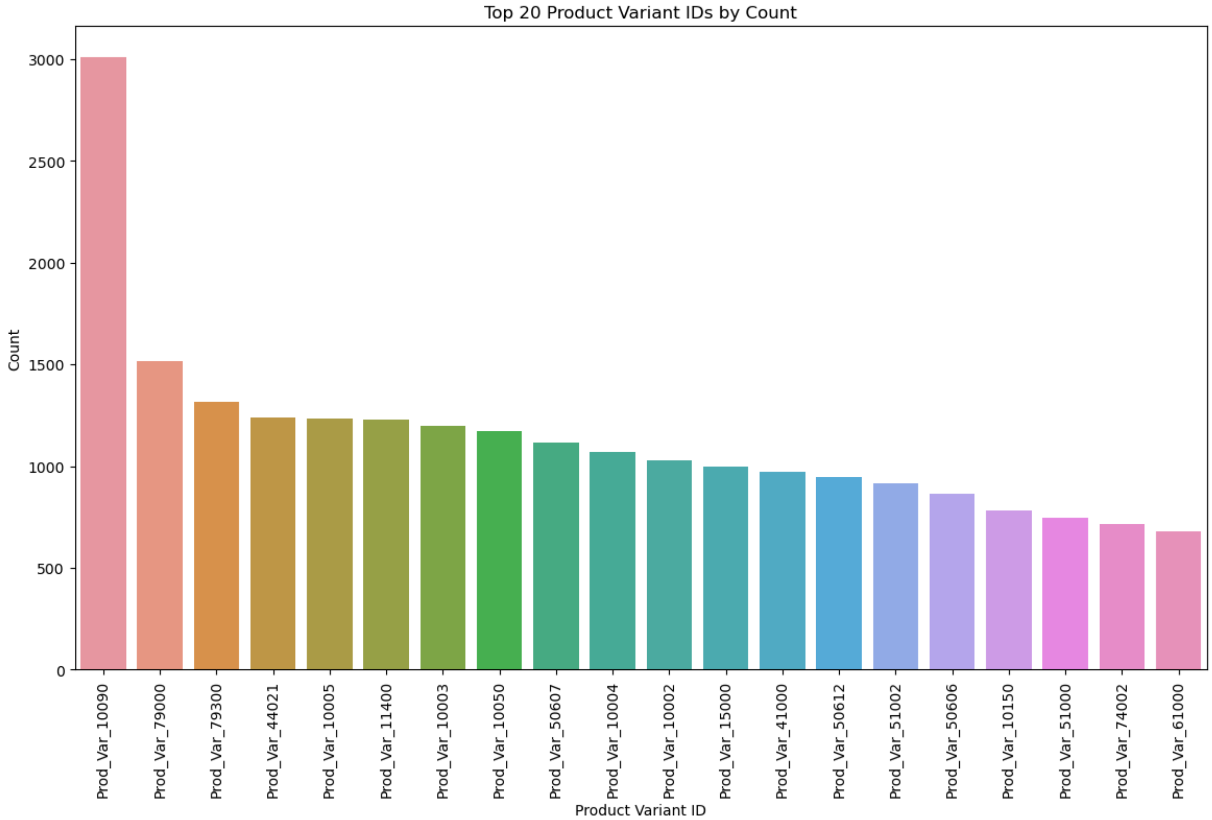
```
# Product Quantity Distribution
warnings.filterwarnings("ignore")
plt.figure(figsize=(10, 6))
sns.histplot(filtered_data['Product Quantity'], kde=True)
plt.title('Distribution of Product Quantity')
plt.xlabel('Product Quantity')
plt.ylabel('Frequency')
plt.show()
```



In [18]:
```
# Boxplot of Product Unit Selling Price
plt.figure(figsize=(10, 6))
sns.boxplot(x=filtered_data['Product Unit Selling Price'])
plt.title('Boxplot of Product Unit Selling Price')
plt.xlabel('Product Unit Selling Price')
plt.show()
```




In [19]:
```
# Top 20 Product Variant IDs by Count
plt.figure(figsize=(14, 8))
product_variant_counts = filtered_data['Product Variant ID'].value_counts().
sns.barplot(x=product_variant_counts.index, y=product_variant_counts.values)
plt.xticks(rotation=90)
plt.title('Top 20 Product Variant IDs by Count')
plt.xlabel('Product Variant ID')
plt.ylabel('Count')
plt.show()
```




From the above plots, we realize the need of scaling.

In [20]:
```
# Applying log transformation to reduce skewness
filtered_data['Total Selling Price'] = np.log1p(filtered_data['Total Selling
filtered_data['Product Quantity'] = np.log1p(filtered_data['Product Quantity
filtered_data['Product Unit Selling Price'] = np.log1p(filtered_data['Produc

# Identifying numeric columns for scaling
numeric_cols = ['Product Unit Selling Price', 'Product Quantity', 'Total Sel

scaler = MinMaxScaler()
filtered_data[numeric_cols] = scaler.fit_transform(filtered_data[numeric_col
filtered_data.head()
```

Out[20]:

| | Order ID | Order Status | Order Date | Order Day of Week | Order Month | Order Year | Customer ID | Company ID | Product |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 104 | 1 | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_3161 | Company_87239 | Prod_50 |
| **1** | 104 | 1 | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_3161 | Company_87239 | Prod_700 |
| **2** | 107 | 1 | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_2040 | Company_83024 | Prod_10 |
| **3** | 107 | 1 | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_2040 | Company_83024 | Prod_10 |
| **4** | 107 | 1 | 2013-06-06 | 3.0 | 6.0 | 2013.0 | Cust_2040 | Company_83024 | Prod_10 |

5 rows × 21 columns

## Visualizing the same plots after scaling and transformations

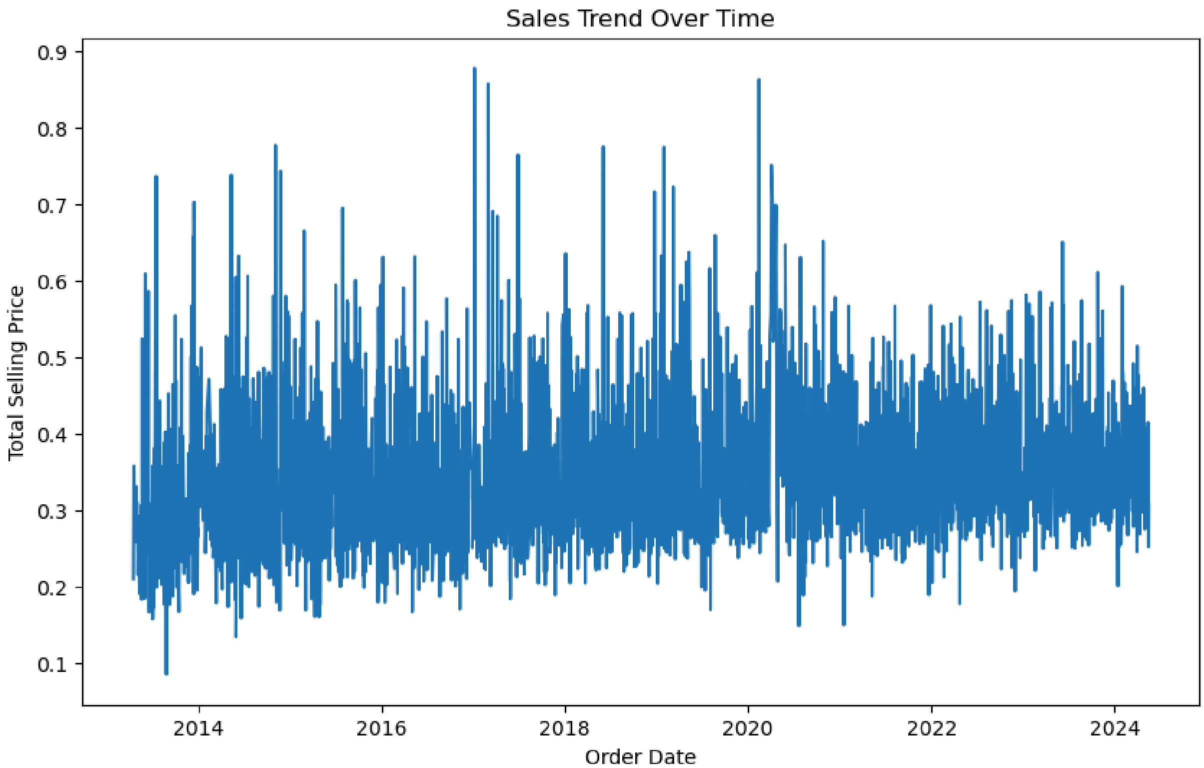
In [21]:

```
desc_stats_scaled = filtered_data.describe()
desc_stats_scaled
```

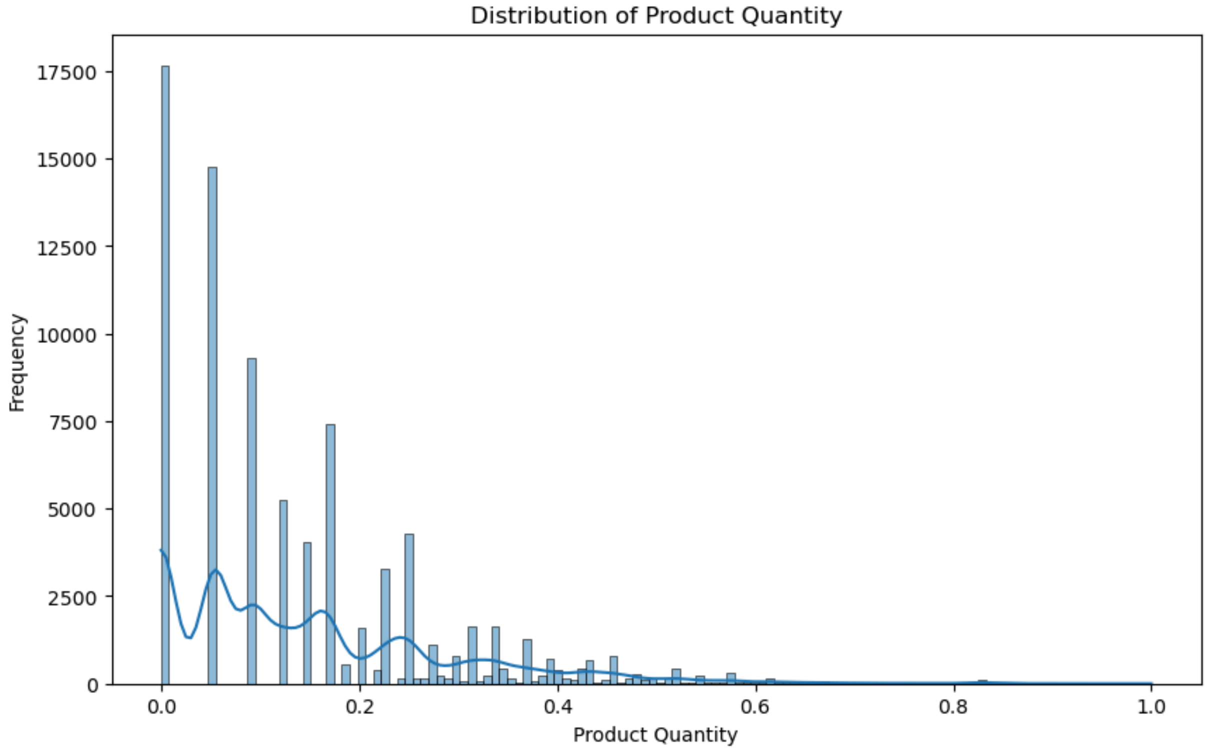
Out[21]:

| | Order ID | Order Status | Order Date | Order Day of Week | Order Month |
|---|---|---|---|---|---|
| **count** | 83217.000000 | 83217.000000 | 83217 | 83217.000000 | 83217.000000 |
| **mean** | 5497.746470 | 1.008087 | 2019-01-12 11:11:13.694076928 | 2.294651 | 6.324297 |
| **min** | 5.000000 | 0.000000 | 2013-04-17 00:00:00 | 0.000000 | 1.000000 |
| **25%** | 2258.000000 | 1.000000 | 2016-09-06 00:00:00 | 1.000000 | 4.000000 |
| **50%** | 4810.000000 | 1.000000 | 2019-01-23 00:00:00 | 2.000000 | 6.000000 |
| **75%** | 8481.000000 | 1.000000 | 2021-07-21 00:00:00 | 4.000000 | 9.000000 |
| **max** | 13831.000000 | 3.000000 | 2024-05-16 00:00:00 | 6.000000 | 12.000000 |
| **std** | 3820.420299 | 0.241356 | NaN | 1.751071 | 3.129121 |

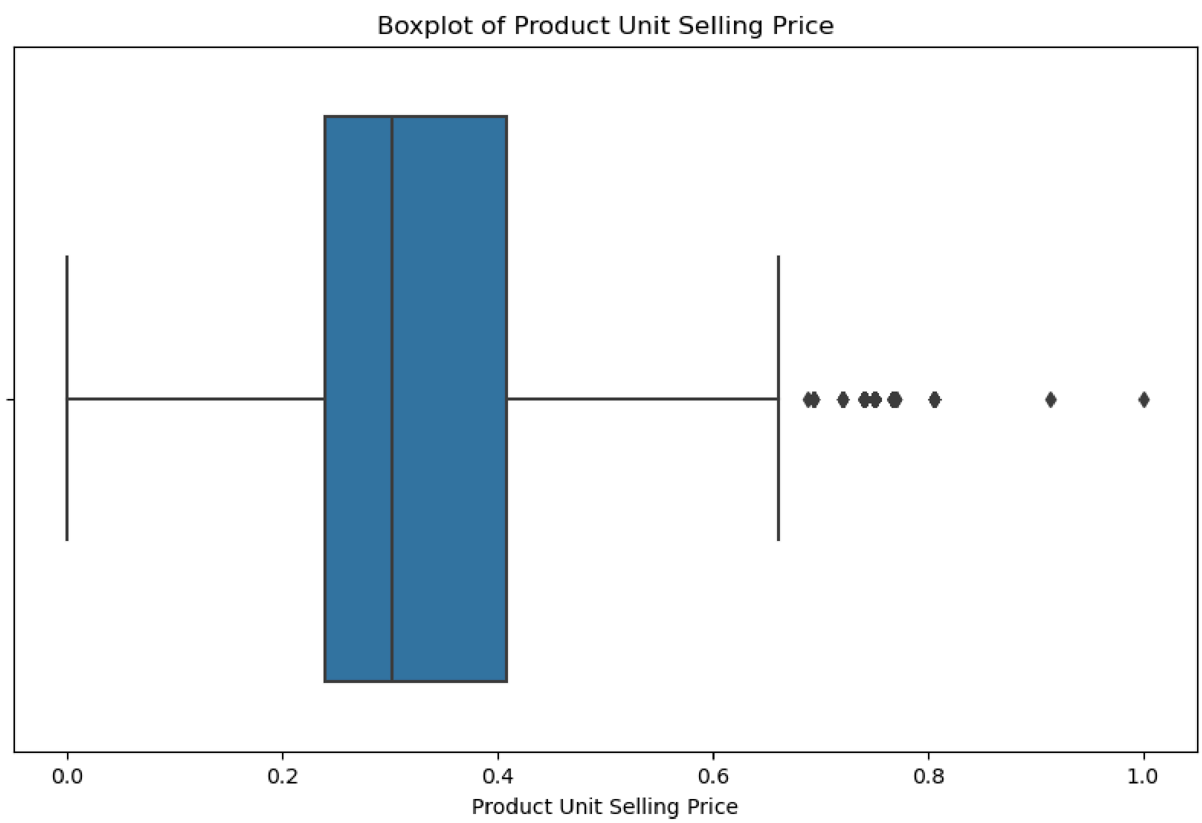
In [22]:
```
# Sales Trend Over Time
warnings.filterwarnings("ignore")
plt.figure(figsize=(10, 6))
sns.lineplot(data=filtered_data, x='Order Date', y='Total Selling Price', er
plt.title('Sales Trend Over Time')
plt.xlabel('Order Date')
plt.ylabel('Total Selling Price')
plt.show()
```



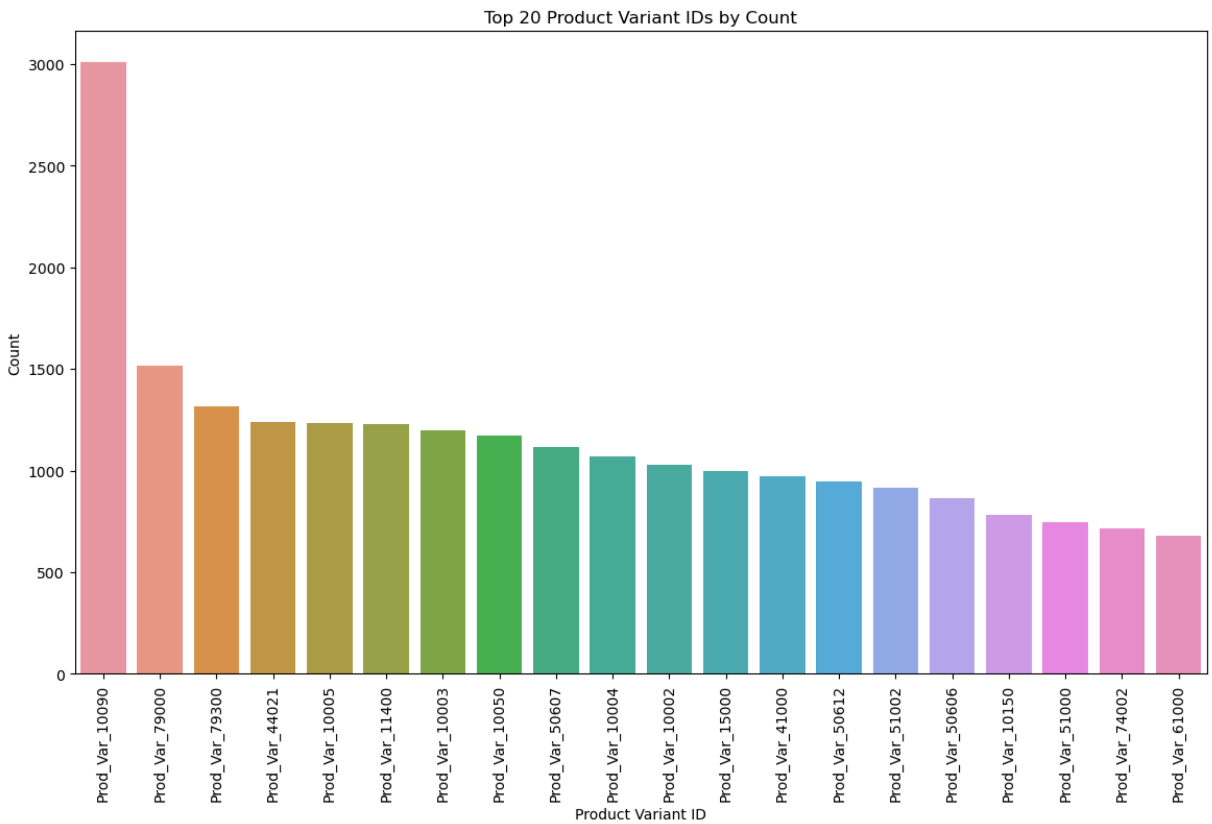

In [23]:
```
# Product Quantity Distribution
warnings.filterwarnings("ignore")
plt.figure(figsize=(10, 6))
sns.histplot(filtered_data['Product Quantity'], kde=True)
plt.title('Distribution of Product Quantity')
plt.xlabel('Product Quantity')
plt.ylabel('Frequency')
plt.show()
```




In [24]:
```python
# Boxplot of Product Unit Selling Price
plt.figure(figsize=(10, 6))
sns.boxplot(x=filtered_data['Product Unit Selling Price'])
plt.title('Boxplot of Product Unit Selling Price')
plt.xlabel('Product Unit Selling Price')
plt.show()
```



In [25]:
```
# Top 20 Product Variant IDs by Count
plt.figure(figsize=(14, 8))
product_variant_counts = filtered_data['Product Variant ID'].value_counts().
sns.barplot(x=product_variant_counts.index, y=product_variant_counts.values)
plt.xticks(rotation=90)
plt.title('Top 20 Product Variant IDs by Count')
plt.xlabel('Product Variant ID')
plt.ylabel('Count')
plt.show()
```




## Feature Engineering

In [26]:
```
filtered_data['Order Quarter'] = filtered_data['Order Date'].dt.quarter
filtered_data['Is Weekend'] = filtered_data['Order Date'].dt.weekday >= 5
```

In [27]:
```
data = filtered_data
```

In [28]:
```
# Converting Order Date to datetime
data['Order Date'] = pd.to_datetime(data['Order Date'])
```

In [29]:
```
# Aggregating sales data by month
monthly_sales = data.resample('M', on='Order Date')['Total Selling Price'].s

# Preparing data
monthly_sales.set_index('Order Date', inplace=True)
monthly_sales = monthly_sales['Total Selling Price']
```

In [30]:
```
# Train-test split
train_end_date = '2022-12-31'
train = monthly_sales[monthly_sales.index <= train_end_date]
test = monthly_sales[monthly_sales.index > train_end_date]
```

In [32]:
```
# Define function to calculate Accuracy
def cal_accuracy(y_true, y_pred):
    return (100-np.mean(np.abs((y_true - y_pred) / y_true)) * 100)
```

## Building Time Series Forecasting Model

In [33]:
```
def sarima_grid_search(train, test, p_values, d_values, q_values, P_values, 
    best_score, best_cfg, best_mape = float("inf"), None, None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                for P in P_values:
                    for D in D_values:
                        for Q in Q_values:
                            try:
                                order = (p,d,q)
                                seasonal_order = (P,D,Q,m)
                                model = SARIMAX(train, order=order, seasonal
                                model_fit = model.fit(disp=False)
                                forecast = model_fit.forecast(steps=len(test

                                # Align test and forecast data
                                aligned_test = test
                                aligned_forecast = pd.Series(forecast, index

                                mae = mean_absolute_error(aligned_test, alig
                                accuracy = cal_accuracy(aligned_test, aligne

                                if mae < best_score:
                                    best_score, best_cfg, best_accuracy = ma
                                print(f'SARIMA{order}{seasonal_order} MAE={m
                            except:
                                continue
    print(f'Best SARIMA{best_cfg} MAE={best_score:.3f} Accuracy={best_accura
    return best_cfg, best_score, best_mape
```

In [34]:
```
# Defining parameter ranges
p_values = [0, 1, 2]
d_values = [0, 1]
q_values = [0, 1, 2]
P_values = [0, 1, 2]
D_values = [0, 1]
Q_values = [0, 1, 2]
m = 12
```

In [35]:
```
warnings.filterwarnings("ignore")

best_cfg, best_mae, best_mape = sarima_grid_search(train, test, p_values, d_
```

```
model = SARIMAX(train, order=best_cfg[0], seasonal_order=best_cfg[1])
model_fit = model.fit(disp=False)

# Forecasting the next 2 years
forecast_steps = 24
forecast = model_fit.forecast(steps=forecast_steps)
```

```
SARIMA(0, 0, 0)(0, 0, 0, 12) MAE=175.752 Accuracy=0.000%
SARIMA(0, 0, 0)(0, 0, 1, 12) MAE=142.487 Accuracy=16.291%
SARIMA(0, 0, 0)(0, 0, 2, 12) MAE=63.774 Accuracy=61.397%
SARIMA(0, 0, 0)(0, 1, 0, 12) MAE=75.132 Accuracy=53.797%
SARIMA(0, 0, 0)(0, 1, 1, 12) MAE=58.116 Accuracy=61.344%
SARIMA(0, 0, 0)(0, 1, 2, 12) MAE=56.230 Accuracy=62.517%
SARIMA(0, 0, 0)(1, 0, 0, 12) MAE=47.473 Accuracy=70.882%
SARIMA(0, 0, 0)(1, 0, 1, 12) MAE=50.771 Accuracy=66.005%
SARIMA(0, 0, 0)(1, 0, 2, 12) MAE=49.903 Accuracy=66.549%
SARIMA(0, 0, 0)(1, 1, 0, 12) MAE=72.848 Accuracy=54.452%
SARIMA(0, 0, 0)(1, 1, 1, 12) MAE=57.240 Accuracy=61.895%
SARIMA(0, 0, 0)(1, 1, 2, 12) MAE=51.512 Accuracy=65.615%
SARIMA(0, 0, 0)(2, 0, 0, 12) MAE=54.071 Accuracy=65.908%
SARIMA(0, 0, 0)(2, 0, 1, 12) MAE=50.480 Accuracy=66.191%
SARIMA(0, 0, 0)(2, 0, 2, 12) MAE=45.704 Accuracy=69.377%
SARIMA(0, 0, 0)(2, 1, 0, 12) MAE=42.580 Accuracy=71.267%
SARIMA(0, 0, 0)(2, 1, 1, 12) MAE=45.153 Accuracy=69.148%
SARIMA(0, 0, 0)(2, 1, 2, 12) MAE=31.931 Accuracy=80.131%
SARIMA(0, 0, 1)(0, 0, 0, 12) MAE=177.654 Accuracy=-1.101%
SARIMA(0, 0, 1)(0, 0, 1, 12) MAE=132.615 Accuracy=22.786%
SARIMA(0, 0, 1)(0, 0, 2, 12) MAE=56.718 Accuracy=65.888%
SARIMA(0, 0, 1)(0, 1, 0, 12) MAE=73.308 Accuracy=54.967%
SARIMA(0, 0, 1)(0, 1, 1, 12) MAE=54.441 Accuracy=63.129%
SARIMA(0, 0, 1)(0, 1, 2, 12) MAE=56.071 Accuracy=62.114%
SARIMA(0, 0, 1)(1, 0, 0, 12) MAE=35.903 Accuracy=78.464%
SARIMA(0, 0, 1)(1, 0, 1, 12) MAE=67757.709 Accuracy=-43123.282%
SARIMA(0, 0, 1)(1, 0, 2, 12) MAE=47.032 Accuracy=67.816%
SARIMA(0, 0, 1)(1, 1, 0, 12) MAE=72.647 Accuracy=54.381%
SARIMA(0, 0, 1)(1, 1, 1, 12) MAE=55.674 Accuracy=62.364%
SARIMA(0, 0, 1)(1, 1, 2, 12) MAE=50.650 Accuracy=65.608%
SARIMA(0, 0, 1)(2, 0, 0, 12) MAE=49.522 Accuracy=68.413%
SARIMA(0, 0, 1)(2, 0, 1, 12) MAE=35.708 Accuracy=78.593%
SARIMA(0, 0, 1)(2, 0, 2, 12) MAE=37.533 Accuracy=74.282%
SARIMA(0, 0, 1)(2, 1, 0, 12) MAE=44.971 Accuracy=69.357%
SARIMA(0, 0, 1)(2, 1, 1, 12) MAE=50.263 Accuracy=65.495%
SARIMA(0, 0, 1)(2, 1, 2, 12) MAE=50.150 Accuracy=65.801%
SARIMA(0, 0, 2)(0, 0, 0, 12) MAE=174.208 Accuracy=0.727%
SARIMA(0, 0, 2)(0, 0, 1, 12) MAE=126.870 Accuracy=26.538%
SARIMA(0, 0, 2)(0, 0, 2, 12) MAE=69.326 Accuracy=59.133%
SARIMA(0, 0, 2)(0, 1, 0, 12) MAE=73.040 Accuracy=55.201%
SARIMA(0, 0, 2)(0, 1, 1, 12) MAE=50.169 Accuracy=65.456%
SARIMA(0, 0, 2)(0, 1, 2, 12) MAE=47.285 Accuracy=67.275%
SARIMA(0, 0, 2)(1, 0, 0, 12) MAE=30.309 Accuracy=82.040%
SARIMA(0, 0, 2)(1, 0, 1, 12) MAE=37.975 Accuracy=73.352%
SARIMA(0, 0, 2)(1, 0, 2, 12) MAE=123.740 Accuracy=29.266%
SARIMA(0, 0, 2)(1, 1, 0, 12) MAE=73.163 Accuracy=54.139%
SARIMA(0, 0, 2)(1, 1, 1, 12) MAE=48.262 Accuracy=66.663%
SARIMA(0, 0, 2)(1, 1, 2, 12) MAE=46.116 Accuracy=68.165%
SARIMA(0, 0, 2)(2, 0, 0, 12) MAE=40.429 Accuracy=74.012%
SARIMA(0, 0, 2)(2, 0, 1, 12) MAE=30.285 Accuracy=82.058%
SARIMA(0, 0, 2)(2, 0, 2, 12) MAE=36.985 Accuracy=73.983%
SARIMA(0, 0, 2)(2, 1, 0, 12) MAE=43.573 Accuracy=70.422%
SARIMA(0, 0, 2)(2, 1, 1, 12) MAE=47.870 Accuracy=66.659%
SARIMA(0, 0, 2)(2, 1, 2, 12) MAE=60.560 Accuracy=58.697%
SARIMA(0, 1, 0)(0, 0, 0, 12) MAE=82.727 Accuracy=56.488%
SARIMA(0, 1, 0)(0, 0, 1, 12) MAE=78.576 Accuracy=58.441%
```

```
SARIMA(0, 1, 0)(0, 0, 2, 12) MAE=63.680 Accuracy=66.307%
SARIMA(0, 1, 0)(0, 1, 0, 12) MAE=44.032 Accuracy=73.181%
SARIMA(0, 1, 0)(0, 1, 1, 12) MAE=37.315 Accuracy=74.632%
SARIMA(0, 1, 0)(0, 1, 2, 12) MAE=39.079 Accuracy=73.463%
SARIMA(0, 1, 0)(1, 0, 0, 12) MAE=75.934 Accuracy=59.803%
SARIMA(0, 1, 0)(1, 0, 1, 12) MAE=41.902 Accuracy=75.646%
SARIMA(0, 1, 0)(1, 0, 2, 12) MAE=43.692 Accuracy=74.143%
SARIMA(0, 1, 0)(1, 1, 0, 12) MAE=69.441 Accuracy=56.454%
SARIMA(0, 1, 0)(1, 1, 1, 12) MAE=39.124 Accuracy=73.374%
SARIMA(0, 1, 0)(1, 1, 2, 12) MAE=37.158 Accuracy=75.155%
SARIMA(0, 1, 0)(2, 0, 0, 12) MAE=53.815 Accuracy=71.666%
SARIMA(0, 1, 0)(2, 0, 1, 12) MAE=43.302 Accuracy=74.348%
SARIMA(0, 1, 0)(2, 0, 2, 12) MAE=42.178 Accuracy=75.810%
SARIMA(0, 1, 0)(2, 1, 0, 12) MAE=35.451 Accuracy=78.028%
SARIMA(0, 1, 0)(2, 1, 1, 12) MAE=38.870 Accuracy=73.841%
SARIMA(0, 1, 0)(2, 1, 2, 12) MAE=39.503 Accuracy=74.293%
SARIMA(0, 1, 1)(0, 0, 0, 12) MAE=67.285 Accuracy=64.151%
SARIMA(0, 1, 1)(0, 0, 1, 12) MAE=58.444 Accuracy=68.855%
SARIMA(0, 1, 1)(0, 0, 2, 12) MAE=44.037 Accuracy=75.784%
SARIMA(0, 1, 1)(0, 1, 0, 12) MAE=43.148 Accuracy=73.328%
SARIMA(0, 1, 1)(0, 1, 1, 12) MAE=41.790 Accuracy=69.951%
SARIMA(0, 1, 1)(0, 1, 2, 12) MAE=44.035 Accuracy=68.319%
SARIMA(0, 1, 1)(1, 0, 0, 12) MAE=48.952 Accuracy=73.762%
SARIMA(0, 1, 1)(1, 0, 1, 12) MAE=36.070 Accuracy=74.940%
SARIMA(0, 1, 1)(1, 0, 2, 12) MAE=38.419 Accuracy=73.186%
SARIMA(0, 1, 1)(1, 1, 0, 12) MAE=62.541 Accuracy=61.138%
SARIMA(0, 1, 1)(1, 1, 1, 12) MAE=43.547 Accuracy=68.660%
SARIMA(0, 1, 1)(1, 1, 2, 12) MAE=40.142 Accuracy=71.558%
SARIMA(0, 1, 1)(2, 0, 0, 12) MAE=32.803 Accuracy=80.337%
SARIMA(0, 1, 1)(2, 0, 1, 12) MAE=37.875 Accuracy=73.572%
SARIMA(0, 1, 1)(2, 0, 2, 12) MAE=35.683 Accuracy=75.434%
SARIMA(0, 1, 1)(2, 1, 0, 12) MAE=38.078 Accuracy=77.749%
SARIMA(0, 1, 1)(2, 1, 1, 12) MAE=41.628 Accuracy=70.757%
SARIMA(0, 1, 1)(2, 1, 2, 12) MAE=41.559 Accuracy=71.026%
SARIMA(0, 1, 2)(0, 0, 0, 12) MAE=74.088 Accuracy=61.066%
SARIMA(0, 1, 2)(0, 0, 1, 12) MAE=66.419 Accuracy=64.793%
SARIMA(0, 1, 2)(0, 0, 2, 12) MAE=53.588 Accuracy=71.477%
SARIMA(0, 1, 2)(0, 1, 0, 12) MAE=43.194 Accuracy=73.337%
SARIMA(0, 1, 2)(0, 1, 1, 12) MAE=41.665 Accuracy=70.060%
SARIMA(0, 1, 2)(0, 1, 2, 12) MAE=44.004 Accuracy=68.345%
SARIMA(0, 1, 2)(1, 0, 0, 12) MAE=56.709 Accuracy=69.908%
SARIMA(0, 1, 2)(1, 0, 1, 12) MAE=35.807 Accuracy=75.253%
SARIMA(0, 1, 2)(1, 0, 2, 12) MAE=38.094 Accuracy=73.485%
SARIMA(0, 1, 2)(1, 1, 0, 12) MAE=62.957 Accuracy=60.856%
SARIMA(0, 1, 2)(1, 1, 1, 12) MAE=43.506 Accuracy=68.696%
SARIMA(0, 1, 2)(1, 1, 2, 12) MAE=39.988 Accuracy=71.693%
SARIMA(0, 1, 2)(2, 0, 0, 12) MAE=37.352 Accuracy=78.980%
SARIMA(0, 1, 2)(2, 0, 1, 12) MAE=37.478 Accuracy=73.933%
SARIMA(0, 1, 2)(2, 0, 2, 12) MAE=35.871 Accuracy=75.169%
SARIMA(0, 1, 2)(2, 1, 0, 12) MAE=38.025 Accuracy=77.767%
SARIMA(0, 1, 2)(2, 1, 1, 12) MAE=38.172 Accuracy=74.021%
SARIMA(0, 1, 2)(2, 1, 2, 12) MAE=41.536 Accuracy=71.053%
SARIMA(1, 0, 0)(0, 0, 0, 12) MAE=111.362 Accuracy=39.787%
SARIMA(1, 0, 0)(0, 0, 1, 12) MAE=106.270 Accuracy=41.745%
SARIMA(1, 0, 0)(0, 0, 2, 12) MAE=88.359 Accuracy=52.041%
SARIMA(1, 0, 0)(0, 1, 0, 12) MAE=68.039 Accuracy=58.391%
```

```
SARIMA(1, 0, 0)(0, 1, 1, 12) MAE=38.931 Accuracy=72.097%
SARIMA(1, 0, 0)(0, 1, 2, 12) MAE=43.230 Accuracy=69.161%
SARIMA(1, 0, 0)(1, 0, 0, 12) MAE=101.814 Accuracy=44.067%
SARIMA(1, 0, 0)(1, 0, 1, 12) MAE=40.454 Accuracy=77.111%
SARIMA(1, 0, 0)(1, 0, 2, 12) MAE=42.126 Accuracy=75.967%
SARIMA(1, 0, 0)(1, 1, 0, 12) MAE=72.441 Accuracy=54.430%
SARIMA(1, 0, 0)(1, 1, 1, 12) MAE=42.485 Accuracy=69.661%
SARIMA(1, 0, 0)(1, 1, 2, 12) MAE=45.417 Accuracy=67.619%
SARIMA(1, 0, 0)(2, 0, 0, 12) MAE=69.780 Accuracy=62.543%
SARIMA(1, 0, 0)(2, 0, 1, 12) MAE=41.831 Accuracy=76.105%
SARIMA(1, 0, 0)(2, 0, 2, 12) MAE=42.415 Accuracy=75.808%
SARIMA(1, 0, 0)(2, 1, 0, 12) MAE=44.769 Accuracy=69.598%
SARIMA(1, 0, 0)(2, 1, 1, 12) MAE=45.277 Accuracy=67.823%
SARIMA(1, 0, 0)(2, 1, 2, 12) MAE=46.258 Accuracy=67.124%
SARIMA(1, 0, 1)(0, 0, 0, 12) MAE=90.200 Accuracy=52.390%
SARIMA(1, 0, 1)(0, 0, 1, 12) MAE=79.941 Accuracy=57.493%
SARIMA(1, 0, 1)(0, 0, 2, 12) MAE=63.373 Accuracy=66.091%
SARIMA(1, 0, 1)(0, 1, 0, 12) MAE=59.796 Accuracy=63.416%
SARIMA(1, 0, 1)(0, 1, 1, 12) MAE=34.306 Accuracy=76.594%
SARIMA(1, 0, 1)(0, 1, 2, 12) MAE=35.051 Accuracy=76.063%
SARIMA(1, 0, 1)(1, 0, 0, 12) MAE=68.680 Accuracy=63.418%
SARIMA(1, 0, 1)(1, 0, 1, 12) MAE=34.738 Accuracy=78.291%
SARIMA(1, 0, 1)(1, 0, 2, 12) MAE=36.072 Accuracy=77.297%
SARIMA(1, 0, 1)(1, 1, 0, 12) MAE=69.870 Accuracy=56.069%
SARIMA(1, 0, 1)(1, 1, 1, 12) MAE=34.873 Accuracy=76.219%
SARIMA(1, 0, 1)(1, 1, 2, 12) MAE=34.232 Accuracy=77.092%
SARIMA(1, 0, 1)(2, 0, 0, 12) MAE=40.415 Accuracy=78.413%
SARIMA(1, 0, 1)(2, 0, 1, 12) MAE=35.721 Accuracy=77.519%
SARIMA(1, 0, 1)(2, 0, 2, 12) MAE=34.936 Accuracy=78.678%
SARIMA(1, 0, 1)(2, 1, 0, 12) MAE=39.861 Accuracy=74.017%
SARIMA(1, 0, 1)(2, 1, 1, 12) MAE=37.331 Accuracy=75.822%
SARIMA(1, 0, 1)(2, 1, 2, 12) MAE=37.923 Accuracy=74.917%
SARIMA(1, 0, 2)(0, 0, 0, 12) MAE=101.961 Accuracy=45.770%
SARIMA(1, 0, 2)(0, 0, 1, 12) MAE=92.922 Accuracy=49.430%
SARIMA(1, 0, 2)(0, 0, 2, 12) MAE=78.704 Accuracy=57.664%
SARIMA(1, 0, 2)(0, 1, 0, 12) MAE=64.174 Accuracy=60.693%
SARIMA(1, 0, 2)(0, 1, 1, 12) MAE=34.361 Accuracy=76.518%
SARIMA(1, 0, 2)(0, 1, 2, 12) MAE=34.820 Accuracy=76.241%
SARIMA(1, 0, 2)(1, 0, 0, 12) MAE=80.794 Accuracy=56.214%
SARIMA(1, 0, 2)(1, 0, 1, 12) MAE=35.037 Accuracy=78.491%
SARIMA(1, 0, 2)(1, 0, 2, 12) MAE=35.914 Accuracy=77.741%
SARIMA(1, 0, 2)(1, 1, 0, 12) MAE=70.193 Accuracy=55.878%
SARIMA(1, 0, 2)(1, 1, 1, 12) MAE=34.534 Accuracy=76.389%
SARIMA(1, 0, 2)(1, 1, 2, 12) MAE=34.380 Accuracy=76.651%
SARIMA(1, 0, 2)(2, 0, 0, 12) MAE=52.072 Accuracy=72.486%
SARIMA(1, 0, 2)(2, 0, 1, 12) MAE=35.614 Accuracy=77.907%
SARIMA(1, 0, 2)(2, 0, 2, 12) MAE=35.206 Accuracy=78.870%
SARIMA(1, 0, 2)(2, 1, 0, 12) MAE=40.171 Accuracy=73.587%
SARIMA(1, 0, 2)(2, 1, 1, 12) MAE=38.095 Accuracy=74.912%
SARIMA(1, 0, 2)(2, 1, 2, 12) MAE=37.938 Accuracy=74.934%
SARIMA(1, 1, 0)(0, 0, 0, 12) MAE=67.959 Accuracy=63.854%
SARIMA(1, 1, 0)(0, 0, 1, 12) MAE=59.842 Accuracy=68.257%
SARIMA(1, 1, 0)(0, 0, 2, 12) MAE=47.789 Accuracy=74.222%
SARIMA(1, 1, 0)(0, 1, 0, 12) MAE=43.659 Accuracy=73.340%
SARIMA(1, 1, 0)(0, 1, 1, 12) MAE=39.721 Accuracy=71.673%
SARIMA(1, 1, 0)(0, 1, 2, 12) MAE=40.513 Accuracy=71.105%
```

```
SARIMA(1, 1, 0)(1, 0, 0, 12) MAE=50.548 Accuracy=73.089%
SARIMA(1, 1, 0)(1, 0, 1, 12) MAE=34.937 Accuracy=76.477%
SARIMA(1, 1, 0)(1, 0, 2, 12) MAE=36.019 Accuracy=75.644%
SARIMA(1, 1, 0)(1, 1, 0, 12) MAE=63.902 Accuracy=60.234%
SARIMA(1, 1, 0)(1, 1, 1, 12) MAE=40.318 Accuracy=71.244%
SARIMA(1, 1, 0)(1, 1, 2, 12) MAE=37.798 Accuracy=73.548%
SARIMA(1, 1, 0)(2, 0, 0, 12) MAE=36.687 Accuracy=79.194%
SARIMA(1, 1, 0)(2, 0, 1, 12) MAE=35.657 Accuracy=75.909%
SARIMA(1, 1, 0)(2, 1, 0, 12) MAE=36.058 Accuracy=78.758%
SARIMA(1, 1, 0)(2, 1, 1, 12) MAE=37.852 Accuracy=74.048%
SARIMA(1, 1, 0)(2, 1, 2, 12) MAE=38.216 Accuracy=73.596%
SARIMA(1, 1, 1)(0, 0, 0, 12) MAE=69.036 Accuracy=63.350%
SARIMA(1, 1, 1)(0, 0, 1, 12) MAE=60.610 Accuracy=67.837%
SARIMA(1, 1, 1)(0, 0, 2, 12) MAE=48.557 Accuracy=73.852%
SARIMA(1, 1, 1)(0, 1, 0, 12) MAE=43.235 Accuracy=73.372%
SARIMA(1, 1, 1)(0, 1, 1, 12) MAE=41.574 Accuracy=70.138%
SARIMA(1, 1, 1)(0, 1, 2, 12) MAE=44.007 Accuracy=68.344%
SARIMA(1, 1, 1)(1, 0, 0, 12) MAE=51.077 Accuracy=72.826%
SARIMA(1, 1, 1)(1, 0, 1, 12) MAE=56.244 Accuracy=57.297%
SARIMA(1, 1, 1)(1, 0, 2, 12) MAE=54.799 Accuracy=58.673%
SARIMA(1, 1, 1)(1, 1, 0, 12) MAE=63.344 Accuracy=60.592%
SARIMA(1, 1, 1)(1, 1, 1, 12) MAE=43.466 Accuracy=68.729%
SARIMA(1, 1, 1)(1, 1, 2, 12) MAE=39.971 Accuracy=71.700%
SARIMA(1, 1, 1)(2, 0, 0, 12) MAE=35.856 Accuracy=79.513%
SARIMA(1, 1, 1)(2, 0, 1, 12) MAE=55.884 Accuracy=57.638%
SARIMA(1, 1, 1)(2, 0, 2, 12) MAE=35.135 Accuracy=75.943%
SARIMA(1, 1, 1)(2, 1, 0, 12) MAE=37.978 Accuracy=77.785%
SARIMA(1, 1, 1)(2, 1, 1, 12) MAE=38.177 Accuracy=74.044%
SARIMA(1, 1, 1)(2, 1, 2, 12) MAE=41.610 Accuracy=70.983%
SARIMA(1, 1, 2)(0, 0, 0, 12) MAE=72.992 Accuracy=61.580%
SARIMA(1, 1, 2)(0, 0, 1, 12) MAE=64.213 Accuracy=65.935%
SARIMA(1, 1, 2)(0, 0, 2, 12) MAE=52.188 Accuracy=72.157%
SARIMA(1, 1, 2)(0, 1, 0, 12) MAE=43.242 Accuracy=73.367%
SARIMA(1, 1, 2)(0, 1, 1, 12) MAE=42.077 Accuracy=69.771%
SARIMA(1, 1, 2)(0, 1, 2, 12) MAE=43.624 Accuracy=68.632%
SARIMA(1, 1, 2)(1, 0, 0, 12) MAE=54.165 Accuracy=71.235%
SARIMA(1, 1, 2)(1, 0, 1, 12) MAE=35.592 Accuracy=75.407%
SARIMA(1, 1, 2)(1, 0, 2, 12) MAE=37.605 Accuracy=73.862%
SARIMA(1, 1, 2)(1, 1, 0, 12) MAE=85.397 Accuracy=45.656%
SARIMA(1, 1, 2)(1, 1, 1, 12) MAE=43.135 Accuracy=68.977%
SARIMA(1, 1, 2)(1, 1, 2, 12) MAE=40.437 Accuracy=71.335%
SARIMA(1, 1, 2)(2, 0, 0, 12) MAE=36.968 Accuracy=79.112%
SARIMA(1, 1, 2)(2, 0, 1, 12) MAE=40.667 Accuracy=71.600%
SARIMA(1, 1, 2)(2, 0, 2, 12) MAE=36.003 Accuracy=75.072%
SARIMA(1, 1, 2)(2, 1, 0, 12) MAE=38.102 Accuracy=77.762%
SARIMA(1, 1, 2)(2, 1, 1, 12) MAE=41.868 Accuracy=70.608%
SARIMA(1, 1, 2)(2, 1, 2, 12) MAE=41.851 Accuracy=70.875%
SARIMA(2, 0, 0)(0, 0, 0, 12) MAE=90.498 Accuracy=52.234%
SARIMA(2, 0, 0)(0, 0, 1, 12) MAE=81.147 Accuracy=56.829%
SARIMA(2, 0, 0)(0, 0, 2, 12) MAE=67.565 Accuracy=63.854%
SARIMA(2, 0, 0)(0, 1, 0, 12) MAE=61.786 Accuracy=62.153%
SARIMA(2, 0, 0)(0, 1, 1, 12) MAE=34.197 Accuracy=76.414%
SARIMA(2, 0, 0)(0, 1, 2, 12) MAE=34.092 Accuracy=76.477%
SARIMA(2, 0, 0)(1, 0, 0, 12) MAE=69.981 Accuracy=62.704%
SARIMA(2, 0, 0)(1, 0, 1, 12) MAE=35.017 Accuracy=78.555%
SARIMA(2, 0, 0)(1, 0, 2, 12) MAE=35.669 Accuracy=78.152%
```

```
SARIMA(2, 0, 0)(1, 1, 0, 12) MAE=70.460 Accuracy=55.713%
SARIMA(2, 0, 0)(1, 1, 1, 12) MAE=34.153 Accuracy=76.420%
SARIMA(2, 0, 0)(1, 1, 2, 12) MAE=34.061 Accuracy=76.996%
SARIMA(2, 0, 0)(2, 0, 0, 12) MAE=46.724 Accuracy=75.218%
SARIMA(2, 0, 0)(2, 0, 1, 12) MAE=35.535 Accuracy=78.227%
SARIMA(2, 0, 0)(2, 0, 2, 12) MAE=35.338 Accuracy=78.968%
SARIMA(2, 0, 0)(2, 1, 0, 12) MAE=40.824 Accuracy=72.953%
SARIMA(2, 0, 0)(2, 1, 1, 12) MAE=36.000 Accuracy=75.122%
SARIMA(2, 0, 0)(2, 1, 2, 12) MAE=35.670 Accuracy=76.529%
SARIMA(2, 0, 1)(0, 0, 0, 12) MAE=92.478 Accuracy=51.181%
SARIMA(2, 0, 1)(0, 0, 1, 12) MAE=82.666 Accuracy=56.007%
SARIMA(2, 0, 1)(0, 0, 2, 12) MAE=69.248 Accuracy=62.961%
SARIMA(2, 0, 1)(0, 1, 0, 12) MAE=61.543 Accuracy=62.304%
SARIMA(2, 0, 1)(0, 1, 1, 12) MAE=34.288 Accuracy=76.544%
SARIMA(2, 0, 1)(0, 1, 2, 12) MAE=34.675 Accuracy=76.310%
SARIMA(2, 0, 1)(1, 0, 0, 12) MAE=71.309 Accuracy=61.989%
SARIMA(2, 0, 1)(1, 0, 1, 12) MAE=34.731 Accuracy=78.548%
SARIMA(2, 0, 1)(1, 0, 2, 12) MAE=35.613 Accuracy=77.855%
SARIMA(2, 0, 1)(1, 1, 0, 12) MAE=69.991 Accuracy=56.002%
SARIMA(2, 0, 1)(1, 1, 1, 12) MAE=36.389 Accuracy=74.700%
SARIMA(2, 0, 1)(1, 1, 2, 12) MAE=34.280 Accuracy=76.575%
SARIMA(2, 0, 1)(2, 0, 0, 12) MAE=46.395 Accuracy=75.385%
SARIMA(2, 0, 1)(2, 0, 1, 12) MAE=35.333 Accuracy=78.054%
SARIMA(2, 0, 1)(2, 0, 2, 12) MAE=34.961 Accuracy=78.943%
SARIMA(2, 0, 1)(2, 1, 0, 12) MAE=39.972 Accuracy=73.774%
SARIMA(2, 0, 1)(2, 1, 1, 12) MAE=38.153 Accuracy=75.036%
SARIMA(2, 0, 1)(2, 1, 2, 12) MAE=38.022 Accuracy=75.064%
SARIMA(2, 0, 2)(0, 0, 0, 12) MAE=101.849 Accuracy=45.843%
SARIMA(2, 0, 2)(0, 0, 1, 12) MAE=90.633 Accuracy=50.868%
SARIMA(2, 0, 2)(0, 0, 2, 12) MAE=76.890 Accuracy=58.809%
SARIMA(2, 0, 2)(0, 1, 0, 12) MAE=63.987 Accuracy=60.802%
SARIMA(2, 0, 2)(0, 1, 1, 12) MAE=34.229 Accuracy=76.597%
SARIMA(2, 0, 2)(0, 1, 2, 12) MAE=34.637 Accuracy=76.325%
SARIMA(2, 0, 2)(1, 0, 0, 12) MAE=78.127 Accuracy=57.867%
SARIMA(2, 0, 2)(1, 0, 1, 12) MAE=34.817 Accuracy=78.544%
SARIMA(2, 0, 2)(1, 0, 2, 12) MAE=35.603 Accuracy=77.853%
SARIMA(2, 0, 2)(1, 1, 0, 12) MAE=64.383 Accuracy=59.597%
SARIMA(2, 0, 2)(1, 1, 1, 12) MAE=34.479 Accuracy=76.471%
SARIMA(2, 0, 2)(1, 1, 2, 12) MAE=35.482 Accuracy=75.807%
SARIMA(2, 0, 2)(2, 0, 0, 12) MAE=50.969 Accuracy=73.059%
SARIMA(2, 0, 2)(2, 0, 1, 12) MAE=35.349 Accuracy=78.052%
SARIMA(2, 0, 2)(2, 0, 2, 12) MAE=38.101 Accuracy=76.096%
SARIMA(2, 0, 2)(2, 1, 0, 12) MAE=39.982 Accuracy=73.763%
SARIMA(2, 0, 2)(2, 1, 1, 12) MAE=37.345 Accuracy=75.998%
SARIMA(2, 0, 2)(2, 1, 2, 12) MAE=38.295 Accuracy=74.465%
SARIMA(2, 1, 0)(0, 0, 0, 12) MAE=70.161 Accuracy=62.836%
SARIMA(2, 1, 0)(0, 0, 1, 12) MAE=61.126 Accuracy=67.560%
SARIMA(2, 1, 0)(0, 0, 2, 12) MAE=48.900 Accuracy=73.688%
SARIMA(2, 1, 0)(0, 1, 0, 12) MAE=43.189 Accuracy=73.392%
SARIMA(2, 1, 0)(0, 1, 1, 12) MAE=40.454 Accuracy=71.166%
SARIMA(2, 1, 0)(0, 1, 2, 12) MAE=42.021 Accuracy=70.045%
SARIMA(2, 1, 0)(1, 0, 0, 12) MAE=51.315 Accuracy=72.710%
SARIMA(2, 1, 0)(1, 0, 1, 12) MAE=35.126 Accuracy=76.043%
SARIMA(2, 1, 0)(1, 0, 2, 12) MAE=37.019 Accuracy=74.629%
SARIMA(2, 1, 0)(1, 1, 0, 12) MAE=62.596 Accuracy=61.114%
SARIMA(2, 1, 0)(1, 1, 1, 12) MAE=41.683 Accuracy=70.278%
```

```
SARIMA(2, 1, 0)(1, 1, 2, 12) MAE=38.658 Accuracy=72.912%
SARIMA(2, 1, 0)(2, 0, 0, 12) MAE=35.734 Accuracy=79.562%
SARIMA(2, 1, 0)(2, 0, 1, 12) MAE=36.478 Accuracy=75.040%
SARIMA(2, 1, 0)(2, 0, 2, 12) MAE=35.009 Accuracy=76.504%
SARIMA(2, 1, 0)(2, 1, 0, 12) MAE=37.919 Accuracy=77.813%
SARIMA(2, 1, 0)(2, 1, 1, 12) MAE=39.032 Accuracy=73.067%
SARIMA(2, 1, 0)(2, 1, 2, 12) MAE=39.539 Accuracy=72.744%
SARIMA(2, 1, 1)(0, 0, 0, 12) MAE=52.244 Accuracy=61.691%
SARIMA(2, 1, 1)(0, 0, 1, 12) MAE=41.667 Accuracy=69.117%
SARIMA(2, 1, 1)(0, 0, 2, 12) MAE=41.397 Accuracy=68.388%
SARIMA(2, 1, 1)(0, 1, 0, 12) MAE=42.112 Accuracy=73.143%
SARIMA(2, 1, 1)(0, 1, 1, 12) MAE=63.094 Accuracy=54.187%
SARIMA(2, 1, 1)(0, 1, 2, 12) MAE=62.580 Accuracy=54.402%
SARIMA(2, 1, 1)(1, 0, 0, 12) MAE=38.860 Accuracy=70.626%
SARIMA(2, 1, 1)(1, 0, 1, 12) MAE=35.909 Accuracy=74.733%
SARIMA(2, 1, 1)(1, 0, 2, 12) MAE=35.883 Accuracy=74.900%
SARIMA(2, 1, 1)(1, 1, 0, 12) MAE=87.941 Accuracy=44.105%
SARIMA(2, 1, 1)(1, 1, 1, 12) MAE=62.736 Accuracy=54.326%
SARIMA(2, 1, 1)(1, 1, 2, 12) MAE=60.319 Accuracy=56.109%
SARIMA(2, 1, 1)(2, 0, 0, 12) MAE=42.015 Accuracy=68.032%
SARIMA(2, 1, 1)(2, 0, 1, 12) MAE=38.692 Accuracy=72.378%
SARIMA(2, 1, 1)(2, 0, 2, 12) MAE=39.004 Accuracy=72.295%
SARIMA(2, 1, 1)(2, 1, 0, 12) MAE=41.682 Accuracy=72.328%
SARIMA(2, 1, 1)(2, 1, 1, 12) MAE=52.733 Accuracy=60.887%
SARIMA(2, 1, 1)(2, 1, 2, 12) MAE=56.651 Accuracy=58.455%
SARIMA(2, 1, 2)(0, 0, 0, 12) MAE=52.324 Accuracy=61.463%
SARIMA(2, 1, 2)(0, 0, 1, 12) MAE=39.281 Accuracy=72.066%
SARIMA(2, 1, 2)(0, 0, 2, 12) MAE=37.779 Accuracy=72.062%
SARIMA(2, 1, 2)(0, 1, 0, 12) MAE=70.340 Accuracy=56.068%
SARIMA(2, 1, 2)(0, 1, 1, 12) MAE=63.187 Accuracy=54.011%
SARIMA(2, 1, 2)(0, 1, 2, 12) MAE=62.979 Accuracy=53.986%
SARIMA(2, 1, 2)(1, 0, 0, 12) MAE=36.862 Accuracy=73.163%
SARIMA(2, 1, 2)(1, 0, 1, 12) MAE=41.608 Accuracy=69.873%
SARIMA(2, 1, 2)(1, 0, 2, 12) MAE=36.037 Accuracy=74.725%
SARIMA(2, 1, 2)(1, 1, 0, 12) MAE=86.139 Accuracy=45.203%
SARIMA(2, 1, 2)(1, 1, 1, 12) MAE=63.591 Accuracy=53.622%
SARIMA(2, 1, 2)(1, 1, 2, 12) MAE=72.143 Accuracy=47.955%
SARIMA(2, 1, 2)(2, 0, 0, 12) MAE=37.285 Accuracy=72.036%
SARIMA(2, 1, 2)(2, 0, 1, 12) MAE=37.106 Accuracy=74.178%
SARIMA(2, 1, 2)(2, 1, 0, 12) MAE=43.813 Accuracy=70.738%
SARIMA(2, 1, 2)(2, 1, 1, 12) MAE=58.421 Accuracy=56.297%
SARIMA(2, 1, 2)(2, 1, 2, 12) MAE=65.528 Accuracy=51.423%
Best SARIMA((0, 0, 2), (2, 0, 1, 12)) MAE=30.285 Accuracy=82.058%
```

In [36]:

```python
# Creating a DataFrame for the forecast
forecast_index = pd.date_range(start=train.index[-1], periods=forecast_steps
forecast_df = pd.DataFrame({'Forecast': forecast}, index=forecast_index)
```

## Forecast for Test data

In [37]:

```python
forecast_df
```

Out[37]:

| | Forecast |
|---|---|
| **2023-01-31** | 183.650401 |
| **2023-02-28** | 121.343062 |
| **2023-03-31** | 239.249995 |
| **2023-04-30** | 256.120274 |
| **2023-05-31** | 290.327093 |
| **2023-06-30** | 249.514476 |
| **2023-07-31** | 205.457528 |
| **2023-08-31** | 178.095600 |
| **2023-09-30** | 151.381112 |
| **2023-10-31** | 130.912313 |
| **2023-11-30** | 145.821805 |
| **2023-12-31** | 76.109053 |
| **2024-01-31** | 144.975830 |
| **2024-02-29** | 95.779130 |
| **2024-03-31** | 188.869219 |
| **2024-04-30** | 202.192260 |
| **2024-05-31** | 229.181115 |
| **2024-06-30** | 196.951613 |
| **2024-07-31** | 162.161675 |
| **2024-08-31** | 140.566812 |
| **2024-09-30** | 119.482731 |
| **2024-10-31** | 103.316958 |
| **2024-11-30** | 115.098102 |
| **2024-12-31** | 60.059244 |

## Evaluating the model on the test set

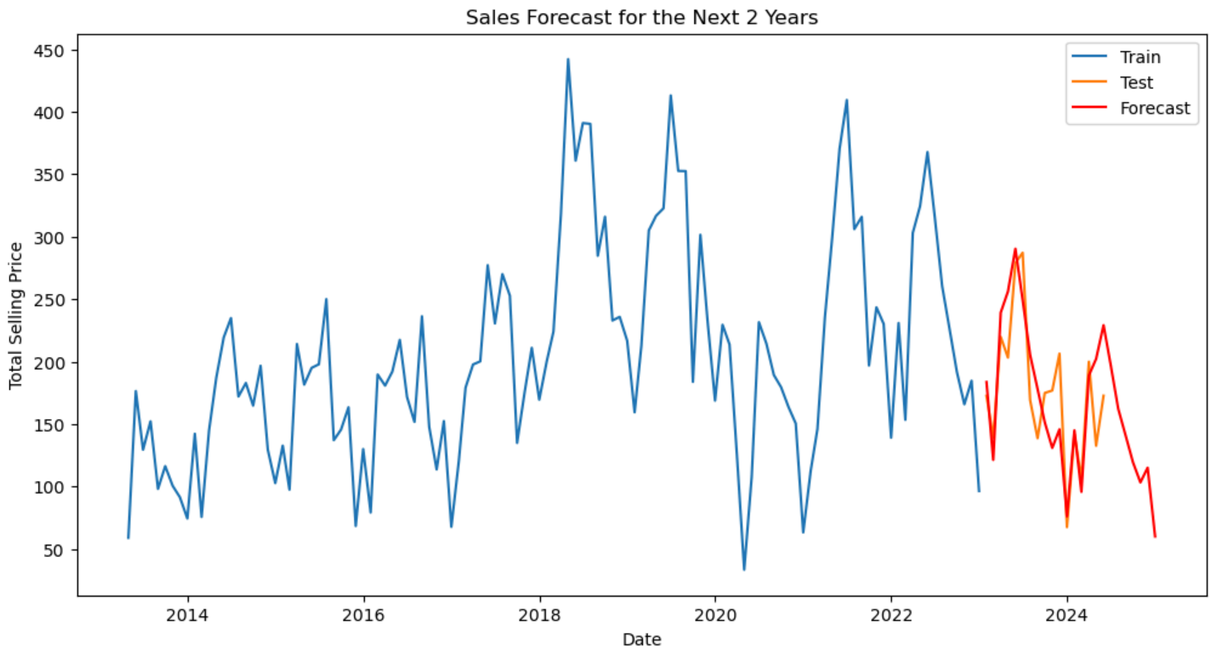
In [39]:
```
aligned_test = test.loc[test.index.intersection(forecast_df.index)]
aligned_forecast = forecast_df.loc[aligned_test.index]

mae_test = mean_absolute_error(aligned_test, aligned_forecast['Forecast'])
print(f'Mean Absolute Error (MAE): {mae_test}')

accuracy_test = cal_accuracy(aligned_test, aligned_forecast['Forecast'])
print(f'Accuracy for test: {accuracy_test}')
```

```
Mean Absolute Error (MAE): 30.285349469748947
Accuracy for test: 82.05759002050891
```

In [40]:
```
# Plot the forecast
plt.figure(figsize=(12, 6))
plt.plot(train.index, train, label='Train')
plt.plot(test.index, test, label='Test')
plt.plot(forecast_df.index, forecast_df['Forecast'], label='Forecast', color
plt.title('Sales Forecast for the Next 2 Years')
plt.xlabel('Date')
plt.ylabel('Total Selling Price')
plt.legend()
plt.show()
```




## Predicting the total amount of sales in the next 2 years for the Company as a whole

In [41]:
```
# Using the best SARIMA model from the previous grid search
best_order = (0, 0, 2)
best_seasonal_order = (1, 0, 1, 12)
```

In [42]:
```
# Training the SARIMA model
model = SARIMAX(train, order=best_order, seasonal_order=best_seasonal_order)
model_fit = model.fit(disp=False)
```

In [43]:
```
# Forecasting the next 2 years beyond the current data, which includes the t
additional_forecast_steps = 24
```

In [44]:
```
# Refit the model using all available data to forecast beyond the test peric
model_full = SARIMAX(monthly_sales, order=best_order, seasonal_order=best_se
model_full_fit = model_full.fit(disp=False)
```

In [45]:
```
# Forecast the next 2 years - including test period and additional 2 years
forecast_steps = additional_forecast_steps
full_forecast = model_full_fit.forecast(steps=forecast_steps)
```

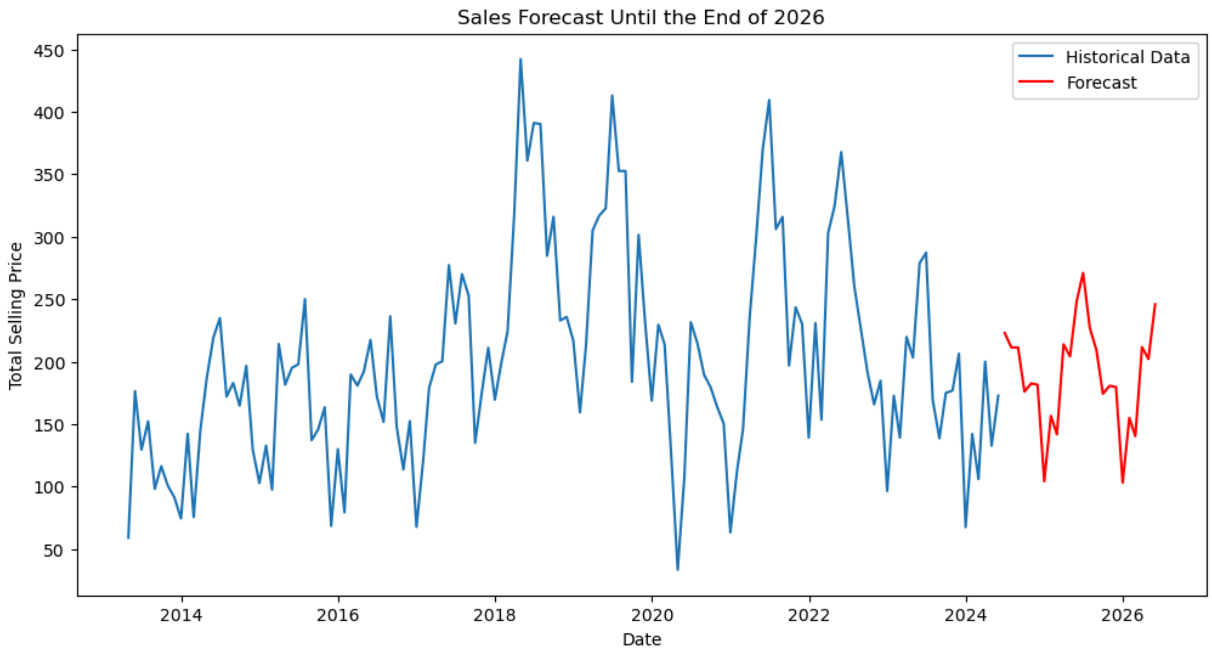
In [46]:
```
# Create a DataFrame for the forecast
forecast_index = pd.date_range(start=monthly_sales.index[-1], periods=foreca
new_forecast_df = pd.DataFrame({'Forecast': full_forecast[-additional_foreca
```

In [47]:
```
new_forecast_df
```

Out[47]:

| | Forecast |
|---|---|
| **2024-06-30** | 222.985941 |
| **2024-07-31** | 211.308247 |
| **2024-08-31** | 211.348079 |
| **2024-09-30** | 176.026202 |
| **2024-10-31** | 182.507709 |
| **2024-11-30** | 181.521740 |
| **2024-12-31** | 104.192347 |
| **2025-01-31** | 156.601193 |
| **2025-02-28** | 141.766598 |
| **2025-03-31** | 213.792764 |
| **2025-04-30** | 204.215675 |
| **2025-05-31** | 248.446325 |
| **2025-06-30** | 271.017989 |
| **2025-07-31** | 226.939837 |
| **2025-08-31** | 209.190965 |
| **2025-09-30** | 174.229599 |
| **2025-10-31** | 180.644953 |
| **2025-11-30** | 179.669047 |
| **2025-12-31** | 103.128913 |
| **2026-01-31** | 155.002850 |
| **2026-02-28** | 140.319664 |
| **2026-03-31** | 211.610699 |
| **2026-04-30** | 202.131358 |
| **2026-05-31** | 245.910569 |

In [52]:
```
# Plotting the forecast for the company as a whole
plt.figure(figsize=(12, 6))
plt.plot(monthly_sales.index, monthly_sales, label='Historical Data')
plt.plot(new_forecast_df.index, new_forecast_df['Forecast'], label='Forecast
plt.title('Sales Forecast Until the End of 2026')
plt.xlabel('Date')
plt.ylabel('Total Selling Price')
plt.legend()
plt.show()
```




## Predict the total amount of sales in the next 2 years for each Product ID, each Company ID, and each Country

In [53]:
```
def forecast_sales_by_group(data, group_col, date_col, sales_col, forecast_s
    unique_groups = data[group_col].unique()
    all_forecasts = []

    for group in unique_groups:
        group_data = data[data[group_col] == group]

        # Aggregate sales data by month
        monthly_sales = group_data.resample('M', on=date_col)[sales_col].sum

        # Skip groups with insufficient data
        if len(monthly_sales) < 24:
            continue

        # Train-test split
        train_end_date = '2022-12-31'
        train = monthly_sales[:train_end_date]
        test = monthly_sales[train_end_date:]

        # Define the SARIMA model
        best_order = (0, 0, 2)
```

```
        best_seasonal_order = (1, 0, 1, 12)

        # Train the SARIMA model
        model = SARIMAX(train, order=best_order, seasonal_order=best_seasona
        model_fit = model.fit(disp=False)

        # Refit the model using all available data to forecast beyond the te
        model_full = SARIMAX(monthly_sales, order=best_order, seasonal_order
        model_full_fit = model_full.fit(disp=False)

        # Forecast the next 2 years
        forecast = model_full_fit.forecast(steps=forecast_steps)

        # Create a DataFrame for the forecast
        forecast_index = pd.date_range(start=monthly_sales.index[-1], perioc
        forecast_df = pd.DataFrame({'Group': group, 'Date': forecast_index,

        all_forecasts.append(forecast_df)

    # Combine all forecasts into a single DataFrame
    all_forecasts_df = pd.concat(all_forecasts)

    return all_forecasts_df
```

In [54]:
```
forecast_steps = 24
```

## Forecasting sales for each Product ID

In [55]:
```
product_forecasts = forecast_sales_by_group(data, 'Product ID', 'Order Date'
```

In [56]:
```
product_forecasts
```

Out[56]:

| | Group | Date | Forecast |
|---|---|---|---|
| **2024-03-31** | Prod_5030 | 2024-03-31 | 0.152870 |
| **2024-04-30** | Prod_5030 | 2024-04-30 | 0.113541 |
| **2024-05-31** | Prod_5030 | 2024-05-31 | 0.073331 |
| **2024-06-30** | Prod_5030 | 2024-06-30 | 0.057857 |
| **2024-07-31** | Prod_5030 | 2024-07-31 | 0.121690 |
| **...** | ... | ... | ... |
| **2025-12-31** | Prod_74003 | 2025-12-31 | -0.000556 |
| **2026-01-31** | Prod_74003 | 2026-01-31 | -0.002988 |
| **2026-02-28** | Prod_74003 | 2026-02-28 | -0.001122 |
| **2026-03-31** | Prod_74003 | 2026-03-31 | -0.005643 |
| **2026-04-30** | Prod_74003 | 2026-04-30 | -0.003912 |

5280 rows × 3 columns

## Forecasting sales for each Company ID

In [57]:
```
company_forecasts = forecast_sales_by_group(data, 'Company ID', 'Order Date'
```

In [58]:
```
company_forecasts
```

Out[58]:

| | Group | Date | Forecast |
|---|---|---|---|
| **2018-10-31** | Company_87239 | 2018-10-31 | 1.632470e-01 |
| **2018-11-30** | Company_87239 | 2018-11-30 | -3.754704e-02 |
| **2018-12-31** | Company_87239 | 2018-12-31 | 8.437542e-17 |
| **2019-01-31** | Company_87239 | 2019-01-31 | 1.097423e-16 |
| **2019-02-28** | Company_87239 | 2019-02-28 | -3.256720e-04 |
| **...** | ... | ... | ... |
| **2025-07-31** | Company_13782 | 2025-07-31 | 1.706155e-21 |
| **2025-08-31** | Company_13782 | 2025-08-31 | 7.678840e-11 |
| **2025-09-30** | Company_13782 | 2025-09-30 | 6.077018e-12 |
| **2025-10-31** | Company_13782 | 2025-10-31 | 1.318255e-10 |
| **2025-11-30** | Company_13782 | 2025-11-30 | 2.714007e-05 |

10272 rows × 3 columns

## Forecasting sales for each Country

In [59]:
```
country_forecasts = forecast_sales_by_group(data, 'Shipping Country', 'Order
```

In [60]:
```
country_forecasts
```

Out[60]:

| | Group | Date | Forecast |
|---|---|---|---|
| **2024-06-30** | United States | 2024-06-30 | 1.828996e+02 |
| **2024-07-31** | United States | 2024-07-31 | 8.183174e+01 |
| **2024-08-31** | United States | 2024-08-31 | 8.255626e+01 |
| **2024-09-30** | United States | 2024-09-30 | -2.097172e+00 |
| **2024-10-31** | United States | 2024-10-31 | 1.097324e+02 |
| **...** | ... | ... | ... |
| **2025-05-31** | Dominican Republic | 2025-05-31 | -3.147056e-12 |
| **2025-06-30** | Dominican Republic | 2025-06-30 | 3.151665e-25 |
| **2025-07-31** | Dominican Republic | 2025-07-31 | 4.549198e-26 |
| **2025-08-31** | Dominican Republic | 2025-08-31 | -9.486245e-13 |
| **2025-09-30** | Dominican Republic | 2025-09-30 | -2.582897e-12 |

216 rows × 3 columns