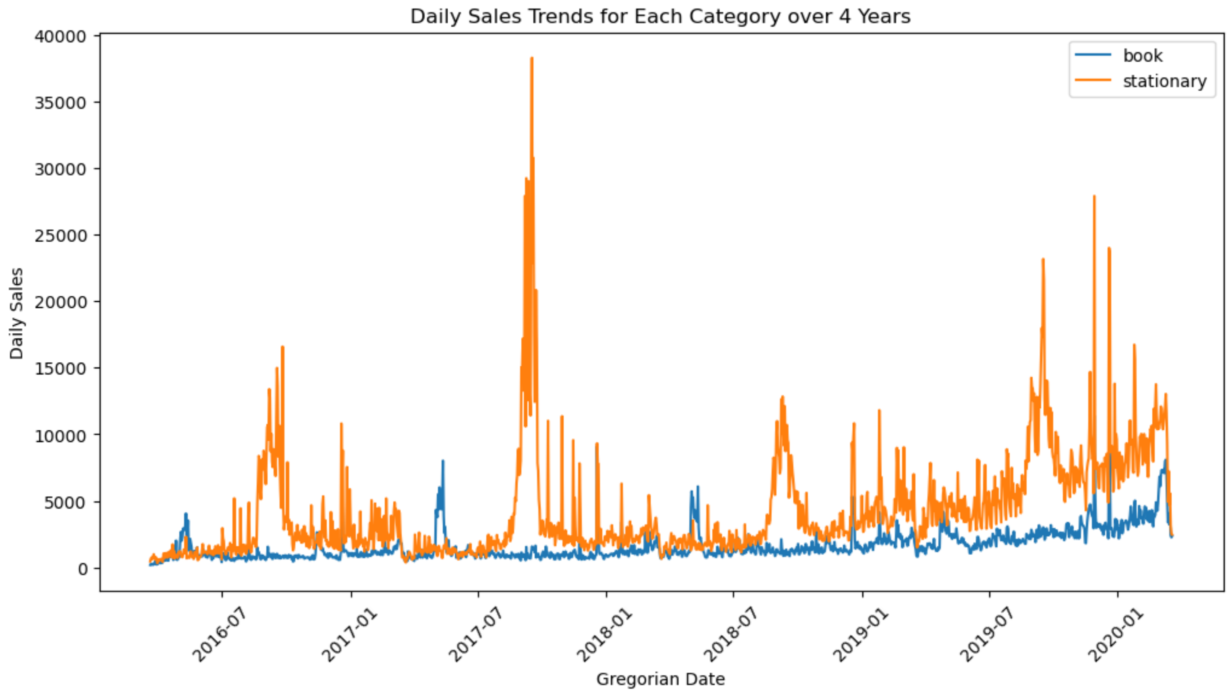
1) Evaluate the performance and trends of each category by exploring the dataset. Do you find any specific pattern or anomaly? Are there any notable changes or improvements over the 4-year period?

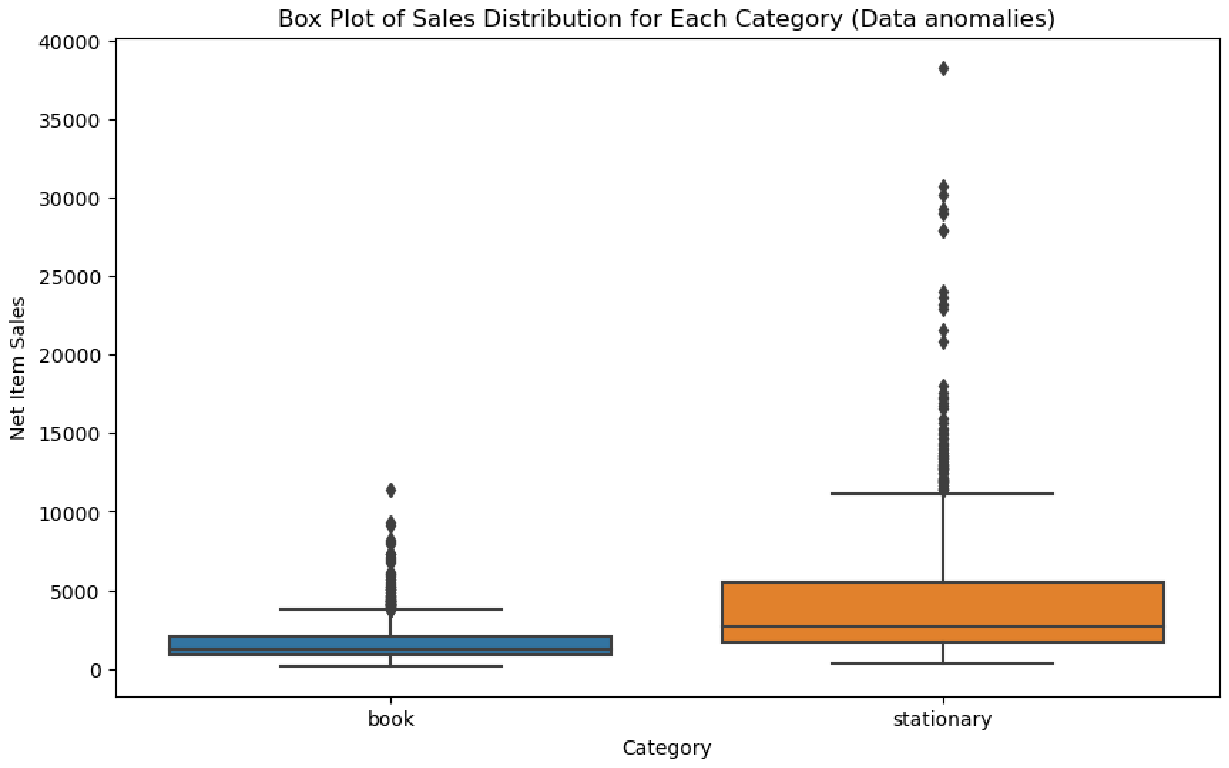
To analyze the performance and trends of each category ("book" and "stationary"), I would start by calculating key metrics such as average daily sales, total sales over the 4-year period, and identifying any anomalies. I would use tools and libraries like Pandas, numpy, statsmodels, and seaborn for data manipulation and analysis. Here's an outline of the approach: - Load the dataset and perform basic data cleaning. - Calculate average daily sales and total sales for each category. - Plot line charts showing daily sales trends over the 4-year period for each category. - Use statistical techniques to identify anomalies or outliers in the sales data.

In [1]:
```python
# Import required libraries
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import datetime as dt
from persiantools.jdatetime import JalaliDate
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
import seaborn as sns
import calendar


# Load the dataset
file_path = r'C:\Users\Asus\OneDrive\Desktop\Digikala task\Planning_Task.csv'
df = pd.read_csv(file_path)

# Rename the column
df.rename(columns={' Net Item Fcast ': 'Net Item Sales'}, inplace=True)

# Define a mapping between month names and month numbers
month_mapping = {
    'Farvardin': '01',
    'Ordibehesht': '02',
    'Khordad': '03',
    'Tir': '04',
    'Mordad': '05',
    'Shahrivar': '06',
    'Mehr': '07',
    'Aaban': '08',
    'Aazar': '09',
    'Dey': '10',
    'Bahman': '11',
    'Esfand': '12'
}

# Map month names to month numbers
df['Persian Month Number'] = df['Persian Month Name En'].map(month_mapping)

# Remove thousands separators (commas) from the 'Net Item Sales' column
df['Net Item Sales'] = df['Net Item Sales'].str.replace(',', '')

# Calculate daily sales for each category
df['Net Item Sales'] = pd.to_numeric(df['Net Item Sales'], errors='coerce')  # Convert

# Create a pseudo-date column
def convert_to_gregorian(row):
    jalali_date = JalaliDate(int(row['Persian Year']), int(row['Persian Month Number']
    return jalali_date.to_gregorian()

df['Gregorian Date'] = df.apply(convert_to_gregorian, axis=1)

df['Gregorian Date'] = pd.to_datetime(df['Gregorian Date'])  # Convert to datetime
# Extract month number and create a new column
df['Gregorian Month Number'] = df['Gregorian Date'].dt.month
# Extract month number and create a new column
df['Gregorian Year'] = df['Gregorian Date'].dt.year

# Calculate average daily sales and total sales for each category
category_sales = df.groupby('Category')['Net Item Sales'].sum()
category_days = df.groupby('Category')['Gregorian Date'].nunique()
category_avg_daily_sales = category_sales / category_days
total_sales_4_years = category_sales.sum()

# Visualize trends: Line charts for daily sales over 4 years
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x='Gregorian Date', y='Net Item Sales', hue='Category')
plt.xlabel('Gregorian Date')
plt.ylabel('Daily Sales')
plt.title('Daily Sales Trends for Each Category over 4 Years')
plt.legend()
plt.xticks(rotation=45,size=10)
plt.show()
```



In [2]:
```
# Set 'Gregorian Date' column as the index
df.set_index('Gregorian Date', inplace=True)  # Set as index

# Identify anomalies using z-score
def detect_anomalies(series):
    z_scores = np.abs((series - series.mean()) / series.std())
    threshold = 3
    anomalies = series[z_scores > threshold]
    return anomalies
```

```
anomalies_book = detect_anomalies(df[df['Category'] == 'book']['Net Item Sales'])
anomalies_stationary = detect_anomalies(df[df['Category'] == 'stationary']['Net Item S

# Visualize anomalies using box plots
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Category', y='Net Item Sales')
plt.title('Box Plot of Sales Distribution for Each Category (Data anomalies)')
plt.show()
```



In [3]:
```
# Summarize findings
summary_df1 = pd.DataFrame({'Average Daily Sales': category_avg_daily_sales})
print("Total Sales over 4 Years: {:.2f}".format(total_sales_4_years))
summary_df2 = pd.DataFrame({'Anomalies in "book" category': anomalies_book})
summary_df3 = pd.DataFrame({'Anomalies in "stationary" category': anomalies_stationary
display(summary_df1)
display(summary_df2)
display(summary_df3)
```

```
Total Sales over 4 Years: 8490104.00
```

	Average Daily Sales
Category	
book	1646.373288
stationary	4168.766438

	Anomalies in "book" category
Gregorian Date	
2016-12-20	5447
2017-05-06	5491
2017-05-08	6006
2017-05-12	5270
2017-05-13	8014
2017-12-19	9092
2017-12-20	5645
2018-05-03	5708
2018-05-05	5250
2018-05-12	6080
2018-12-18	5313
2018-12-21	5272
2019-11-29	11388
2019-11-30	5891
2019-12-20	9288
2019-12-21	8210
2020-02-29	5929
2020-03-01	6815
2020-03-02	6103
2020-03-03	6899
2020-03-04	7307
2020-03-05	7155
2020-03-06	7316
2020-03-07	7346
2020-03-08	7081
2020-03-09	7977
2020-03-10	8089
2020-03-11	6982

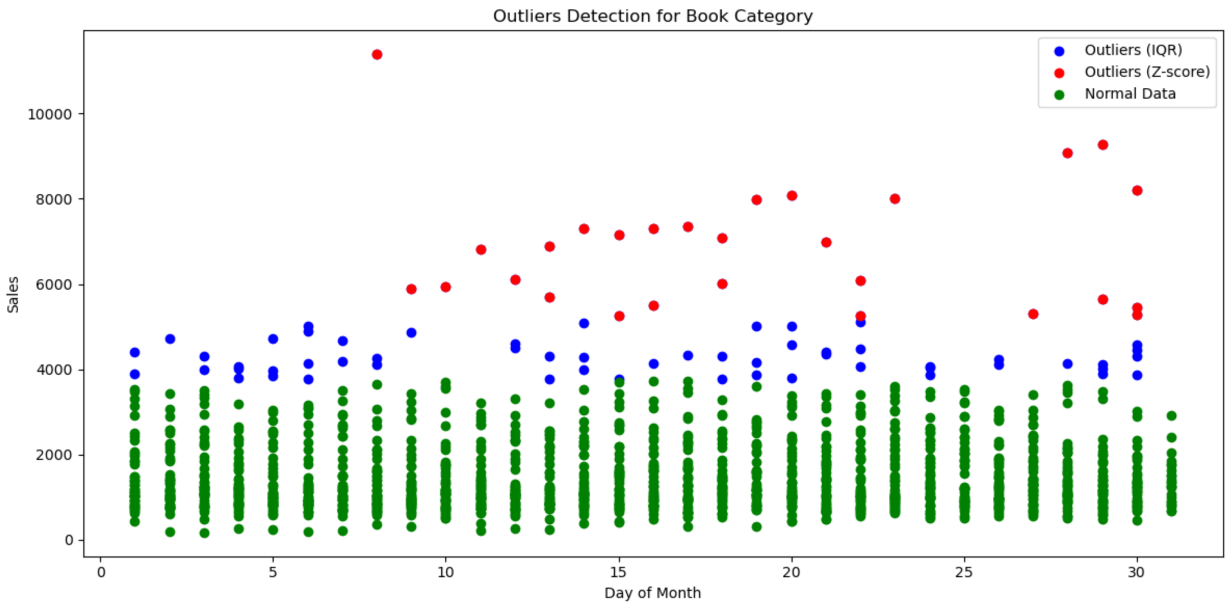
Anomalies in "stationary" category

Gregorian Date	
2016-09-26	16587
2017-09-04	17182
2017-09-07	27894
2017-09-09	29229
2017-09-12	29007
2017-09-16	30205
2017-09-17	38273
2017-09-18	22896
2017-09-19	30758
2017-09-20	17598
2017-09-23	20829
2017-09-24	16906
2019-09-14	15933
2019-09-15	17979
2019-09-16	17312
2019-09-17	23162
2019-09-18	21596
2019-11-29	27892
2019-12-20	23996
2019-12-21	23687
2020-01-25	16726

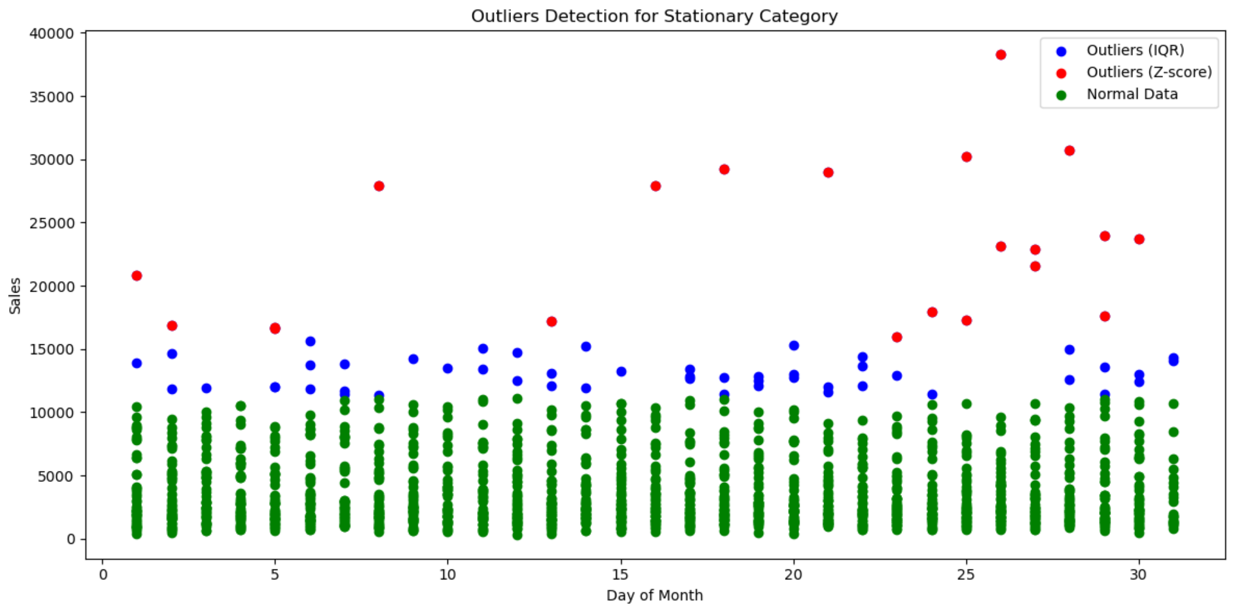
These anomalies could result from various factors such as special promotions, marketing campaigns, events, holidays, or even data recording errors. It's crucial to investigate each anomaly further to determine the underlying cause. These anomalies could potentially provide insights for optimizing inventory management, identifying successful marketing efforts, or addressing potential issues affecting sales performance.

In [4]:
```
# Group data by category, year, and quarter, and sum the quarterly sales
grouped_data = df.groupby(['Category', 'Persian Year', 'Persian Month Name En', 'Persi

# Separate data for each category
book_data = grouped_data[grouped_data['Category'] == 'book']
stationary_data = grouped_data[grouped_data['Category'] == 'stationary']

# Define a function to detect outliers using Z-score
def detect_outliers_zscore(data, threshold=3):
    z_scores = np.abs((data - np.mean(data)) / np.std(data))
    return z_scores > threshold

# Define a function to detect outliers using IQR
def detect_outliers_iqr(data, multiplier=1.5):
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    iqr = q3 - q1
    lower_bound = q1 - multiplier * iqr
    upper_bound = q3 + multiplier * iqr
    return (data < lower_bound) | (data > upper_bound)

# Detect outliers for 'book' category using Z-score
outliers_zscore_book = detect_outliers_zscore(book_data['Net Item Sales'])
outliers_zscore_stationary = detect_outliers_zscore(stationary_data['Net Item Sales']

# Detect outliers for 'stationary' category using IQR
outliers_iqr_book = detect_outliers_iqr(book_data['Net Item Sales'])
outliers_iqr_stationary = detect_outliers_iqr(stationary_data['Net Item Sales'])
```

In [5]:
```
# Plot outliers for 'book' category
plt.figure(figsize=(12, 6))
plt.scatter(book_data[outliers_iqr_book]['Persian Day Number Of Month'], book_data[out
plt.scatter(book_data[outliers_zscore_book]['Persian Day Number Of Month'], book_data[
plt.scatter(book_data[~outliers_zscore_book & ~outliers_iqr_book]['Persian Day Number 
plt.xlabel('Day of Month')
plt.ylabel('Sales')
plt.title('Outliers Detection for Book Category')
plt.legend()
plt.tight_layout()
plt.show()
```

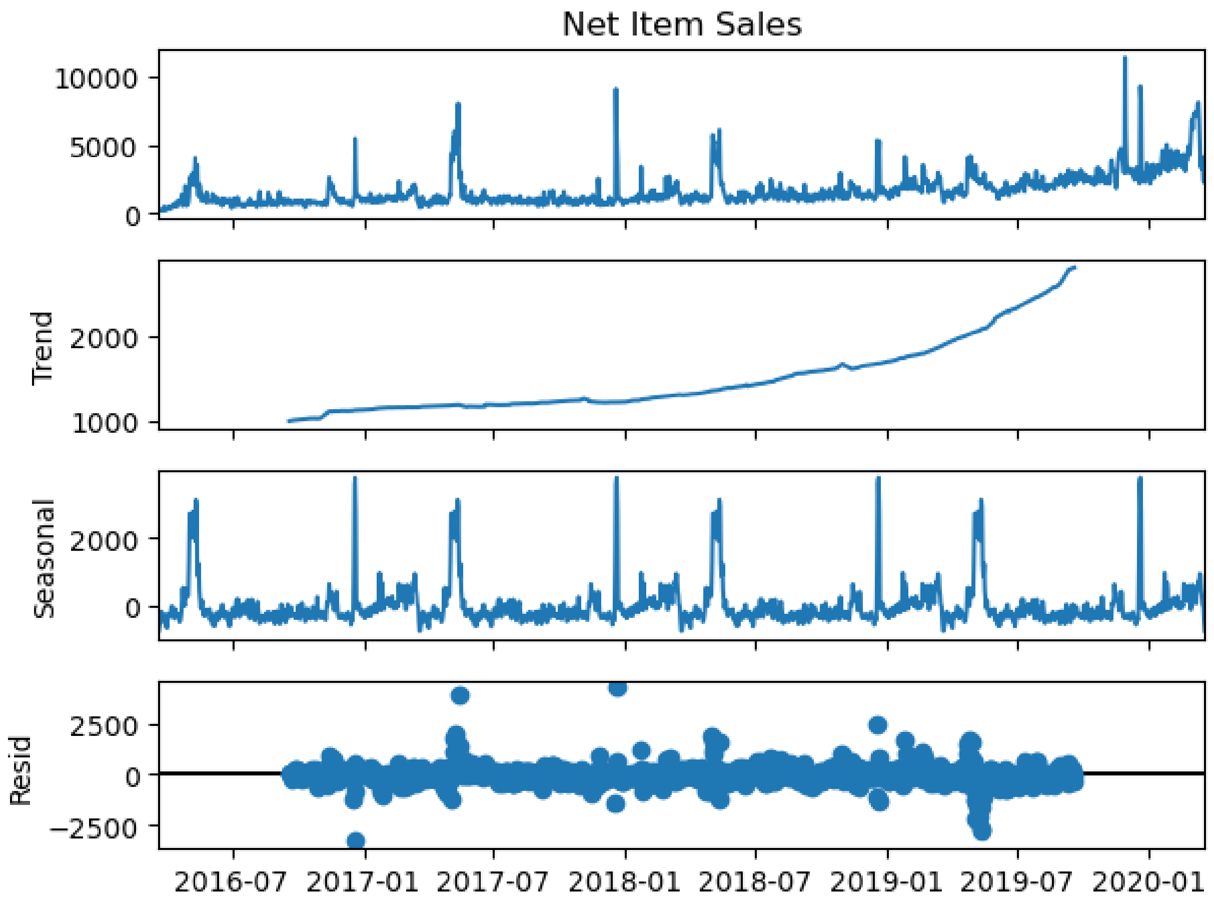


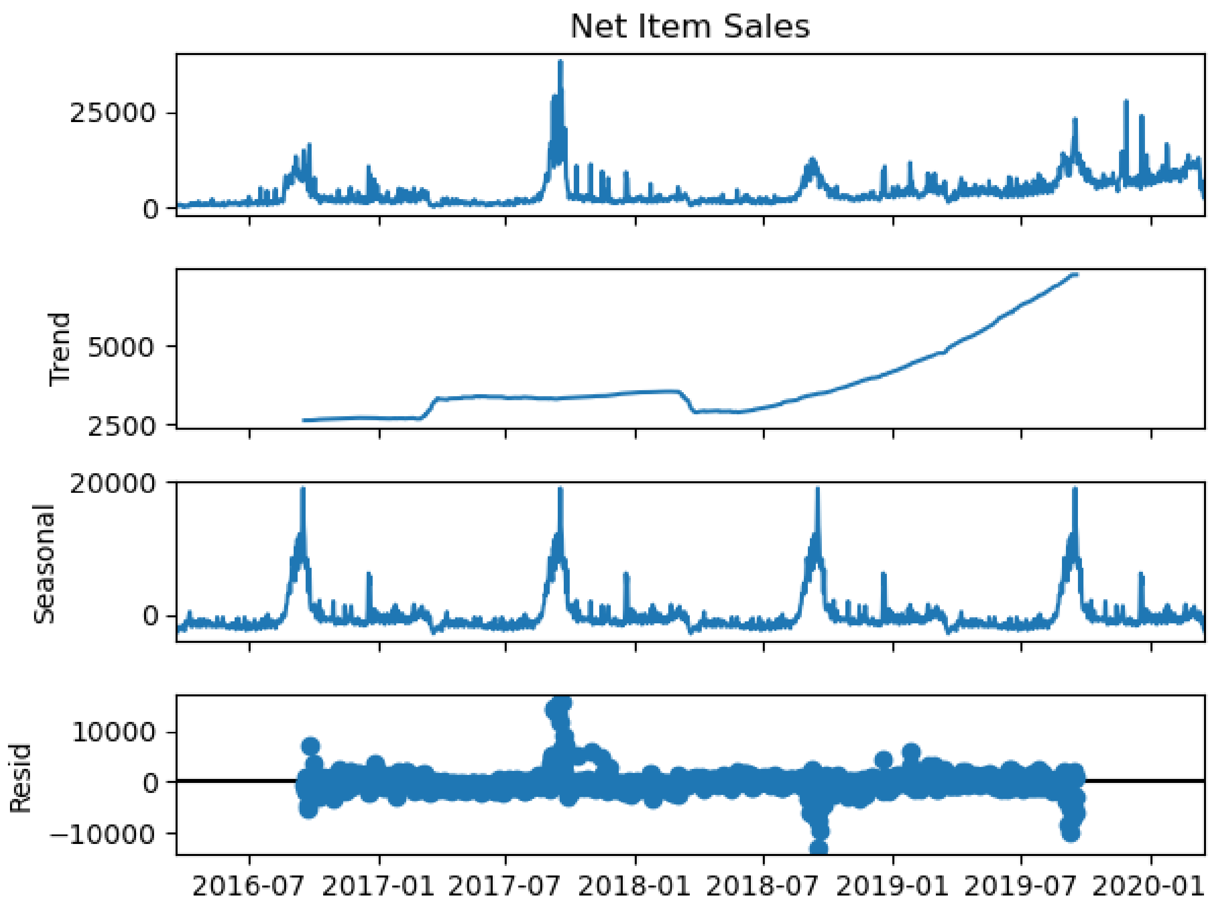
In [6]:
```
# Plot outliers for 'stationary' category
plt.figure(figsize=(12, 6))
plt.scatter(stationary_data[outliers_iqr_stationary]['Persian Day Number Of Month'], s
plt.scatter(stationary_data[outliers_zscore_stationary]['Persian Day Number Of Month']
plt.scatter(stationary_data[~outliers_zscore_stationary & ~outliers_iqr_stationary]['P
plt.xlabel('Day of Month')
plt.ylabel('Sales')
plt.title('Outliers Detection for Stationary Category')
plt.legend()
plt.tight_layout()
plt.show()
```



2) Apply statistical techniques or models to identify and analyze seasonality patterns in the data. Explain if there are any significant seasonal effects for each category.

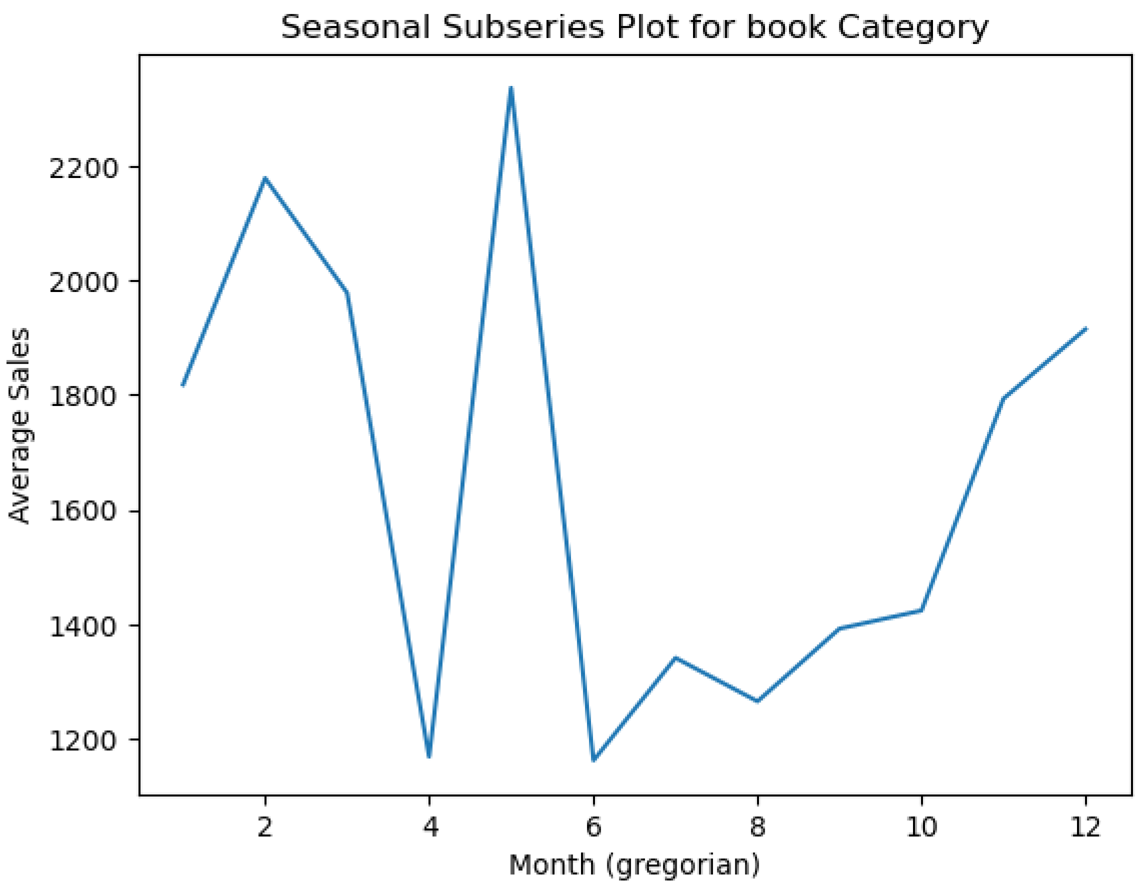
In [7]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the time series for each category
for category in df['Category'].unique():
    category_data = df[df['Category'] == category]['Net Item Sales']
    result = seasonal_decompose(category_data, model='additive', period=365)  # Adjust
    result.plot()
```

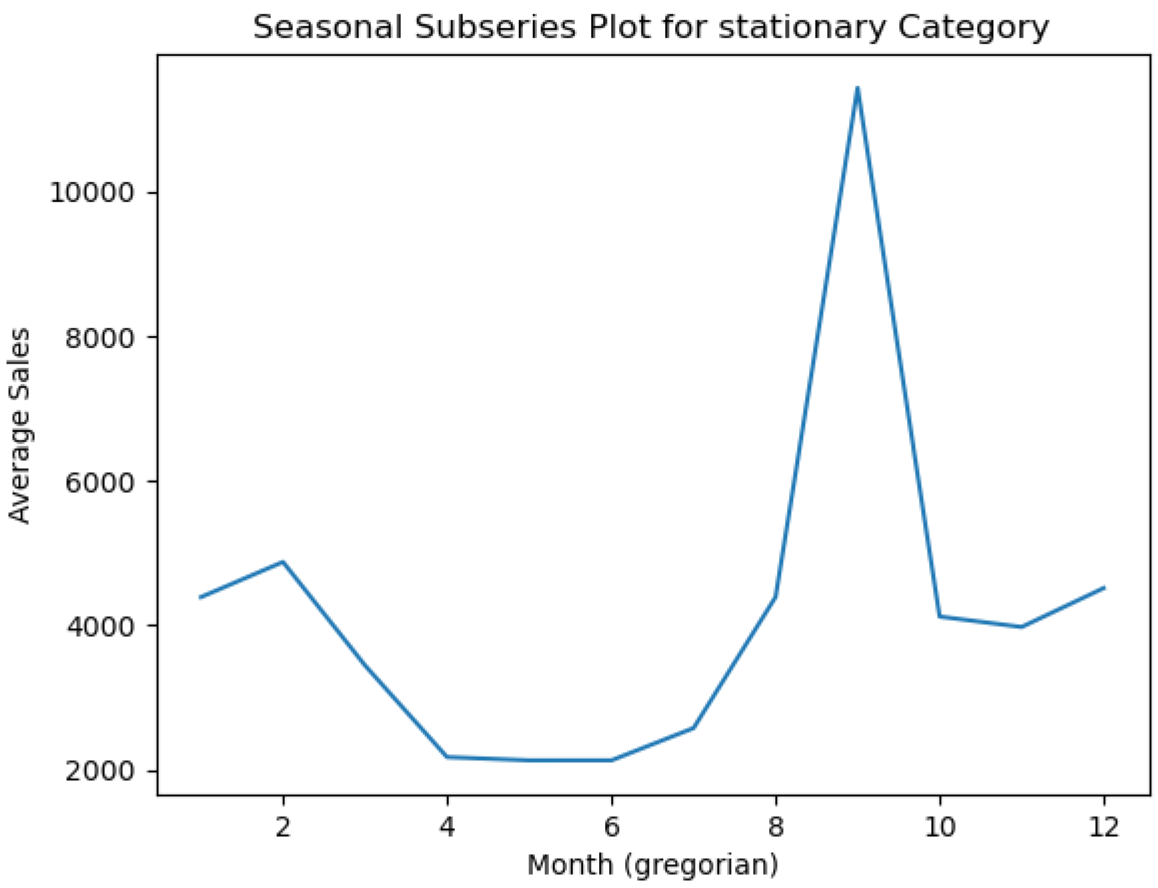




The analysis of sales patterns reveals distinct seasonal trends within the "book" and "stationery" categories over the course of each year. In the "book" category, a consistent seasonal pattern emerges, characterized by a subtle upturn in sales during February, followed by a noticeable decline in March. As the year progresses, a substantial peak in sales occurs during May, reflecting heightened consumer demand. Notably, a second, more concentrated peak in sales becomes evident in December. Turning to the "stationery" category, a similar yet distinct seasonal pattern is observed. Sales exhibit a decline during March, followed by a substantial peak in September, indicative of elevated purchasing activity. Furthermore, a smaller peak in sales is noticeable during December, contributing to the overall sales trend. These findings underscore the presence of significant seasonality in both categories. In the "book" category, the sales pattern highlights a dual-peak phenomenon, with May and December as prominent sales periods. Meanwhile, the "stationery" category demonstrates a seasonal trend characterized by a substantial September peak, with additional sales spikes in December and a corresponding dip in March. These insights provide valuable information for understanding and strategizing around the sales dynamics within each category.

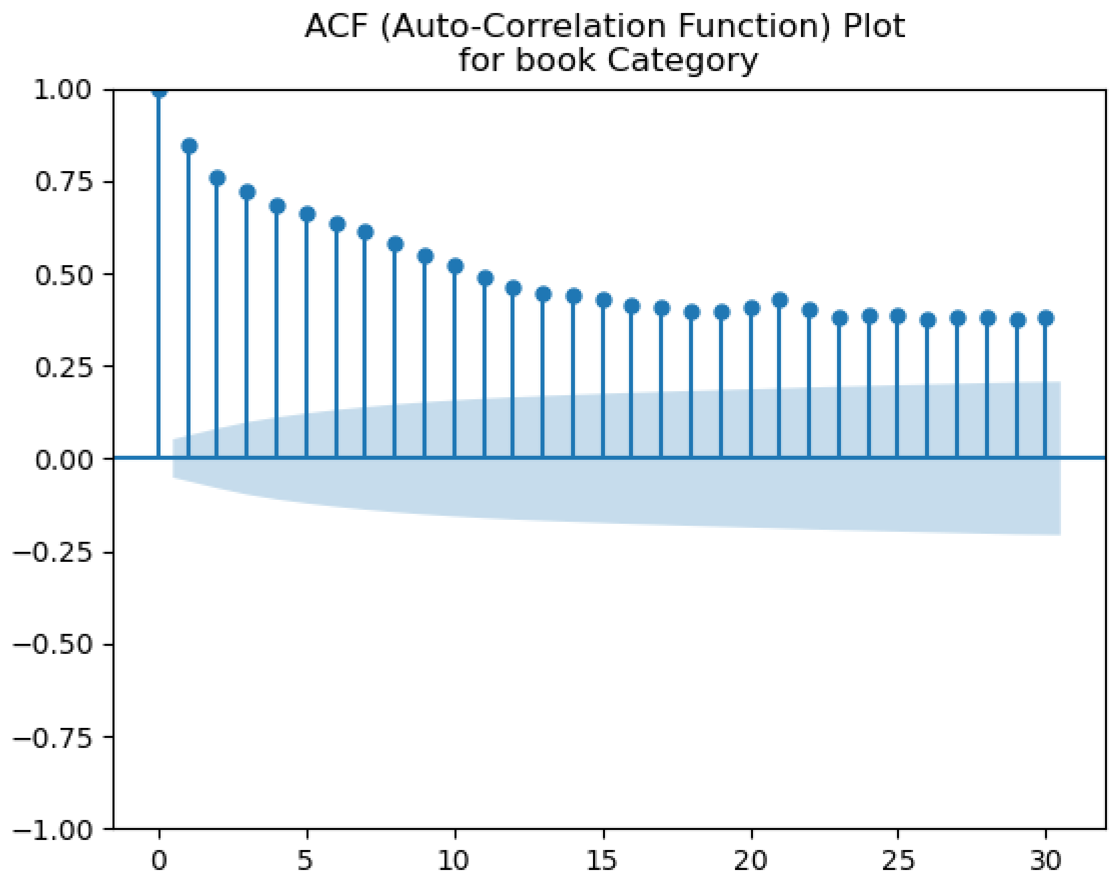
In [8]:
```
# Plot seasonal subseries for each category
for category in df['Category'].unique():
    category_data = df[df['Category'] == category]['Net Item Sales']
    seasonal_subseries = category_data.groupby(category_data.index.month).mean()  # Gr
    plt.plot(seasonal_subseries)
    plt.title(f'Seasonal Subseries Plot for {category} Category')
    plt.xlabel('Month (gregorian)')
    plt.ylabel('Average Sales')
    plt.show()
```

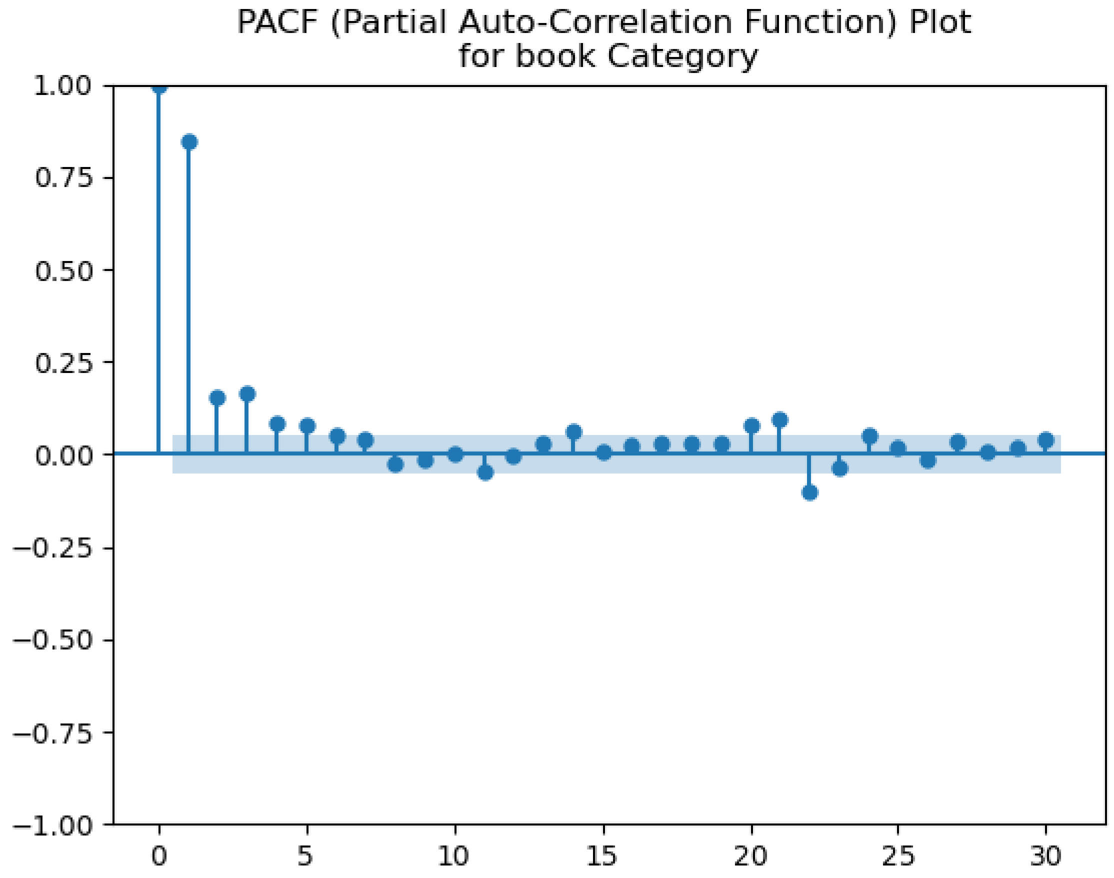


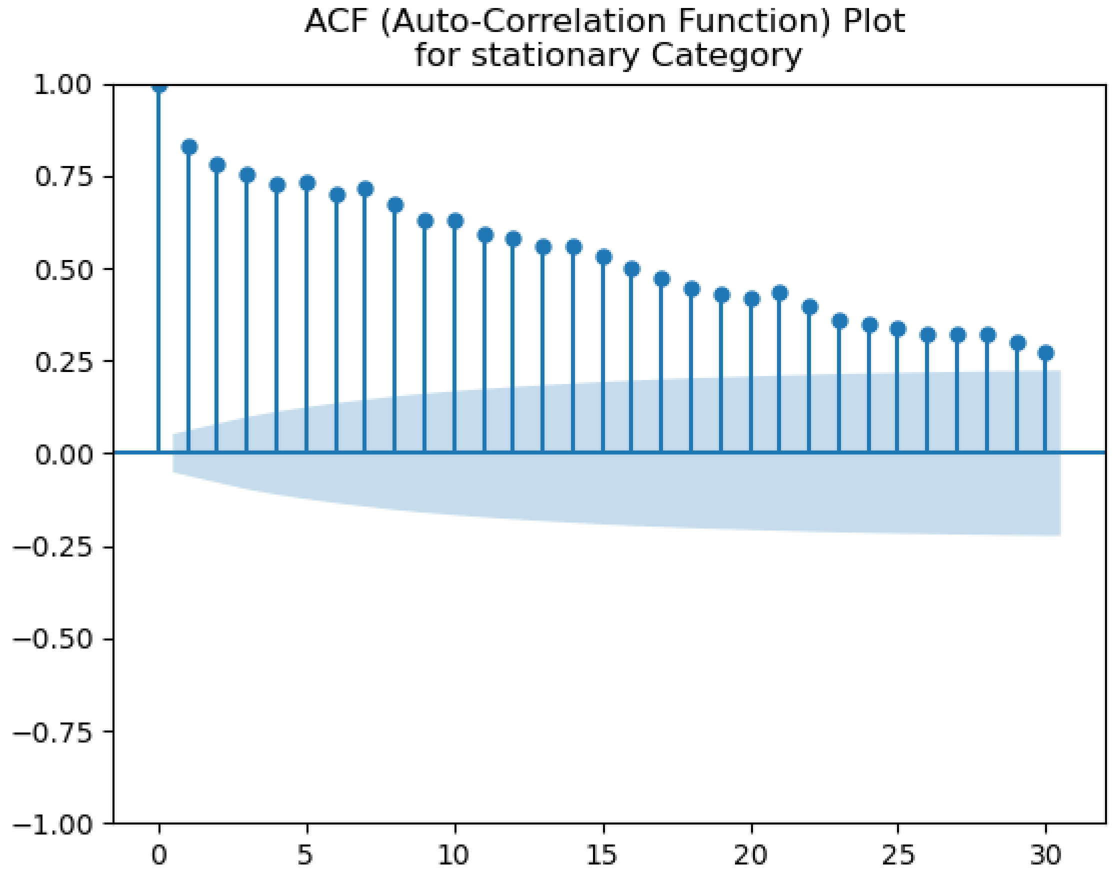


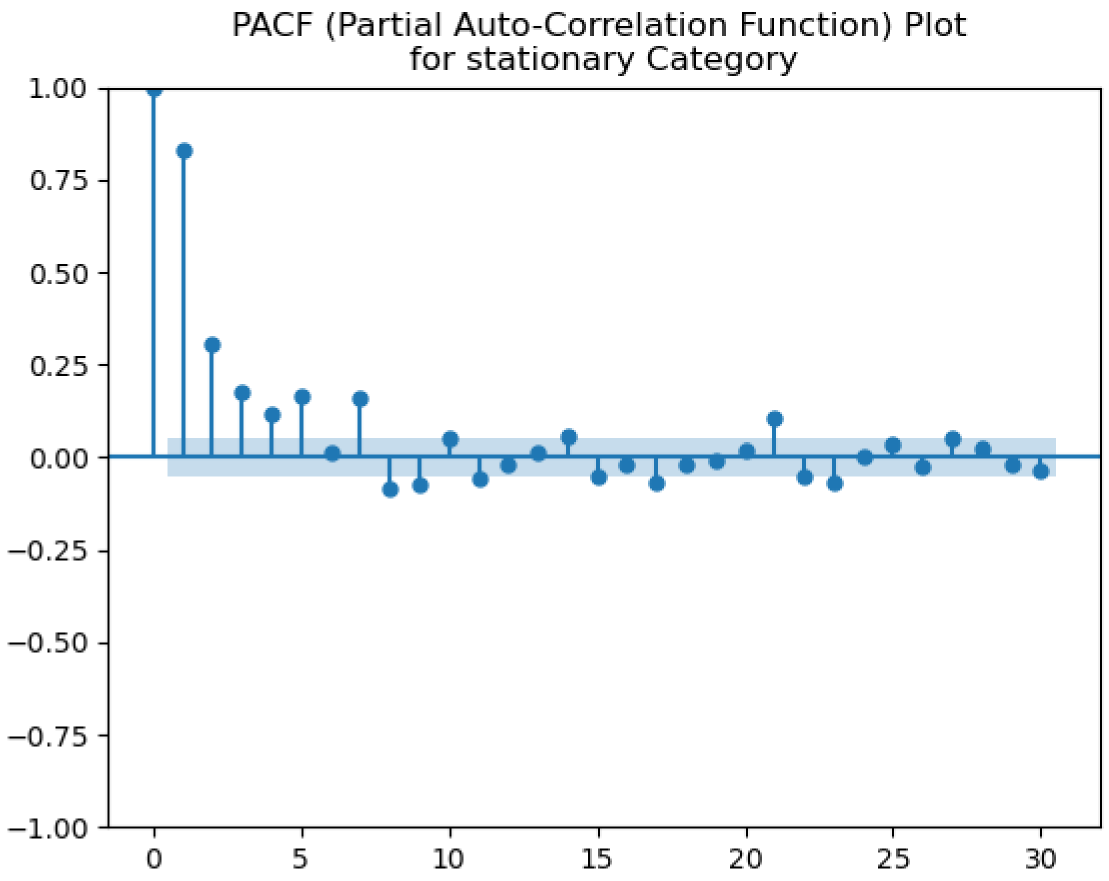
Based on the average monthly sales trends for the "book" and "stationary" categories, the following analytical statements can be made: - Monthly Sales Fluctuations: Both categories, "book" and "stationary," show fluctuating patterns in their average monthly sales throughout the year. - Seasonal Trends: For the "book" category, the highest average monthly sales are observed in the second month (February) and the lowest in the fourth month (April). There seems to be a trend of higher sales during the first and last quarters of the year. In contrast, the "stationary" category experiences its peak sales during the ninth month (September) and lowest sales during the fourth month (April). There's a significant spike in sales during the back-to-school season (September) for stationary products. - Intra-Year Variability: Both categories experience variations in sales throughout the year, indicating potential seasonal demand shifts. The "book" category's average monthly sales show a gradual decline from the beginning of the year, reaching its lowest point in the middle of the year, before starting to rise again. The "stationary" category, on the other hand, experiences a sudden surge in sales during the ninth month, suggesting a unique period of high demand. - Potential Influences: The observed patterns in both categories might be influenced by factors such as academic calendars, holidays, or specific events. The sales trends could be attributed to various factors, such as school semesters, holidays, and promotional periods that affect the demand for these categories of products. Overall, understanding these seasonal sales trends can help businesses in managing their inventory, planning marketing strategies, and optimizing their supply chain to align with periods of increased or decreased demand for "book" and "stationary" products.

In [9]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Create ACF and PACF plots for each category
for category in df['Category'].unique():
    category_data = df[df['Category'] == category]['Net Item Sales']
    plot_acf(category_data, lags=30)  # Adjust lags as needed
    plt.title(f'ACF (Auto-Correlation Function) Plot \nfor {category} Category')
    plot_pacf(category_data, lags=30)  # Adjust lags as needed
    plt.title(f'PACF (Partial Auto-Correlation Function) Plot \nfor {category} Categor
```




PACF (Partial Auto-Correlation Function) Plot
for book Category
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30

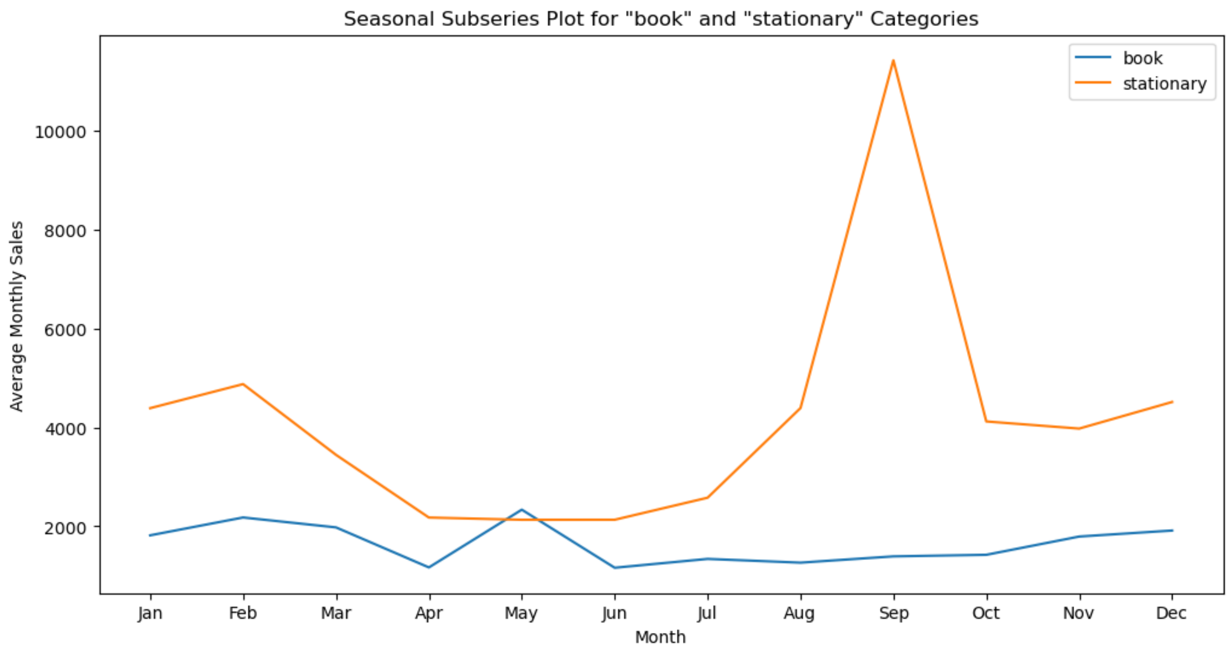

ACF (Auto-Correlation Function) Plot
for stationary Category
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30



For both the "book" and "Stationary" categories, the gradual decline in the ACF plot suggests that there is some correlation between the current values and their lagged values. This is a common characteristic in time series data. However, the presence of significant spikes in the PACF plot suggests that there is a direct effect of certain lags on the current values, while accounting for the correlations at shorter lags. In summary, the observed patterns in the ACF and PACF plots suggest that the "book" and "Stationary" categories may have a significant seasonal component, and a seasonal ARIMA model could potentially capture and explain the underlying seasonality in the data for accurate forecasting and analysis.

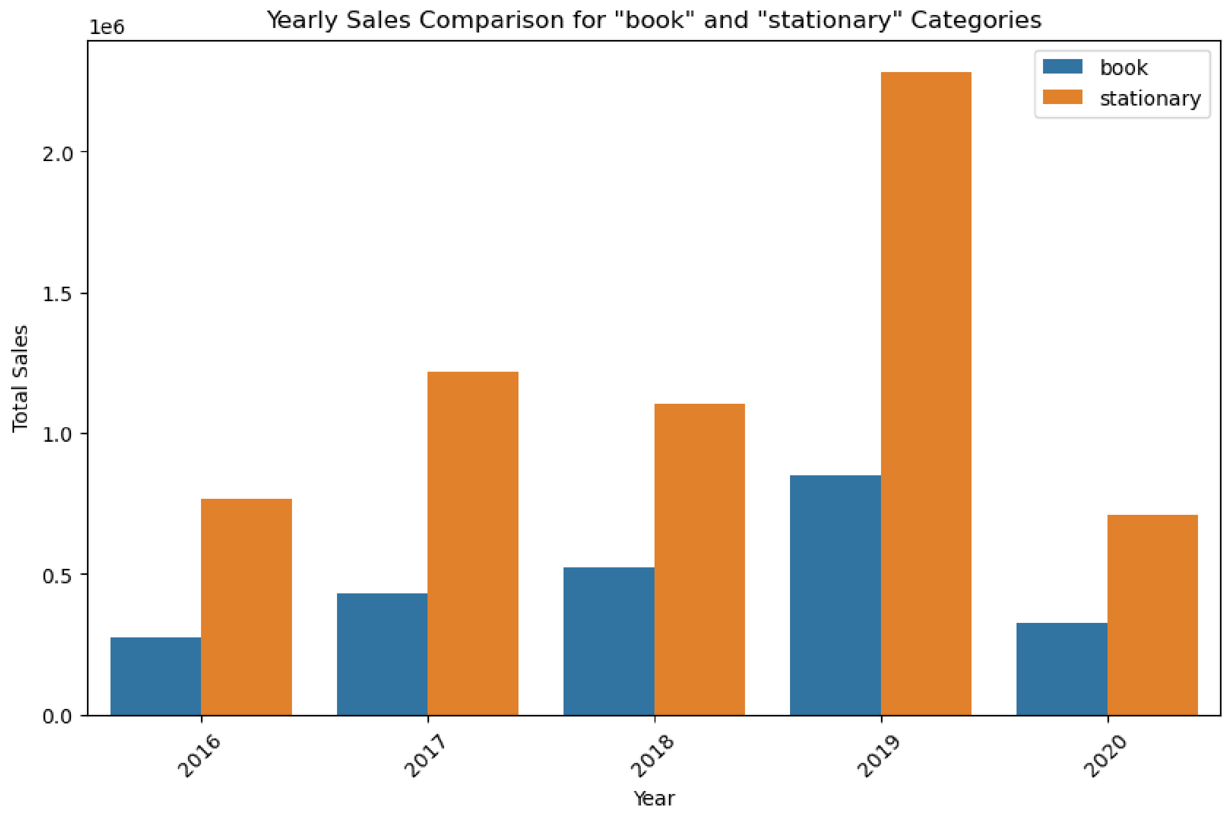
3) Use Visualization techniques to describe sales trends for these 2 categories over the 4-year period. Clearly indicate the insights derived from each chart or graph.

In [10]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Seasonal Subseries Plot
plt.figure(figsize=(12, 6))
for category in df['Category'].unique():
    category_data = df[df['Category'] == category]['Net Item Sales']
    seasonal_subseries = category_data.groupby(category_data.index.month).mean()
    plt.plot(seasonal_subseries, label=category)
plt.title('Seasonal Subseries Plot for "book" and "stationary" Categories')
plt.xlabel('Month')
plt.ylabel('Average Monthly Sales')
plt.xticks(range(1, 13), calendar.month_abbr[1:])
plt.legend()
plt.show()
```



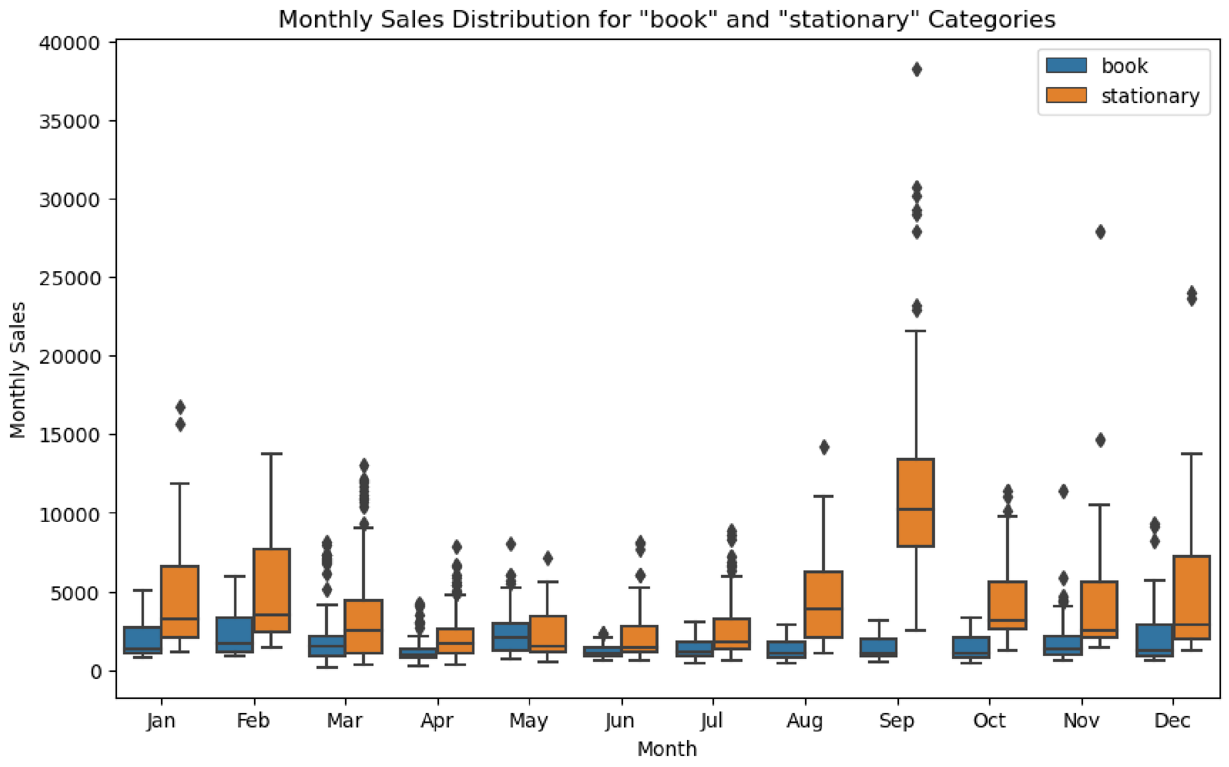
Peaks and troughs in sales can be observed in both categories. These peaks indicate the months when sales are at their highest, while troughs represent periods of lower sales. For the "book" category, the peak sales months appear to be around February, May, and November. In contrast, the "stationary" category experiences higher sales peaks in February, September, and December.

```
In [11]: # Bar Chart - Yearly Sales Comparison
         yearly_sales = df.groupby(['Category', df.index.year])['Net Item Sales'].sum().reset_i
         plt.figure(figsize=(10, 6))
         sns.barplot(data=yearly_sales, x='Gregorian Date', y='Net Item Sales', hue='Category')
         plt.title('Yearly Sales Comparison for "book" and "stationary" Categories')
         plt.xlabel('Year')
         plt.ylabel('Total Sales')
         plt.xticks(rotation=45)
         plt.legend()
         plt.show()
```



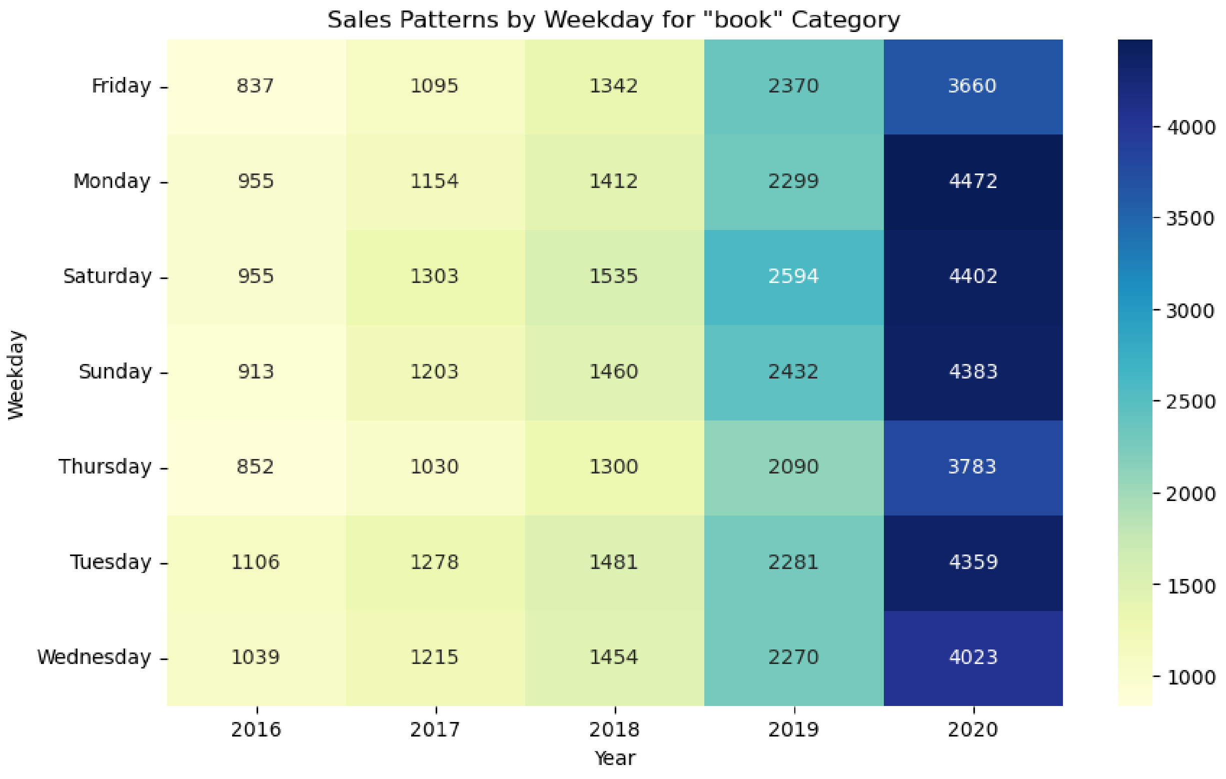
This Bar Chart visually summarizes the total sales performance of the "book" and "stationary" categories over a four-year period. It enables easy comparison of yearly sales trends, growth or decline, and relative performance between the categories. The chart helps identify peak sales years, differences in sales, and potential opportunities for improving business strategies. It offers insights into stable versus fluctuating sales performance and highlights the impact of external factors on sales variations. Overall, the chart provides valuable information for making informed decisions about inventory management, marketing, and resource allocation to enhance overall sales performance.

In [12]:
```
# Box Plot - Monthly Sales Distribution
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x=df.index.month, y='Net Item Sales', hue='Category')
plt.title('Monthly Sales Distribution for "book" and "stationary" Categories')
plt.xlabel('Month')
plt.ylabel('Monthly Sales')
plt.xticks(range(12), calendar.month_abbr[1:])
plt.legend()
plt.show()
```

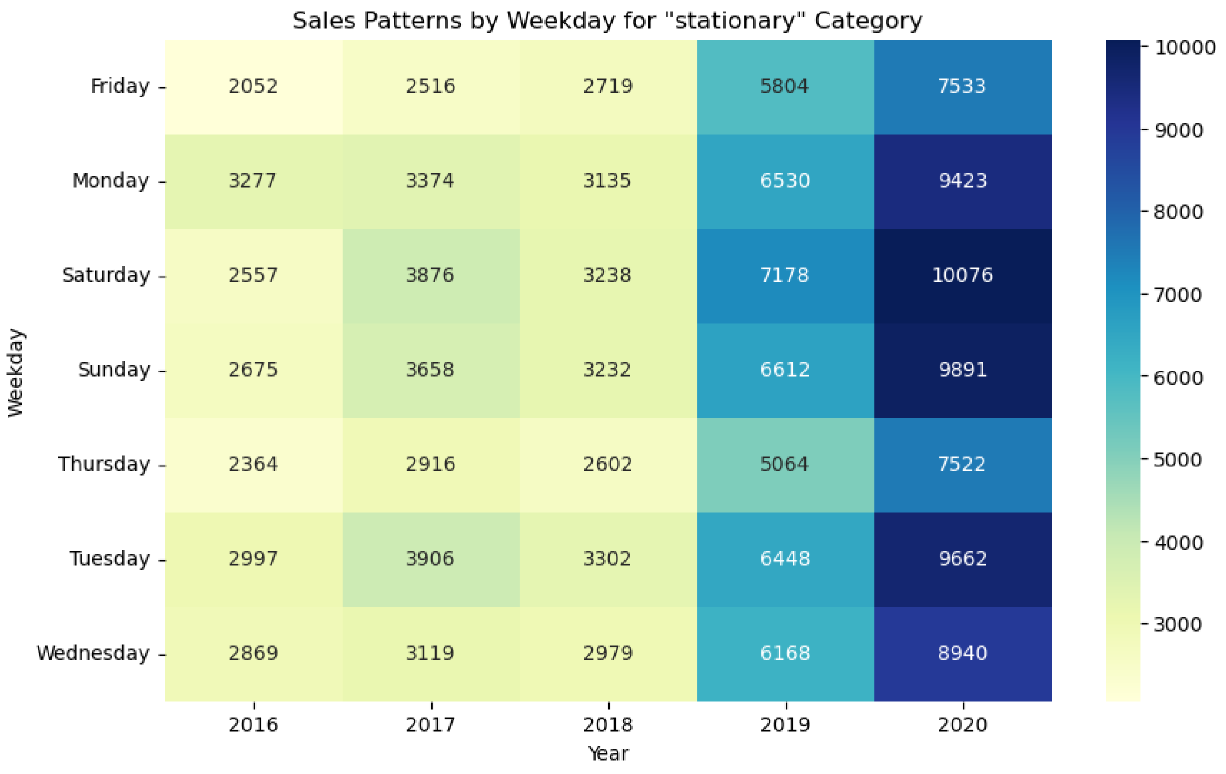


This Box Plot provides insights into the variation, central tendency, and outliers in monthly sales for both "book" and "stationary" categories. It highlights differences in sales patterns, identifies potential anomalies, and aids in understanding the spread of sales data across months. This visualization helps assess seasonality, monthly trends, and differences between categories for informed decision-making in inventory and sales strategies.

In [13]:
```
import calendar

# Extract weekday information from the index
df['Weekday'] = df.index.weekday

# Map weekday numbers to weekday names
weekday_names = [calendar.day_name[i] for i in range(7)]
df['Weekday'] = df['Weekday'].map(lambda x: weekday_names[x])

# Separate the data for the "book" and "stationary" categories
book_data = df[df['Category'] == 'book']
stationary_data = df[df['Category'] == 'stationary']

# Pivot the data for the heatmap - Sales Patterns by Weekday for "book" Category
book_weekday_sales = book_data.pivot_table(index='Weekday', columns=book_data.index.ye
plt.figure(figsize=(10, 6))
sns.heatmap(book_weekday_sales, cmap='YlGnBu', annot=True, fmt='.0f')
plt.title('Sales Patterns by Weekday for "book" Category')
plt.xlabel('Year')
plt.ylabel('Weekday')
plt.show()

# Pivot the data for the heatmap - Sales Patterns by Weekday for "stationary" Category
stationary_weekday_sales = stationary_data.pivot_table(index='Weekday', columns=static
plt.figure(figsize=(10, 6))
sns.heatmap(stationary_weekday_sales, cmap='YlGnBu', annot=True, fmt='.0f')
plt.title('Sales Patterns by Weekday for "stationary" Category')
plt.xlabel('Year')
plt.ylabel('Weekday')
plt.show()
```



Weekday	2016	2017	2018	2019	2020
Friday	837	1095	1342	2370	3660
Monday	955	1154	1412	2299	4472
Saturday	955	1303	1535	2594	4402
Sunday	913	1203	1460	2432	4383
Thursday	852	1030	1300	2090	3783
Tuesday	1106	1278	1481	2281	4359
Wednesday	1039	1215	1454	2270	4023

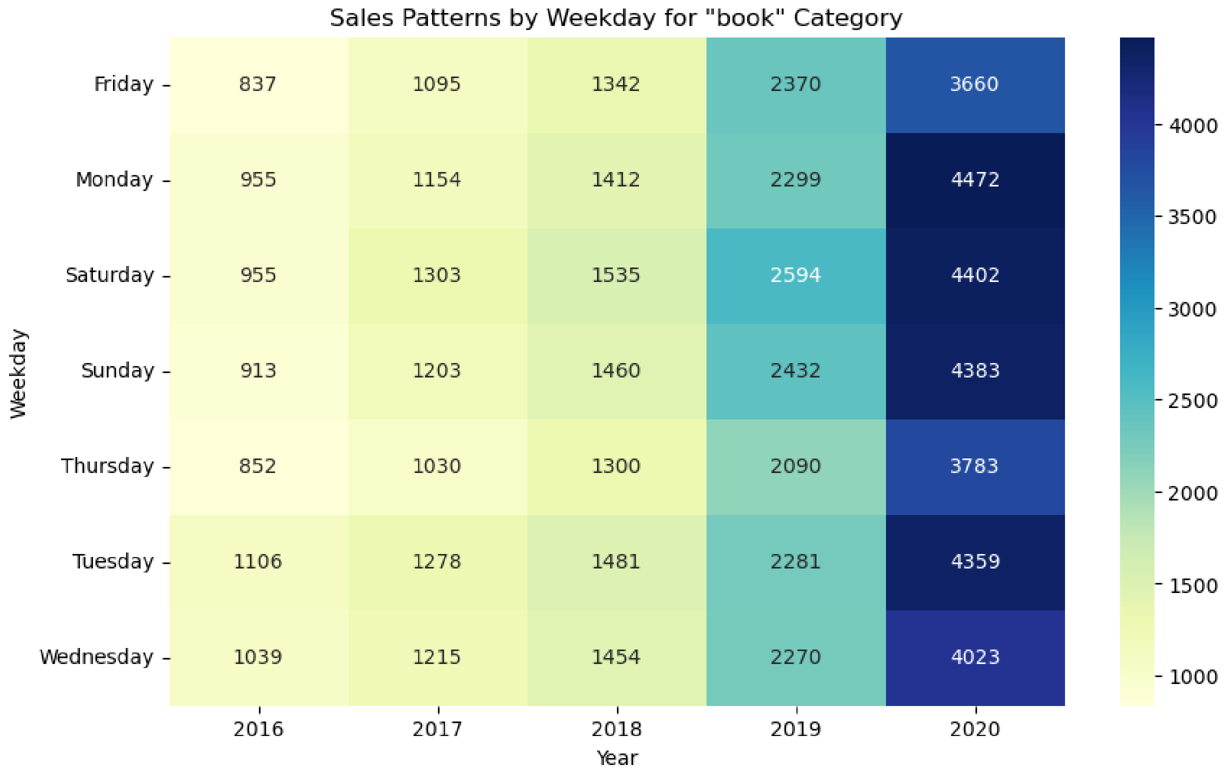


Weekday	2016	2017	2018	2019	2020
Friday	2052	2516	2719	5804	7533
Monday	3277	3374	3135	6530	9423
Saturday	2557	3876	3238	7178	10076
Sunday	2675	3658	3232	6612	9891
Thursday	2364	2916	2602	5064	7522
Tuesday	2997	3906	3302	6448	9662
Wednesday	2869	3119	2979	6168	8940

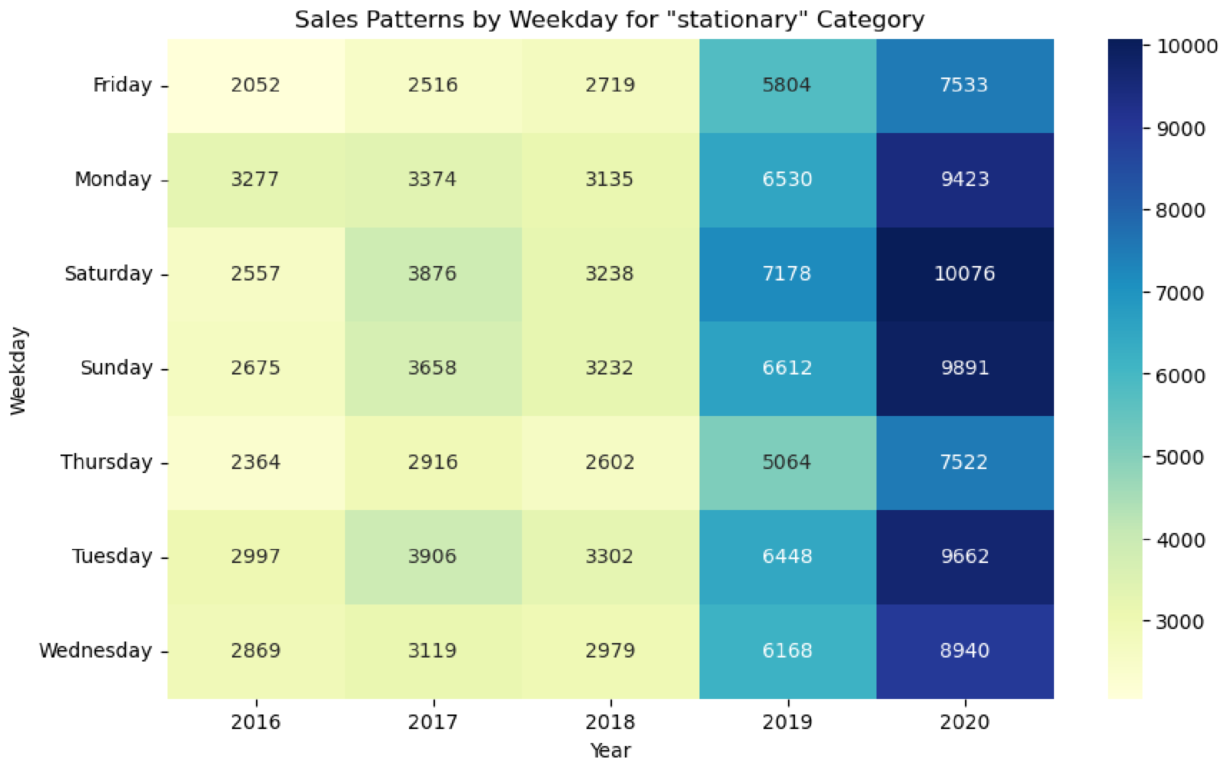
In [14]:

```python
import calendar

# Extract weekday information from the index
df['Weekday'] = df.index.weekday

# Map weekday numbers to weekday names
weekday_names = [calendar.day_name[i] for i in range(7)]
df['Weekday'] = df['Weekday'].map(lambda x: weekday_names[x])

# Separate the data for the "book" and "stationary" categories
```

```
book_data = df[df['Category'] == 'book']
stationary_data = df[df['Category'] == 'stationary']

# Pivot the data for the heatmap - Sales Patterns by Weekday for "book" Category
book_weekday_sales = book_data.pivot_table(index='Weekday', columns=book_data.index.ye
plt.figure(figsize=(10, 6))
sns.heatmap(book_weekday_sales, cmap='YlGnBu', annot=True, fmt='.0f')
plt.title('Sales Patterns by Weekday for "book" Category')
plt.xlabel('Year')
plt.ylabel('Weekday')
plt.show()

# Pivot the data for the heatmap - Sales Patterns by Weekday for "stationary" Category
stationary_weekday_sales = stationary_data.pivot_table(index='Weekday', columns=static
plt.figure(figsize=(10, 6))
sns.heatmap(stationary_weekday_sales, cmap='YlGnBu', annot=True, fmt='.0f')
plt.title('Sales Patterns by Weekday for "stationary" Category')
plt.xlabel('Year')
plt.ylabel('Weekday')
plt.show()
```

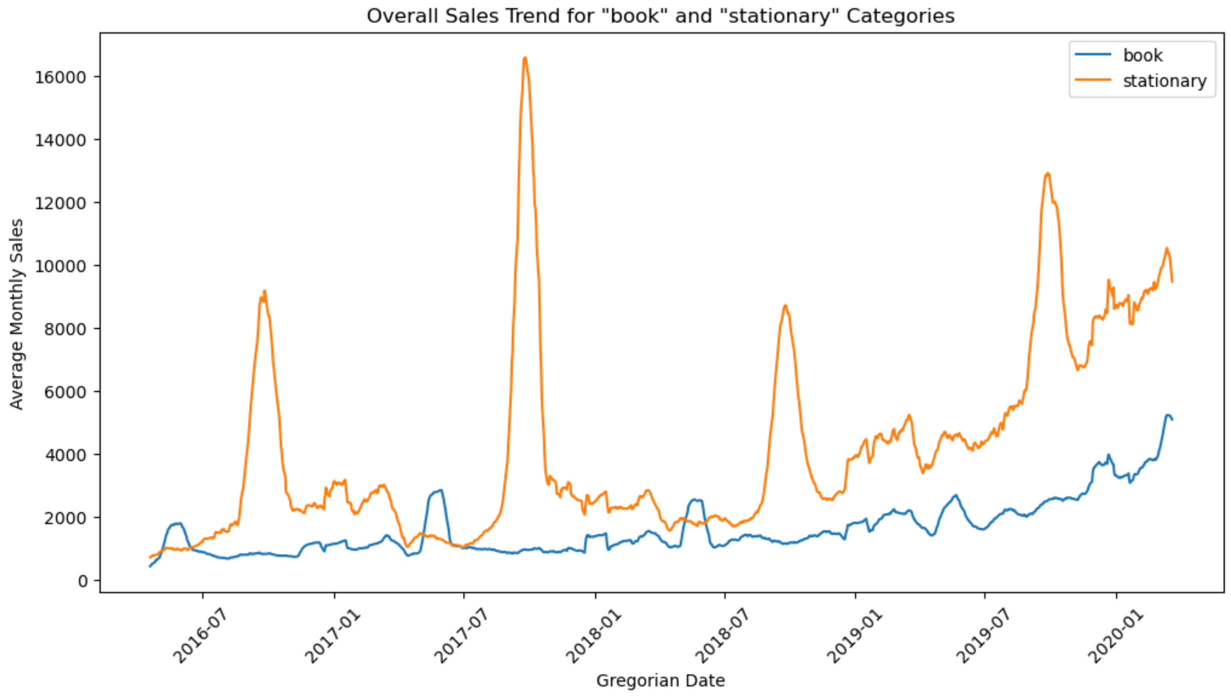


Weekday	2016	2017	2018	2019	2020
Friday	837	1095	1342	2370	3660
Monday	955	1154	1412	2299	4472
Saturday	955	1303	1535	2594	4402
Sunday	913	1203	1460	2432	4383
Thursday	852	1030	1300	2090	3783
Tuesday	1106	1278	1481	2281	4359
Wednesday	1039	1215	1454	2270	4023



Weekday	2016	2017	2018	2019	2020
Friday	2052	2516	2719	5804	7533
Monday	3277	3374	3135	6530	9423
Saturday	2557	3876	3238	7178	10076
Sunday	2675	3658	3232	6612	9891
Thursday	2364	2916	2602	5064	7522
Tuesday	2997	3906	3302	6448	9662
Wednesday	2869	3119	2979	6168	8940

In [15]:
```
# Line Chart with Trendline - Overall Trend
plt.figure(figsize=(12, 6))
for category in df['Category'].unique():
    category_data = df[df['Category'] == category]['Net Item Sales']
    trend = category_data.rolling(window=30).mean()
    plt.plot(category_data.index, trend, label=category)
plt.title('Overall Sales Trend for "book" and "stationary" Categories')
plt.xlabel('Gregorian Date')
plt.ylabel('Average Monthly Sales')
plt.xticks(rotation=45)
plt.legend()
plt.show()

# # Create a line chart for monthly total sales for each category
# plt.figure(figsize=(12, 6))
# sns.lineplot(data=grouped_data, x='Gregorian Month Number', y='Net Item Sales', hue=
# plt.xlabel('Gregorian Month Number')
# plt.ylabel('Total Sales')
# plt.title('Monthly Total Sales Trends for Each Category')
# plt.legend()
# plt.xticks(range(1, 13), calendar.month_abbr[1:])  # Use abbreviated month names
# #plt.tight_layout()
# plt.show()
```



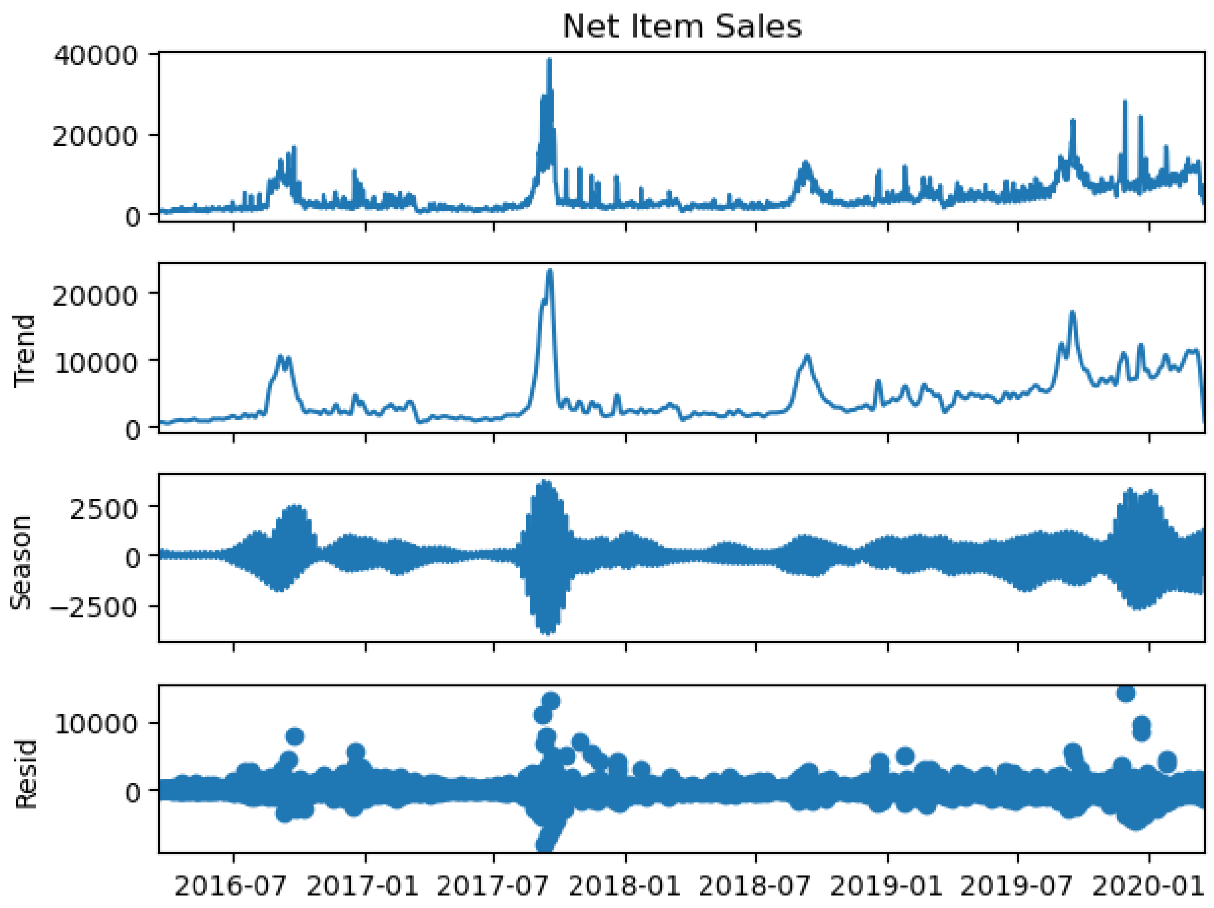
4) Based on the provided historical data, utilize forecasting techniques to predict monthly sales for "stationary" in the year 1399. In your response, clearly state your assumptions, outline the specific factors you considered, and elaborate on the forecasting methods or models you employed.

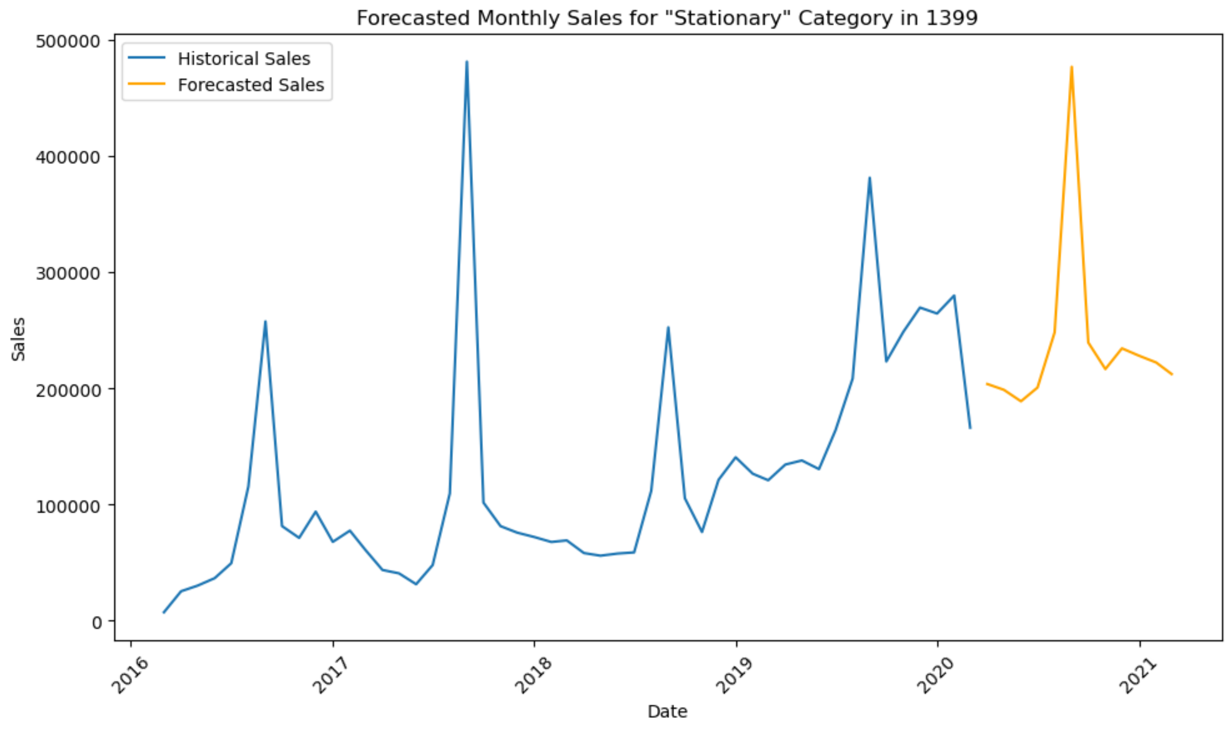
Forecasting Monthly Sales for "Stationary" Category in the Year 1399 Assumptions and Factors Considered: 1. Seasonality: The historical data exhibits distinct seasonal patterns, as observed during the analysis. These patterns are assumed to continue in the year 1399. 2. Yearly Trends: Any underlying yearly trends or growth patterns are assumed to be consistent with previous years. 3. External Factors: The forecast is based solely on historical sales data and does not incorporate external factors like economic conditions, marketing campaigns, or industry trends. 4. Stability: The assumption is that the underlying sales process remains relatively stable and does not undergo significant shifts or disruptions. Forecasting Methodology: 1. Time Series Decomposition: A seasonal decomposition technique such as Seasonal and Trend decomposition using LOESS (STL) can be employed to separate the time series data into its constituent components: trend, seasonality, and remainder. 2. Exponential Smoothing: An exponential smoothing method such as Holt-Winters can be used to capture the trend and seasonality in the data and provide forecasts for the upcoming year. 3. ARIMA Modeling: Autoregressive Integrated Moving Average (ARIMA) models can be explored to capture the autoregressive and moving average components of the time series and make forecasts accordingly. Forecasting Process: 1. Data Preparation: The historical sales data for the "stationary" category will be preprocessed to create a time series. 2. Decomposition: The time series will be decomposed using STL to extract trend, seasonality, and remainder components. 3. Forecasting Model Selection: Exponential smoothing and ARIMA models will be considered for forecasting. 4. Model Fitting: The selected model(s) will be fitted to the historical data, taking into account the decomposed components. 5. Forecast Generation: Using the fitted model(s), forecasts for the year 1399 will be generated, incorporating both trend and seasonality. 6. Evaluation and Validation: The forecasted values will be compared to the actual sales data from 1399 (if available) to evaluate the accuracy of the chosen forecasting method. Expected Outcome: The forecasting methods utilized should provide a reliable estimate of monthly sales for the "stationary" category in the year 1399, taking into account the observed seasonality and trends in the historical data. However, it's important to note that while the forecasts can provide valuable insights, they are based solely on historical patterns and do not consider potential external factors that could influence sales in 1399. Therefore, the forecasts should be interpreted as a guide rather than an absolute prediction.

In [16]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.seasonal import STL
```

```
# Extract sales data for the "stationary" category
stationary_data = df[df['Category'] == 'stationary']['Net Item Sales']

# Resample the historical data to a monthly basis
stationary_data_monthly = stationary_data.resample('MS').sum()

# Perform STL decomposition for each category
category_data = stationary_data
stl = STL(category_data, seasonal=13)  # Adjust seasonal parameter as needed
result = stl.fit()
result.plot()

# Forecasting using Exponential Smoothing
model = ExponentialSmoothing(stationary_data_monthly, seasonal='add', seasonal_periods
model_fit = model.fit()

# Forecast sales for the year 1399
forecasted_sales = model_fit.forecast(steps=12)  # Assuming 12 months in the year

# Create a date range for the forecasted months in 1399
forecast_dates = pd.date_range(start=JalaliDate(1399, 1, 1).to_gregorian(), periods=12

# Create a DataFrame to store the forecasted sales
forecast_df = pd.DataFrame({'Date': forecast_dates, 'Forecasted Sales': forecasted_sal
forecast_df.set_index('Date', inplace=True)

# Plot the historical sales and forecasted sales
plt.figure(figsize=(10, 6))
plt.plot(stationary_data_monthly.index, stationary_data_monthly.values, label='Histori
plt.plot(forecast_df.index, forecast_df['Forecasted Sales'], label='Forecasted Sales',
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Forecasted Monthly Sales for "Stationary" Category in 1399')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Display the forecasted sales for the year 1399
display(pd.DataFrame(forecast_df))
```

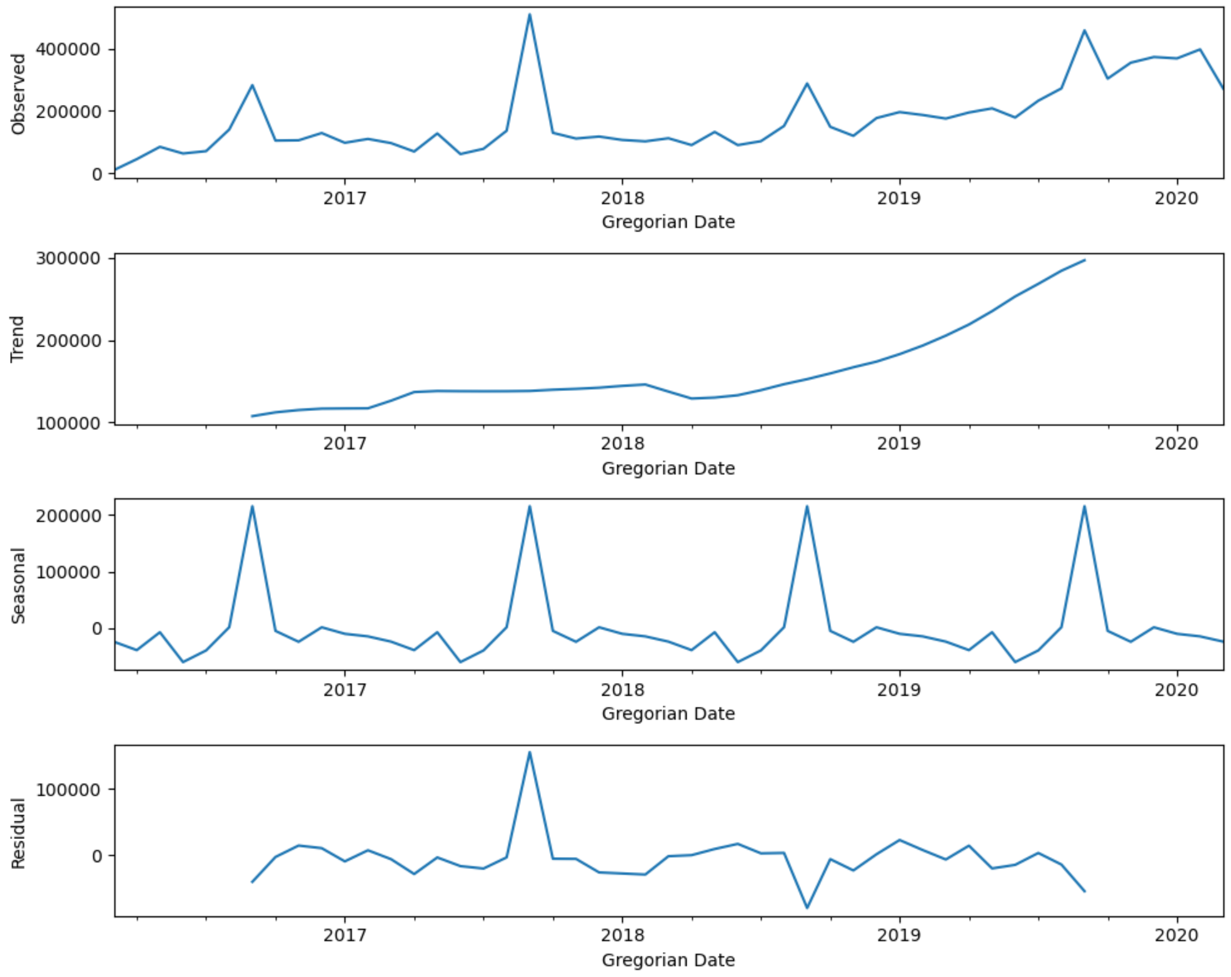
```
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:917: C
onvergenceWarning: Optimization failed to converge. Check mle_retvals.
  warnings.warn(
```


Net Item Sales
40000
20000
0
Trend
20000
10000
0
Season
2500
0
−2500
Resid
10000
0
2016-07
2017-01
2017-07
2018-01
2018-07
2019-01
2019-07
2020-01


Forecasted Monthly Sales for "Stationary" Category in 1399
Historical Sales
Forecasted Sales
Sales
500000
400000
300000
200000
100000
0
2016
2017
2018
2019
2020
2021
Date

	Forecasted Sales
Date	
2020-04-01	203597.349867
2020-05-01	198602.389719
2020-06-01	188720.444553
2020-07-01	200558.210903
2020-08-01	248005.362335
2020-09-01	476509.566763
2020-10-01	239251.919053
2020-11-01	216432.714021
2020-12-01	234266.819513
2021-01-01	227914.938118
2021-02-01	222111.610271
2021-03-01	212162.677340

In [17]:
```
# Resample to monthly frequency and sum the sales
df_monthly = df['Net Item Sales'].resample('M').sum()

# Perform seasonal decomposition
decomposition = sm.tsa.seasonal_decompose(df_monthly, model='additive')

# Plot the decomposition
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(10, 8))
decomposition.observed.plot(ax=ax1)
ax1.set_ylabel('Observed')
decomposition.trend.plot(ax=ax2)
ax2.set_ylabel('Trend')
decomposition.seasonal.plot(ax=ax3)
ax3.set_ylabel('Seasonal')
decomposition.resid.plot(ax=ax4)
ax4.set_ylabel('Residual')
plt.tight_layout()
plt.show()
```



In [18]:

```python
# Fit an ARIMA model to the data
model = sm.tsa.ARIMA(df['Net Item Sales'], order=(1, 1, 1))
results = model.fit()

# Print model summary
print(results.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:         Net Item Sales   No. Observations:                 2920
Model:                 ARIMA(1, 1, 1)   Log Likelihood              -25484.370
Date:                Thu, 10 Aug 2023   AIC                          50974.741
Time:                        11:56:11   BIC                          50992.678
Sample:                             0   HQIC                         50981.202
                               - 2920
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2063      0.009     23.857      0.000       0.189       0.223
ma.L1         -0.7057      0.007    -95.624      0.000      -0.720      -0.691
sigma2      2.317e+06   1.33e+04    174.346      0.000    2.29e+06    2.34e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):            249476.28
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):              18.23   Skew:                             3.19
Prob(H) (two-sided):                  0.00   Kurtosis:                        47.84
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: Valu
eWarning: A date index has been provided, but it has no associated frequency informat
ion and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: Valu
eWarning: A date index has been provided, but it is not monotonic and so will be igno
red when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: Valu
eWarning: A date index has been provided, but it has no associated frequency informat
ion and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: Valu
eWarning: A date index has been provided, but it is not monotonic and so will be igno
red when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: Valu
eWarning: A date index has been provided, but it has no associated frequency informat
ion and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: Valu
eWarning: A date index has been provided, but it is not monotonic and so will be igno
red when e.g. forecasting.
  self._init_dates(dates, freq)
```

In [19]:
```
grouped_data = df.groupby(['Category', 'Gregorian Year', 'Gregorian Month Number'])['N

stationary_data = grouped_data[grouped_data['Category'] == 'stationary']
```

In [20]:
```
# Group data by month and calculate total monthly sales
monthly_sales = stationary_data.groupby(['Gregorian Year', 'Gregorian Month Number'])[

# Reset index to make it easier for forecasting
monthly_sales = monthly_sales.reset_index()
```

```
# Convert the 'Gregorian Month Number' column to integer
monthly_sales['Gregorian Month Number'] = monthly_sales['Gregorian Month Number'].asty

# Create a time series from the data
ts = monthly_sales.set_index(pd.to_datetime(monthly_sales['Gregorian Year'].astype(str
```

In [21]:

```
# Split the data into training and testing sets
train_size = int(len(ts) * 37/49)
train_data = ts.iloc[:train_size]
test_data = ts.iloc[train_size:]
```

In [22]:

```
# Convert Persian start and end dates to Gregorian
forecast_start = JalaliDate(1399, 1, 1).to_gregorian()
forecast_end = JalaliDate(1399, 12, 29).to_gregorian()

# Fit the SARIMA model
order = (1, 1, 1)  # You can adjust these parameters based on model evaluation
seasonal_order = (1, 1, 1, 12)  # Assuming yearly seasonality
model = sm.tsa.SARIMAX(train_data, order=order, seasonal_order=seasonal_order, enforce
results = model.fit()
```

```
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: Valu
eWarning: No frequency information was provided, so inferred frequency MS will be use
d.
  self._init_dates(dates, freq)
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: Valu
eWarning: No frequency information was provided, so inferred frequency MS will be use
d.
  self._init_dates(dates, freq)
C:\Users\Asus\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:866:
UserWarning: Too few observations to estimate starting parameters for seasonal ARMA.
All parameters except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
```

In [23]:

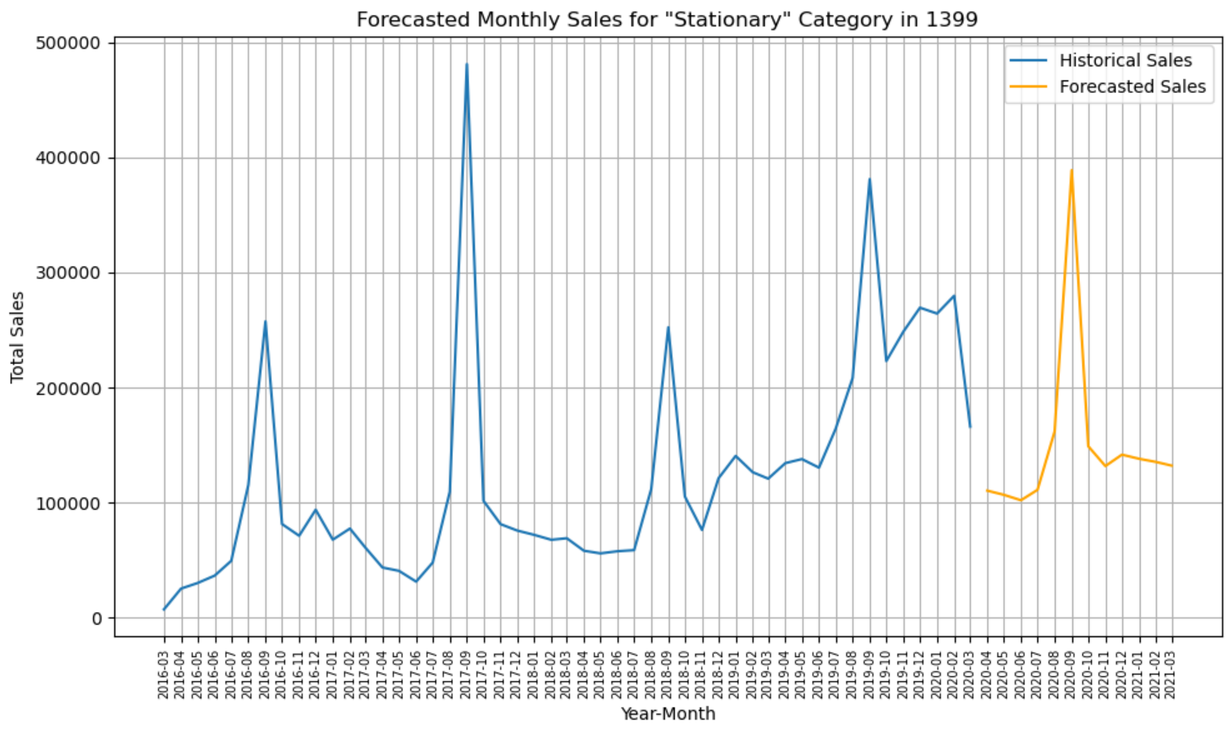
```
# Forecast sales for the year 1399
forecast = results.get_forecast(steps=12)  # Forecasting for 12 months

forecast_index = pd.date_range(start=forecast_start, end=forecast_end, freq='MS')
forecast_values = forecast.predicted_mean.values


pd.DataFrame(forecast.predicted_mean)
```

Out[23]:

	predicted_mean
2019-04-01	110376.805807
2019-05-01	106755.395459
2019-06-01	102030.308218
2019-07-01	111094.326726
2019-08-01	161782.379765
2019-09-01	388873.879259
2019-10-01	148926.155669
2019-11-01	131743.682941
2019-12-01	141650.306641
2020-01-01	138077.169862
2020-02-01	135284.986744
2020-03-01	132134.467158

In [24]:

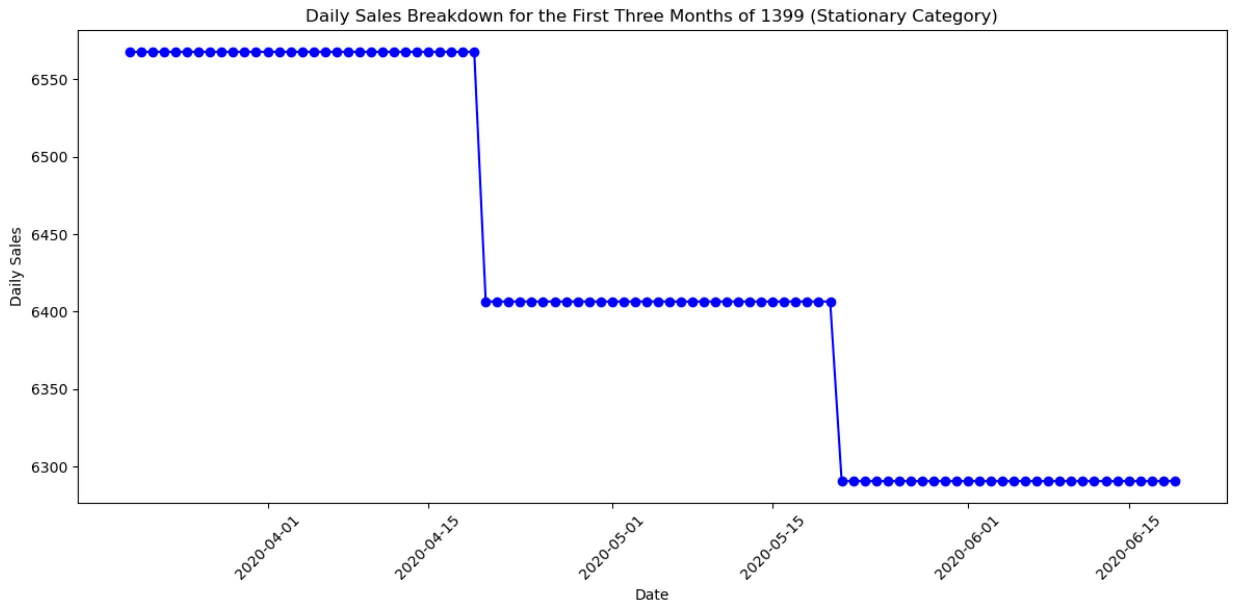
```
# Plot the forecast
plt.figure(figsize=(10, 6))
plt.plot(ts.index, ts.values, label='Historical Sales')
plt.plot(forecast_index, forecast_values, label='Forecasted Sales', color='orange')
plt.xlabel('Year-Month')
plt.ylabel('Total Sales')
plt.title('Forecasted Monthly Sales for "Stationary" Category in 1399')
plt.legend()

xtixks=(ts.index).append(forecast_index)
# Set xticks for historical sales
plt.xticks(xtixks, [date.strftime('%Y-%m') for date in xtixks], rotation=90, size=7)

# Manually adjust the spacing to make xticks for historical sales visible
plt.subplots_adjust(bottom=0.15)

plt.tight_layout()
plt.grid(True)
plt.show()
```



5) Continuing from question 4, considering the "stationary" category, break down your predicted sales for the first 3 months of the year into daily sales figures for each day of the month. Explain factors or methods you consider in making this breakdown.

In [25]:
```python
# Assuming you have 'forecast_df' DataFrame from the previous step
# Calculate the average daily sales during historical first three months
historical_avg_daily_sales = stationary_data_monthly[:3].mean()

# Calculate the total predicted sales for each month
predicted_monthly_sales = forecast_df['Forecasted Sales']

# Define the number of days in each month for the first three months
days_in_month = [31, 31, 30]  # Replace with actual days for the corresponding months

# Initialize a list to store daily sales breakdown
daily_sales_breakdown = []

# Iterate through each month
for i, monthly_sales in enumerate(predicted_monthly_sales):
    if i < len(days_in_month):  # Ensure we have enough months in 'days_in_month'
        days = days_in_month[i]
        avg_daily_sales = monthly_sales / days
        daily_sales = [avg_daily_sales] * days
        daily_sales_breakdown.extend(daily_sales)

# Create a DataFrame to store the daily sales breakdown
daily_sales_df = pd.DataFrame({'Daily Sales': daily_sales_breakdown},
                              index=pd.date_range(start=JalaliDate(1399, 1, 1).to_greg

# Display the daily sales breakdown for the first three months
print(daily_sales_df.head(90))  # Display first 3 months (90 days)

# Plot the daily sales breakdown for the first three months
plt.figure(figsize=(12, 6))
plt.plot(daily_sales_df.index, daily_sales_df['Daily Sales'], marker='o', linestyle='-
plt.xlabel('Date')
plt.ylabel('Daily Sales')
plt.title('Daily Sales Breakdown for the First Three Months of 1399 (Stationary Catego
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
            Daily Sales
2020-03-20  6567.656447
2020-03-21  6567.656447
2020-03-22  6567.656447
2020-03-23  6567.656447
2020-03-24  6567.656447
...                 ...
2020-06-13  6290.681485
2020-06-14  6290.681485
2020-06-15  6290.681485
2020-06-16  6290.681485
2020-06-17  6290.681485

[90 rows x 1 columns]
```

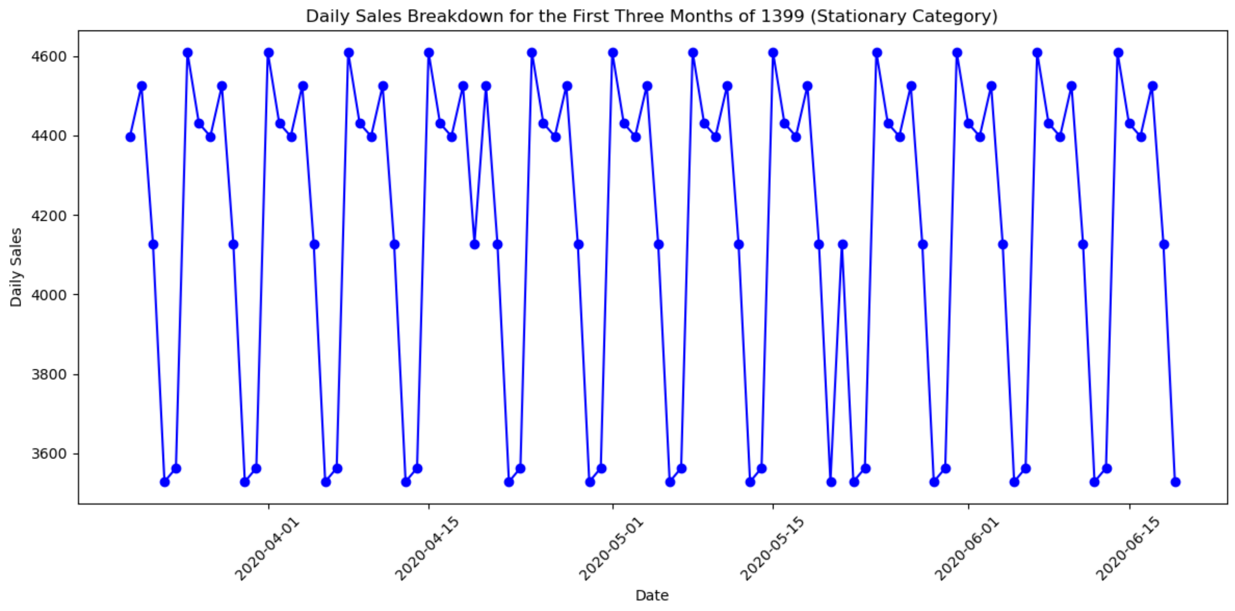


In [30]:

```
# Calculate the average daily sales for each weekday based on historical data
weekday_avg_sales = stationary_data.groupby(stationary_data.index.dayofweek)['Net Iten

# Calculate the total predicted sales for each month
predicted_monthly_sales = forecast_df['Forecasted Sales']

# Define the number of days in each month for the first three months
days_in_month = [31, 31, 30]  # Replace with actual days for the corresponding months

# Initialize a list to store daily sales breakdown
daily_sales_breakdown = []

# Iterate through each month
for i, monthly_sales in enumerate(predicted_monthly_sales):
    if i < len(days_in_month):  # Ensure we have enough months in 'days_in_month'
        days = days_in_month[i]
        avg_daily_sales = monthly_sales / days

        # Distribute average daily sales based on weekday pattern
        daily_sales = []
        for day in range(days):
            weekday = (i + day) % 7  # Calculate the weekday for the day
            avg_sales_weekday = weekday_avg_sales[weekday]
            daily_sales.append(avg_sales_weekday)

        daily_sales_breakdown.extend(daily_sales)

# Create a DataFrame to store the daily sales breakdown
daily_sales_df = pd.DataFrame({'Daily Sales': daily_sales_breakdown},
                              index=pd.date_range(start=JalaliDate(1399, 1, 1).to_greg

# Display the daily sales breakdown for the first three months
display(daily_sales_df.head(90))  # Display first 3 months (90 days)

# Plot the daily sales breakdown for the first three months
plt.figure(figsize=(12, 6))
plt.plot(daily_sales_df.index, daily_sales_df['Daily Sales'], marker='o', linestyle='
plt.xlabel('Date')
plt.ylabel('Daily Sales')
plt.title('Daily Sales Breakdown for the First Three Months of 1399 (Stationary Catego
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

	Daily Sales
2020-03-20	4398
2020-03-21	4525
2020-03-22	4128
2020-03-23	3528
2020-03-24	3563
...	...
2020-06-13	3563
2020-06-14	4610
2020-06-15	4430
2020-06-16	4398
2020-06-17	4525

90 rows × 1 columns



```
In [31]: # Export the first 90 rows of daily_sales_df to Excel
         excel_filename = "daily_sales_breakdown.xlsx"
         daily_sales_df.head(90).to_excel(excel_filename)

         print(f"Daily sales breakdown exported to {excel_filename}")
```

```
Daily sales breakdown exported to daily_sales_breakdown.xlsx
```

```
In [ ]:
```