E-commerce Sales Report

NOVEMBER 15

COMPANY NAME
Authored by: Adedeji

Table of Content

Contents

Introduction

The purpose of this report is to perform an exploratory data analysis (EDA) on historical transaction data for an e-commerce company. The company seeks to optimize its sales and marketing strategies by gaining deeper insights into customer purchasing behavior, product performance, and revenue trends.

The dataset contains key variables such as customer ID, transaction date, product ID, product category, quantity purchased, and total price. By analyzing this data, we aim to uncover patterns and trends that can guide the company in making data-driven decisions.

To achieve this, Python was used as the primary tool, leveraging powerful libraries such as Pandas for data manipulation and Matplotlib/Seaborn for visualizations. The analysis focuses on four key objectives:

1. Analysing monthly fluctuations in total revenue and transaction count.
2. Identifying product categories with the highest revenue and consistent growth.
3. Exploring seasonal variations in product sales.
4. Examining shifts in customer purchasing behavior across multiple transactions.

This report will present actionable insights supported by visualizations and tables to help the company enhance its sales strategy effectively.

Data Preparation

Data Cleaning

The dataset, containing fields such as Customer ID, Transaction Date, Product ID, Product Category, Quantity Purchased, and Total Price, was loaded and examined for quality. Key steps included:

- **Handling Missing Values**: There was no missing value in the dataset.
- **Data Transformation**: InvoiceDate column was initially an object data type but was converted to DateTime data type.
- **Data Structuring:** Aggregation and categorization to facilitate monthly, seasonal, and categorical analyses.
- **Data Filtering**: It was discovered that some input data in the Quantity column were negative due to canceled orders, so they were removed completely from the dataset, for analysis sake.
- **Anomaly Treatment:** It was discovered that the data for the month of December, 2011 was incomplete, the data recorded was from December 1 to 10. So this month was removed to avoid been misrepresented.

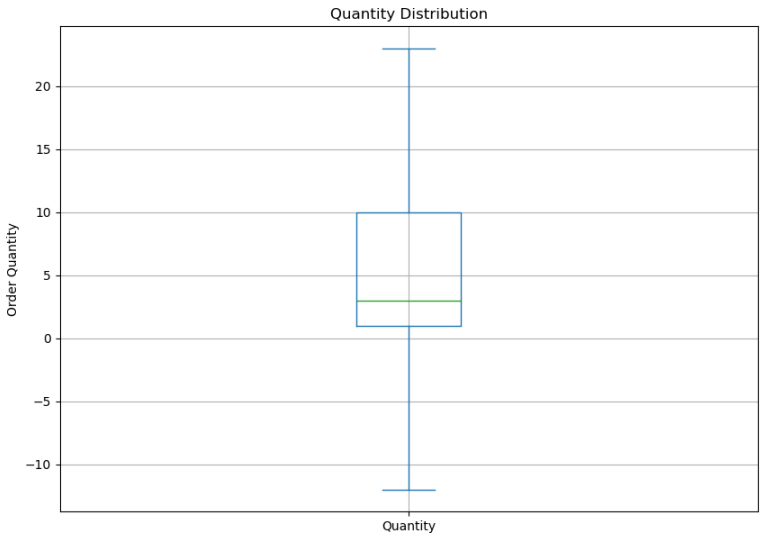


Figure 1: Box Plot of Quantity Distribution

Feature Engineering

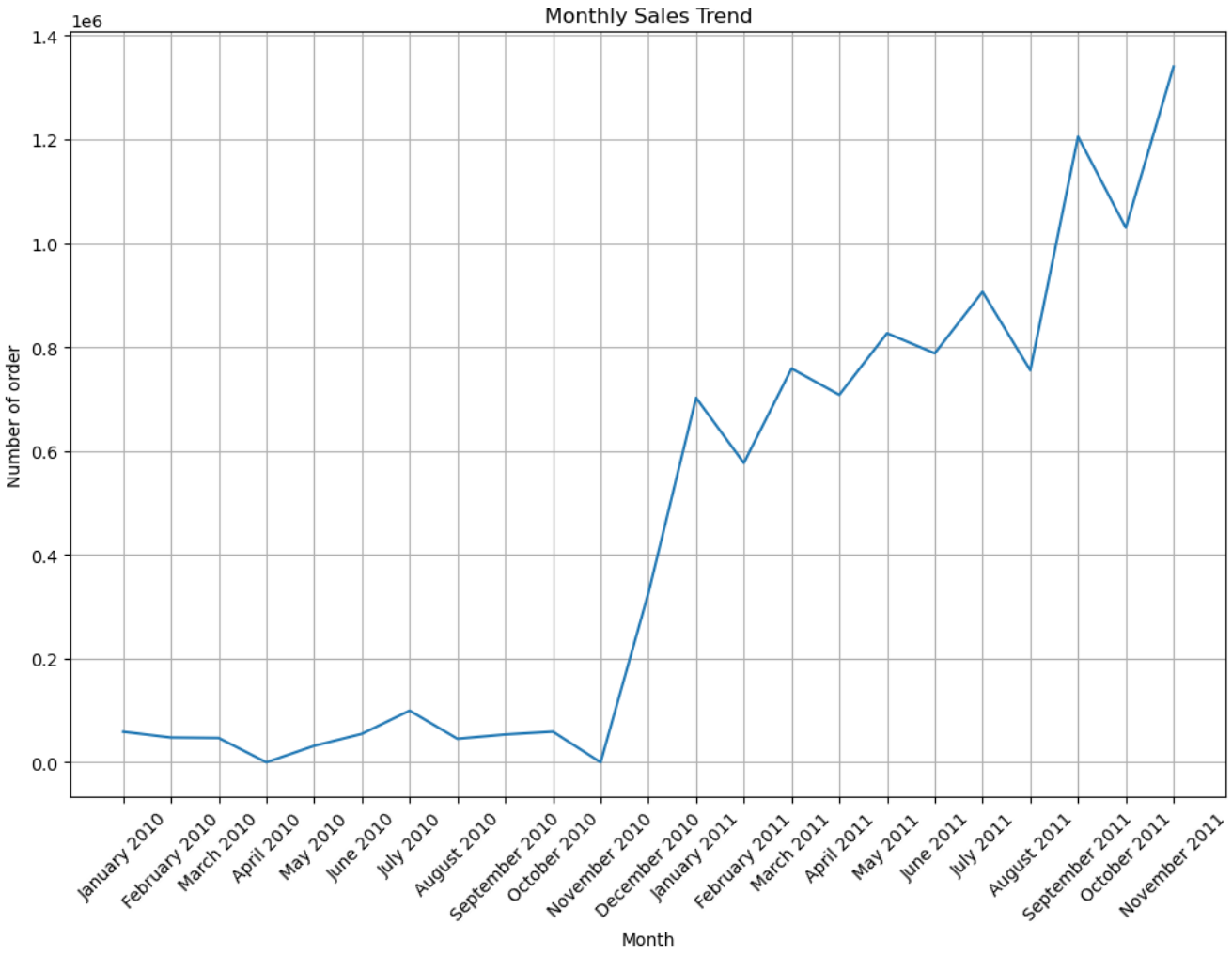
A new column named “TotalPrice” was created to calculate the total price per transaction.

Generated Tables

1. Monthly Revenue Table:
    - Purpose: Captures total revenue for each month
    - Columns:
        i. Month: Year-Month format.
        ii. Total Revenue: Sum of revenue for all transactions in the month.
2. Monthly Transaction Table:
    - Purpose: Captures the Total transaction count for each month.
    - Columns:
        i. Month: Year-Month format.
        ii. Number of Orders: Number of Transaction.
3. Product Category:
    - Purpose: Shows the total quantities of a product category purchased across specific date.
    - Columns:
        i. Invoice Date: Day/month/year format.
        ii. Stock Code: Stock Code of the item.
        iii. Quantity: Total number of the item purchase.
4. Last Month Sorted Table
    - Purpose: Shows the Total quantities of a product category purchased at the end of the time of analysis.
    - Columns:
        i. Stock Code: Stock code of item under consideration.
        ii. Quantity: Total number the item purchased.
5. Top Product Category:
    - Purpose: To group the data by each month and Stock Code, and summing up the quantities sold.
    - Columns:
        i. Invoice Date: year/month/day format.
        ii. Stock Code: Unique Item identifier.
        iii. Quantity: Number of item sold
6. Sustained Product:
    - Purpose: To monitor the growth trend of individual Top performing item.
    - Columns:
        i. Stock Code: A vertical column of the individual item unique identifier.
        ii. Invoice Date: year/month/day format.

Exploratory Data Analysis (EDA)

- **Revenue and Transaction Trends**



The line plot above shows the monthly trend in the number of orders from January 2010 to November 2011. The following insights are observed:

1. **General Growth Trend**:
   - There is a clear upward trend in the number of orders over time. This suggests that the company's customer base or sales volume has been growing steadily across the two-year period.
2. **Key Growth Phases**:
   - **Significant Increase in Late 2010**: Around November 2010, there is a sharp spike in orders, which then settles slightly but remains at a higher level than previous months. This could be attributed to a seasonal event like holiday promotions or an end-of-year sales boost.

- **Another Surge in 2011**: From around June 2011 onward, there is a noticeable and consistent increase in the number of orders, peaking in November 2011. This trend may indicate successful marketing efforts, new product launches, or other factors driving higher demand.

3. **Anomalies and Fluctuations**:
   - **Drop in January 2011**: After the surge in December 2010, there's a drop in January 2011, which could reflect a post-holiday dip, as is common in many retail sectors.
   - **Volatility in Late 2011**: Between July 2011 and November 2011, there are fluctuations, but the overall trend remains upward. These fluctuations might represent intermittent promotional campaigns or stock availability issues.

Insights

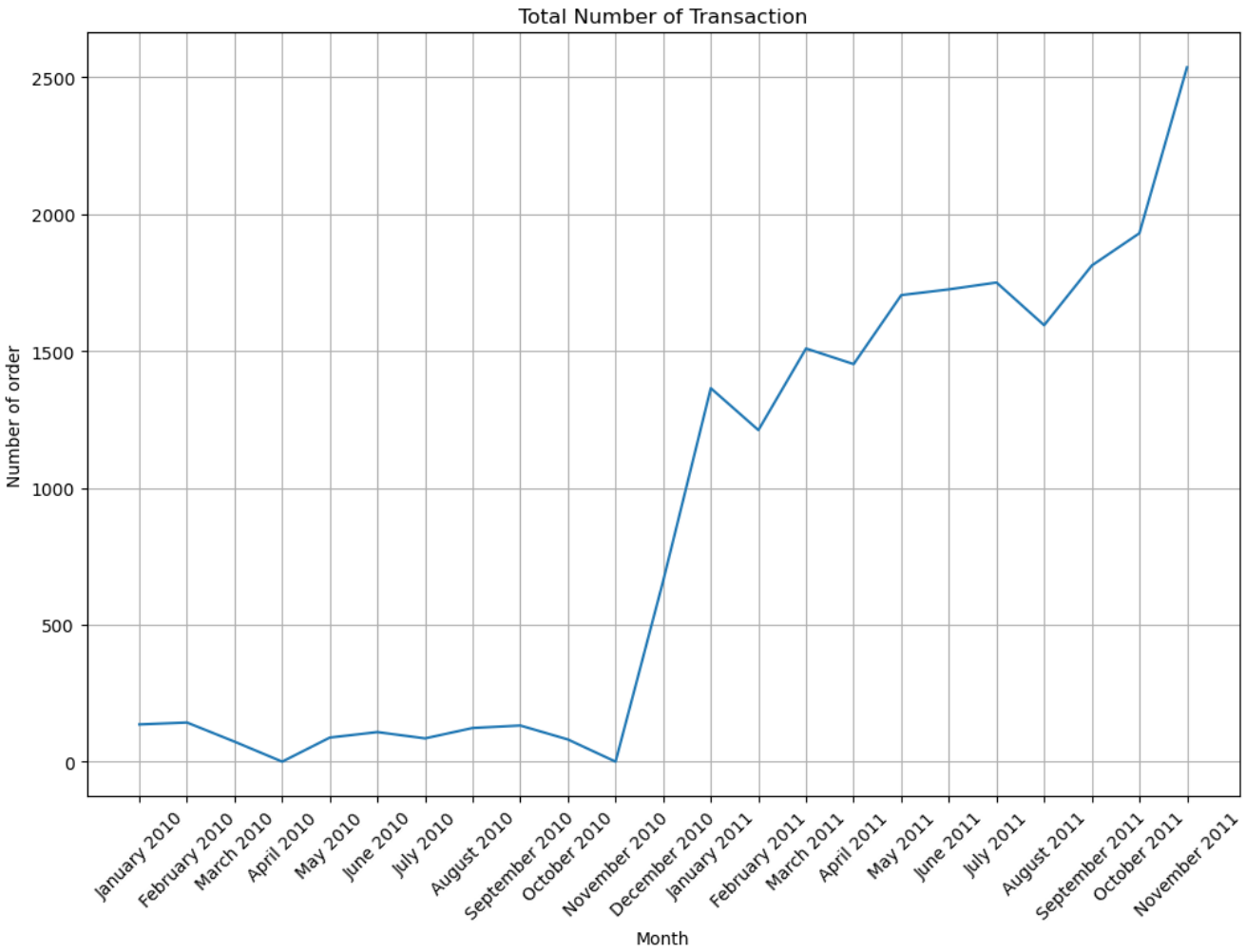
1. Seasonal Patterns: The noticeable increase in orders during December each year suggests that the company experiences a seasonal peak around the holiday season. This insight could help the company plan for inventory and staffing needs for peak months.
2. Sustained Growth: The steady increase in order volume indicates growing demand, which is promising for the business. This growth trend could be leveraged to attract new investments or justify scaling up operations.

Recommendations

Based on the observed trends, the following strategies could be beneficial:

1. Targeted Marketing Campaigns: Consider launching targeted campaigns in the months leading up to December to capitalize on the seasonal peak.
2. Inventory Planning: Ensure sufficient stock levels and logistical support for high-demand periods, especially in Q4.
3. Investigate Fluctuations: Analyze the causes of the fluctuations observed in late 2011 to better understand customer behavior and mitigate potential supply issues.

- Transaction Monthly Trend



The plot above displays the monthly trend in the total number of transactions from January 2010 to November 2011. Several key observations can be made:

1. **Overall Growth in Transactions**:
    - Similar to the revenue trend, there is an overall upward trend in the number of transactions over the two-year period. This steady increase indicates growth in customer activity and engagement over time.
2. **Significant Growth Phases**:
    - **End of 2010 Surge**: There is a noticeable spike in transactions around November 2010, followed by a decline in January 2011, which then stabilizes at a higher level than previous months.
    - **Increased Activity in 2011**: From April 2011 onwards, there is a consistent increase in transaction count, with peaks and troughs throughout, suggesting high engagement but some volatility in customer purchase frequency.
3. **Seasonal Patterns**:

- **Peak in November 2011**: The transactions peak significantly in November 2011, possibly due to holiday shopping or promotional activities leading up to the holiday season.
- **Post-Holiday Drop**: Following the peak at the end of 2010, there is a drop in January 2011, which is consistent with a common trend of lower transaction volumes after the holiday period.

Insights

1. Customer Base Expansion: The increasing number of transactions suggests a growing customer base or an increase in repeat purchases. This trend supports the revenue growth observed in the previous plot.

2. Seasonal Impact on Transactions: The spikes toward the end of each year suggest that the holiday season has a significant impact on transaction volume. This could be due to seasonal promotions, holiday shopping, or increased marketing efforts.

Recommendations

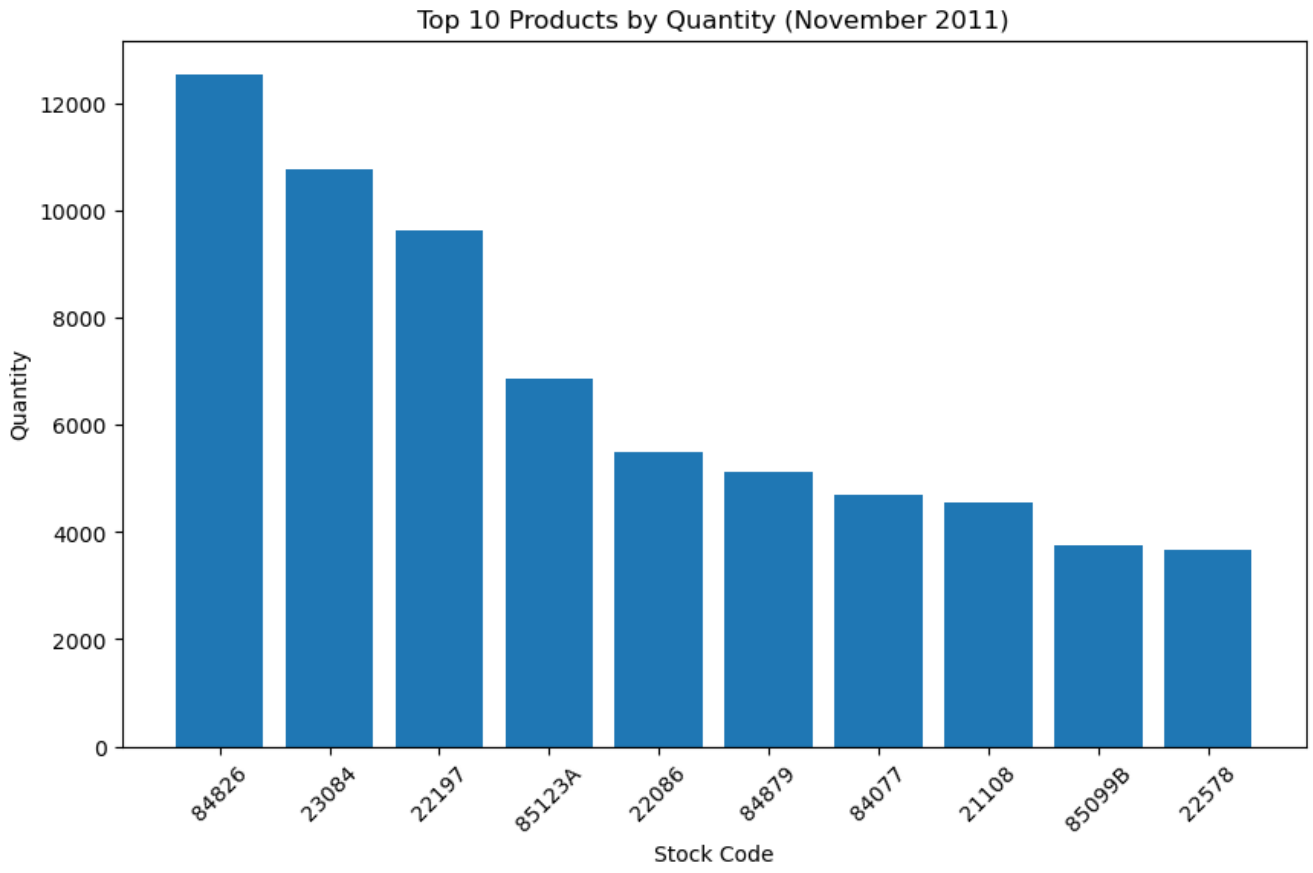
1. Seasonal Promotions: Consider implementing targeted promotional campaigns in October or early November to maximize sales during peak months.

2. Customer Retention: Analyze repeat customer data to identify high-value customers and potentially implement loyalty programs to encourage continued engagement after peak seasons.

3. Investigate Volatility in 2011: The fluctuations throughout 2011 indicate that transactions may be influenced by external factors or intermittent promotions. Further analysis of these peaks and troughs could reveal additional opportunities for optimizing customer engagement.

Product Category Analysis

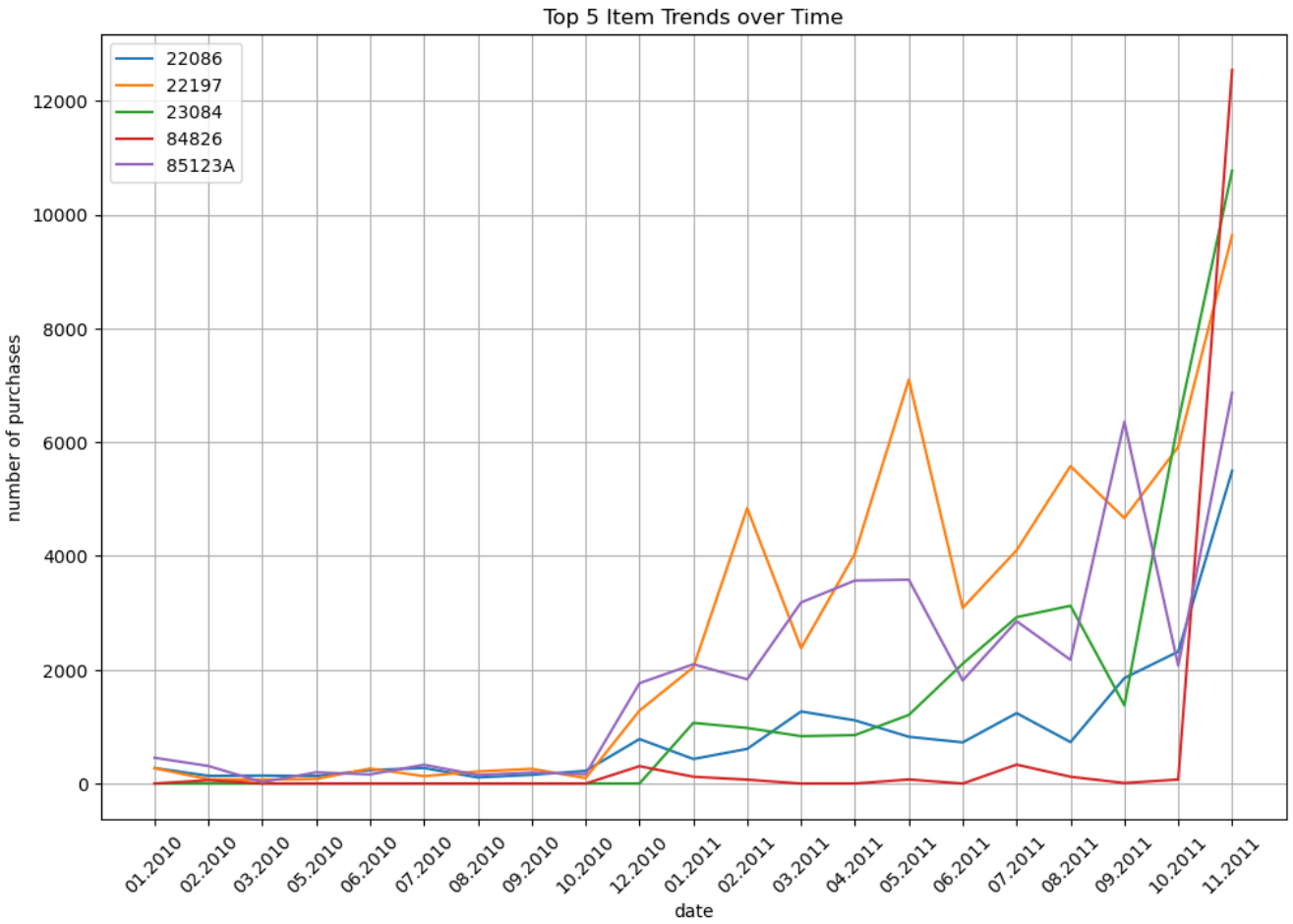
Top-performing categories by revenue



As you can see from this result, the products with the codes 84826, 23084,

22197, 85123A, and 22086 were the top five best-sellers in the month of

November 2011.



Let's examine this time series plot more closely. The sales of these five products surged in November 2011, particularly for the product with the stock code 84826, which had seen minimal sales from February to October 2011 before experiencing a significant spike in November. It may be beneficial to investigate what caused this increase. This could be due to seasonal demand, where the item becomes particularly popular in November, or it might reflect a genuine shift in trends that has made the product more appealing.

In contrast, the popularity of the other top five products—22086, 22197, 23084, and 85123A—appears to have gradually increased in the months leading up to November 2011. As a marketer, it would be valuable to explore the factors contributing to this rising interest in these items. Investigating whether these products tend to be more sought after during colder months or if there's a growing trend in the market could provide insights.

Understanding the trends and shifts in product popularity not only helps you grasp customer preferences but also enables you to tailor your marketing strategies effectively. For instance, you could highlight these increasingly popular items in your marketing emails, calls, or advertisements to enhance customer engagement. Since customers show greater interest in these trending products, promoting them more could lead to improved marketing engagement and potentially higher conversion rates.

Key Insights from the trend chart

1. Major Surge in Late 2011:

- Multiple items (particularly 84826 and 23084) show a dramatic spike in purchases in November 2011

- This surge is far above previous levels, reaching around 12,000 units for item 84826

2. Different Growth Patterns:

- Item 22197 (orange line) shows more volatility throughout 2011 with several peaks and troughs

- Item 85099B (purple line) shows moderate but consistent performance through most of 2011

- Item 22086 (blue line) maintains the most stable pattern with gradual growth

- Item 84826 (red line) remains relatively flat until the dramatic increase at the end

3. Timing of Changes:

- Similar to the previous graph, there's a notable change in behaviour starting around late 2010/early 2011

- Before this point, all items had relatively low and stable purchase numbers

- After this point, there's increased volatility and generally higher volumes

4. Performance Comparison:

- Items showed different levels of success:

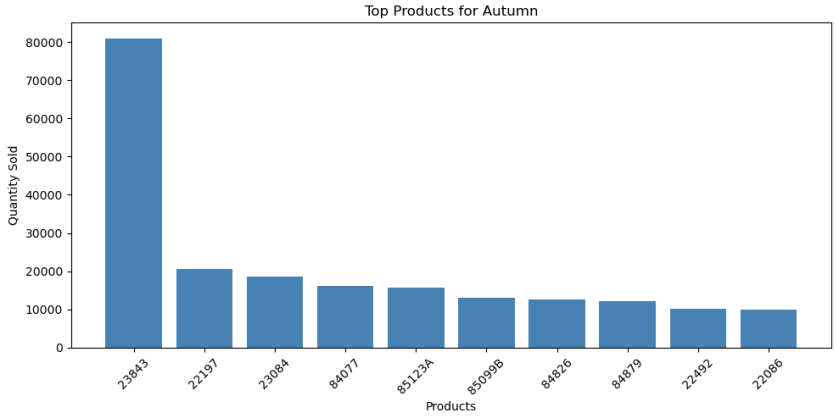
  - Some maintained steady growth

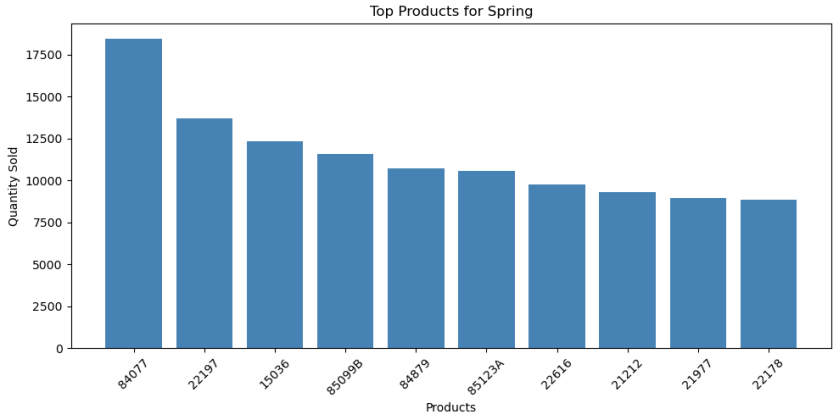
  - Others showed high volatility

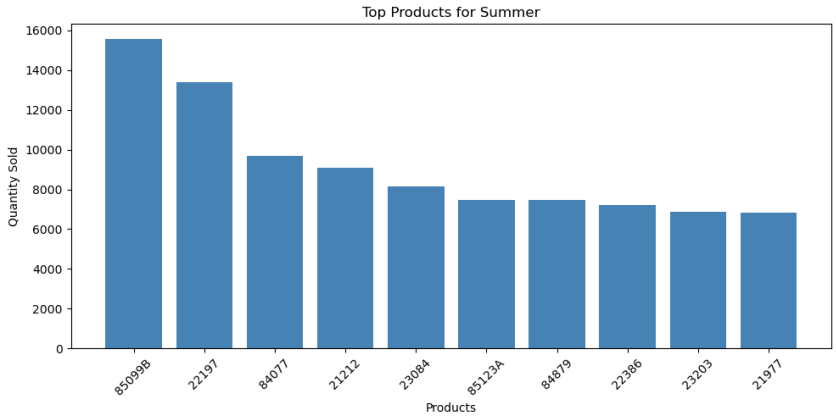
  - A few remained relatively flat until the final surge

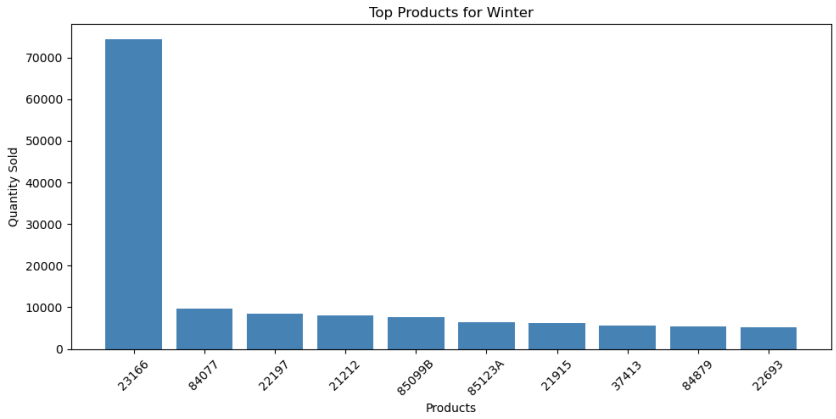
5. The most striking insight is the synchronized sharp uptick in multiple items at the end of the period, suggesting a possible seasonal effect or company-wide event affecting multiple product lines simultaneously.

Seasonal Variation in Customer Demands Across different season









Comprehensive Code Documentation for Ecommerce Sales Data Analysis

This document provides a detailed explanation of every line of code in the sales data analysis project.

---

1. Importing Necessary Libraries

```
#import dependent libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as tick
import seaborn as sns
```

Explanation:

- `pandas` **is used for loading and manipulating the dataset.**
- `numpy` **is employed for efficient numerical operations.**
- `matplotlib.pyplot` **is the foundation for plotting data.**
- `seaborn` **builds on** `matplotlib` **to create more visually appealing plots.**

---

2. Loading the Dataset

```
#Load the data set
df = pd.read_csv('purchase_data.csv',encoding='latin1')
df.head() #Display first 5 rows
```

Explanation:

- **The dataset is loaded using** `read_csv`**.** `encoding='latin1'` **handles special characters in the data.**
- `head()` **is used for an initial inspection of the data structure.**

---

3. Basic Dataset Information

```
df.shape
```

```
(531285, 8)
```

```
df.info()
```

Explanation:

- **`info()` reveals non-null counts and data types for each column, useful for spotting missing data.**
- **`describe()` provides metrics such as mean, standard deviation, and percentiles for numerical columns.**

---

4. Data Cleaning

4.1 Removing Negative Quantities

```
#Removed the the rows with negative value in Quantity
df = df.loc[df['Quantity'] > 0]
```

Explanation:

- **Transactions with negative quantities usually indicate product returns and are excluded to focus on sales.**

---

5. Feature Engineering

5.1 Adding a Total Price Column

```
df['Sales'] = df['Quantity'] * df['UnitPrice'] # Calculate total transaction value for each row
```

Explanation:

- **`TotalPrice` captures the revenue per transaction and is critical for sales analysis.**

---

6. Exploratory Data Analysis (EDA)

6.1 Plotting the Distribution of Quantities

```
plt.figure(figsize=(10, 6))
sns.boxplot(x=df['Quantity'], showfliers=False)  # Boxplot to show quantity distribution without outliers
plt.title('Distribution of Quantity Ordered')
plt.show()
```

Explanation:

- **`showfliers=False` hides outliers to focus on typical order sizes.**
- **`title` adds context to the plot.**

6.2 Time Series Analysis

Monthly Sales Transaction

Table generated

```
monthly_revenue_df = pd.DataFrame(data.set_index('InvoiceDate')['TotalPrice'].resample('M').sum())

# Rename the columns
monthly_revenue_df.columns = ['Total Revenue']

# Reset index to convert the index to a column
monthly_revenue_df.reset_index(inplace=True)

# Format the 'Month' column
monthly_revenue_df['Month'] = monthly_revenue_df['InvoiceDate'].dt.strftime('%B %Y')

# Drop the original 'InvoiceDate' column if not needed
monthly_revenue_df.drop(columns='InvoiceDate', inplace=True)

# Display the DataFrame
monthly_revenue_df
```

Documentation:

1. **Data Aggregation**:
   - This section resamples the `TotalPrice` data by month to calculate monthly revenue.
   - `data.set_index('InvoiceDate')['TotalPrice'].resample('M').sum()` sets `InvoiceDate` as the index and resamples by month ('M') to compute the total revenue for each month.
2. **DataFrame Manipulation**:
   - **Column Renaming**: The resulting DataFrame is renamed to `Total Revenue` for clarity.
   - **Index Reset**: The index is reset to convert `InvoiceDate` back to a column.
   - **Formatting the Month Column**: The `InvoiceDate` is formatted as `'Month Year'` (e.g., 'January 2010') to provide a more readable time format.
   - **Dropping `InvoiceDate`**: If `InvoiceDate` is no longer needed, it is dropped to keep the table clean.
3. **Output**:
   - The `monthly_revenue_df` DataFrame now contains two columns:
     - `Total Revenue`: Monthly revenue for each month in the dataset.
     - `Month`: A formatted string representation of the month and year.

Trend Visualization

```python
#Visualize trend using line graph
ax  = monthly_revenue_df.plot(
    grid = True,
    figsize = (12, 8),
    legend = False
)

ax.set_xlabel('Month')
ax.set_ylabel('Number of Orders')
ax.set_title('Monthly Sales Trend')

plt.xticks(range(len(monthly_revenue_df.index)),
           monthly_revenue_df['Month'],
           rotation = 45
          )
plt.show()
```

Documentation:

1. **Plot Creation**:
    - `monthly_revenue_df.plot()` creates a line plot of the `Total Revenue` column to visualize monthly revenue trends.
    - **Grid and Figure Size**: `grid=True` enables a background grid for easier reading, and `figsize=(12, 8)` sets the plot size for better visibility.
2. **Axes Labels and Title**:
    - `set_xlabel('Month')` and `set_ylabel('Number of Orders')` label the x-axis and y-axis, respectively.
    - `set_title('Monthly Sales Trend')` adds a title to the plot to indicate what is being visualized.
3. **Custom X-Ticks**:
    - `plt.xticks()` is used to format the x-axis labels as the months from the `Month` column, rotated for readability.
4. **Display**:
    - `plt.show()` renders the plot.

Monthly Transaction Trend
Table Generated

```python
monthly_order_df = pd.DataFrame(data.set_index('InvoiceDate')['InvoiceNo'].resample('M').nunique())

# Rename the columns
monthly_order_df.columns = ['Number of Orders']

# Reset index to convert the index to a column
monthly_order_df.reset_index(inplace=True)

# Format the 'Month' column
monthly_order_df['Month'] = monthly_order_df['InvoiceDate'].dt.strftime('%B %Y')

# Drop the original 'InvoiceDate' column if not needed
monthly_order_df.drop(columns='InvoiceDate', inplace=True)

# Display the DataFrame
print(monthly_order_df)
```

Documentation:

1. **Data Aggregation**:
    - This code calculates the monthly count of unique orders by resampling the data.

- `data.set_index('InvoiceDate')['InvoiceNo'].resample('M').nunique()` sets `InvoiceDate` as the index, resamples it by month ('M'), and calculates the number of unique `InvoiceNo` values for each month, representing the total transactions.

2. **DataFrame Manipulation**:
   - **Column Renaming**: The resulting column is renamed to `Number of Orders` for clarity.
   - **Index Reset**: The index is reset to make `InvoiceDate` a regular column.
   - **Formatting the Month Column**: The `InvoiceDate` is formatted as `'Month Year'` (e.g., 'January 2010') for a more readable representation of each period.
   - **Dropping `InvoiceDate`**: The original `InvoiceDate` column is dropped to keep the table organized.
3. **Output**:
   - The `monthly_order_df` DataFrame now contains two columns:
     - `Number of Orders`: Total number of unique transactions for each month.
     - `Month`: A formatted string representing the month and year.

Trend Visualization

```
ax = monthly_order_df.plot(
    grid = True,
    figsize = (12, 8),
    legend = False
)

ax.set_ylabel('Number of order')
ax.set_xlabel('Month')
ax.set_title('Total Number of Transaction')
ax.set_xticks(range(len(monthly_order_df.index)),
              monthly_order_df['Month'],
             rotation = 45)
plt.show()
```

Documentation:

1. **Plot Creation**:
   - `monthly_order_df.plot()` generates a line plot of the `Number of Orders` column to visualize the monthly trend in transaction volume.
   - **Grid and Figure Size**: `grid=True` enables a grid for better readability, and `figsize=(12, 8)` sets the plot dimensions for clarity.
2. **Axes Labels and Title**:
   - `set_ylabel('Number of order')` and `set_xlabel('Month')` label the y-axis and x-axis, respectively.
   - `set_title('Total Number of Transaction')` provides a title to clarify the plot's focus.
3. **Custom X-Ticks**:
   - `ax.set_xticks()` is customized to display the `Month` column values as labels on the x-axis, rotated for improved readability.
4. **Display**:
   - `plt.show()` renders the plot.

6.3 Product Category Analysis

Product with highest Revenue Table

```python
Product_cat_df = pd.DataFrame(
    data.set_index('InvoiceDate').groupby([
        pd.Grouper(freq='M'), 'StockCode'
    ])['Quantity'].sum()
)
Product_cat_df
```

Documentation:

1. **Data Aggregation**:
    - This code groups the data by both month (`InvoiceDate`) and product (`StockCode`) to calculate the total quantity sold for each product by month.
    - `data.set_index('InvoiceDate').groupby([pd.Grouper(freq='M'), 'StockCode'])['Quantity'].sum()`:
        - `pd.Grouper(freq='M')` resamples the data by month.
        - The code then aggregates (`sum()`) the `Quantity` sold for each `StockCode` (product) within each month.
2. **Output**:
    - The resulting `Product_cat_df` DataFrame contains:
        - The month as the index (derived from `InvoiceDate`).
        - `StockCode` as a secondary index.
        - The `Quantity` column, showing the total quantity sold for each product in each month.

Last Month Sorted Table

```python
# Sort Products by the last month sales
last_month_sorted_df = Product_cat_df.loc['2011-11-30'].sort_values(
    by='Quantity', ascending=False
).reset_index()

last_month_sorted_df
```

Documentation:

1. **Last Month's Data Extraction and Sorting**:
    - This section focuses on analyzing the products sold in the last month of the dataset (November 2011).
    - `Product_cat_df.loc['2011-11-30']` extracts sales data for November 2011.
    - `.sort_values(by='Quantity', ascending=False)` sorts the products by quantity sold in descending order to identify the top-selling items.
    - `.reset_index()` resets the index for easier readability.
2. **Output**:
    - The `last_month_sorted_df` DataFrame contains:
        - The product (`StockCode`) and `Quantity` columns, sorted in descending order by quantity sold for November 2011.

Visualizing Top 10 products

```
top_10_products = last_month_sorted_df.head(10)

# Create the bar plot
plt.figure(figsize=(10, 6))
plt.bar(top_10_products['StockCode'], top_10_products['Quantity'])

# Customize the plot
plt.xlabel('Stock Code')
plt.ylabel('Quantity')
plt.title('Top 10 Products by Quantity (November 2011)')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability

plt.show()
```

Documentation:

1. **Selecting Top Products**:
   - `top_10_products = last_month_sorted_df.head(10)` selects the top 10 products by quantity sold from the November 2011 sales data.
2. **Plot Creation**:
   - `plt.bar()` creates a bar plot to visualize the quantities of the top 10 products sold.
   - **Figure Size**: `figsize=(10, 6)` sets the plot dimensions for better visibility.
3. **Axes Labels and Title**:
   - `plt.xlabel('Stock Code')` and `plt.ylabel('Quantity')` label the x-axis and y-axis, respectively.
   - `plt.title('Top 10 Products by Quantity (November 2011)')` adds a title to indicate the specific focus of the plot.
4. **Custom X-Ticks**:
   - `plt.xticks(rotation=45)` rotates the x-axis labels to improve readability of the product codes.
5. **Display**:
   - `plt.show()` renders the plot.

6.4 Monitoring the Product Demand Trend

Growth trend of Products

Sustainability Table generated

```
sustained_prod_df = Top_Product_cat_df.reset_index().pivot('InvoiceDate','StockCode').fillna(0)

sustained_prod_df = sustained_prod_df.reset_index()
sustained_prod_df = sustained_prod_df.set_index('InvoiceDate')
sustained_prod_df.columns = sustained_prod_df.columns.droplevel(0)

sustained_prod_df
```

Documentation:

1. **Pivot Table Creation**:
   - This code reshapes the `Top_Product_cat_df` DataFrame to create a pivot table showing the growth trend of each product (represented by `StockCode`) over time.
   - `Top_Product_cat_df.reset_index().pivot('InvoiceDate','StockCode').fillna(0)`:

- `.pivot('InvoiceDate', 'StockCode')` rearranges the data so that `InvoiceDate` serves as the index and each `StockCode` becomes a separate column.
- `.fillna(0)` replaces any missing values with `0` to ensure a continuous dataset, even if some products were not sold in certain months.

2. **Index and Column Adjustments**:
   - `.reset_index()` resets the index to ensure that `InvoiceDate` is a standard column before setting it as an index again.
   - `.set_index('InvoiceDate')` reassigns `InvoiceDate` as the index for the pivoted DataFrame, representing each month.
   - `.droplevel(0)` removes the extra hierarchical level from column names, simplifying them to just the `StockCode`.
3. **Output**:
   - The resulting `sustained_prod_df` DataFrame provides a monthly view of the quantity sold for each `StockCode`, making it easier to analyze individual product trends over time.

Product Trend Visualized

```
ax = pd.DataFrame(sustained_prod_df.values).plot(
    figsize=(12,8),
    grid=True,
)

ax.set_ylabel('number of purchases')
ax.set_xlabel('date')
ax.set_title('Top 5 Item Trends over Time')

ax.legend(sustained_prod_df.columns, loc='upper left')

plt.xticks(
    range(len(sustained_prod_df.index)),
    [x.strftime('%m.%Y') for x in sustained_prod_df.index],
    rotation=45
)

plt.show()
```

Documentation:

1. **Data Preparation for Plotting**:
   - `pd.DataFrame(sustained_prod_df.values)` creates a DataFrame containing only the values from `sustained_prod_df` to simplify plotting.
2. **Plot Creation**:
   - `.plot(figsize=(12,8), grid=True)` creates a line plot to visualize the monthly growth trends for the top 5 products.
   - **Figure Size**: `figsize=(12,8)` adjusts the plot dimensions for better visibility, while `grid=True` enables grid lines to make the trends easier to follow.
3. **Axes Labels and Title**:
   - `ax.set_ylabel('number of purchases')` and `ax.set_xlabel('date')` label the y-axis and x-axis to indicate the number of purchases and the time period, respectively.
   - `ax.set_title('Top 5 Item Trends over Time')` provides a title that specifies the chart's focus on top product trends.
4. **Legend**:

- `ax.legend(sustained_prod_df.columns, loc='upper left')` adds a legend that displays the `StockCode` for each product line, placed in the upper-left corner for easy identification.

5. **Custom X-Ticks**:
   - `plt.xticks()` formats the x-axis tick labels to show `Month.Year` (e.g., `11.2011`), improving readability.
   - The labels are rotated by 45 degrees to avoid overlap.
6. **Display**:
   - `plt.show()` renders the final plot.

6.5 Customer Preference Across seasons

```python
# Convert InvoiceDate to datetime format and extract seasons
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'], format='%d/%m/%Y %H:%M')
data['Month'] = data['InvoiceDate'].dt.month

# Define seasons based on months
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Autumn'

data['Season'] = data['Month'].apply(get_season)

# Group by Season and Description to analyze product sales patterns
seasonal_sales = data.groupby(['Season', 'StockCode'])['Quantity'].sum().reset_index()

# Display top products for each season
print(seasonal_sales.sort_values(by='Quantity', ascending=False).head(10))

# Aggregate sales quantity by season for visualization
seasonal_summary = data.groupby('Season')['Quantity'].sum().reindex(['Winter', 'Spring', 'Summer', 'Autumn'])
```

Documentation:

1. **Date Conversion and Season Extraction**:
   - `data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'], format='%d/%m/%Y %H:%M')` converts the `InvoiceDate` column to a datetime format for easier date manipulation.
   - `data['Month'] = data['InvoiceDate'].dt.month` extracts the month from each invoice date, allowing for seasonal categorization.
2. **Defining Seasons**:
   - The `get_season` function maps each month to a season:
     - **Winter**: December, January, February
     - **Spring**: March, April, May
     - **Summer**: June, July, August
     - **Autumn**: September, October, November
   - `data['Season'] = data['Month'].apply(get_season)` applies this function to the `Month` column to create a new `Season` column, classifying each transaction by season.
3. **Grouping and Summarizing Seasonal Sales**:
   - `seasonal_sales = data.groupby(['Season', 'StockCode'])['Quantity'].sum().reset_index()` groups the data by `Season` and

`StockCode` (product code), aggregating the total `Quantity` sold for each product in each season.
    - `seasonal_sales.sort_values(by='Quantity', ascending=False).head(10)` displays the top 10 products based on quantity sold across all seasons.
4. **Seasonal Sales Summary**:
    - `seasonal_summary = data.groupby('Season')['Quantity'].sum().reindex(['Winter', 'Spring', 'Summer', 'Autumn'])` calculates the total quantity sold for each season, organizing the data in a way that facilitates seasonal trend analysis.

Visualization for Top Sales Season

```
# Plot seasonal sales patterns
plt.figure(figsize=(10, 6))
sns.barplot(x=seasonal_summary.index, y=seasonal_summary.values, palette='viridis')
plt.title('Total Sales Quantity by Season', fontsize=16)
plt.xlabel('Season', fontsize=14)
plt.ylabel('Total Quantity Sold', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Documentation:

1. **Plot Creation**:
    - `plt.figure(figsize=(10, 6))` sets the figure size to 10 by 6 inches to make the plot easy to view.
    - `sns.barplot()` uses Seaborn to create a bar plot where:
        - `x=seasonal_summary.index` represents the four seasons on the x-axis.
        - `y=seasonal_summary.values` shows the total quantity sold in each season.
        - `palette='viridis'` applies a color gradient to enhance visual appeal.
2. **Titles and Labels**:
    - `plt.title('Total Sales Quantity by Season', fontsize=16)` sets a title, indicating that the plot visualizes seasonal sales quantities.
    - `plt.xlabel('Season', fontsize=14)` and `plt.ylabel('Total Quantity Sold', fontsize=14)` label the x-axis and y-axis for clarity.
3. **Axis Customization**:
    - `plt.xticks(fontsize=12)` and `plt.yticks(fontsize=12)` adjust the font size for tick labels, ensuring readability.
    - `plt.grid(axis='y', linestyle='--', alpha=0.7)` adds horizontal grid lines to the plot, with a dashed line style (`'--'`) and transparency (`alpha=0.7`) for better visualization of seasonal sales differences.
4. **Display**:
    - `plt.show()` renders the plot.

Top Product Sales per season
Tables for individual Seasons

```python
# Get unique seasons
unique_seasons = data['Season'].unique()

# Iterate over each season and display top products
for season in unique_seasons:
    season_df = seasonal_sales[seasonal_sales['Season'] == season]
    top_products = season_df.sort_values(by='Quantity', ascending=False).head(10)
    print(f"Top Products for {season}")
    print(top_products)
    print()
```

Documentation:

1. **Retrieve Unique Seasons**:
    - `unique_seasons = data['Season'].unique()` obtains a list of all unique seasons present in the data. This allows for iteration over each season to analyze top products individually.
2. **Loop Through Each Season**:
    - `for season in unique_seasons:` iterates over each season in the dataset.
    - `season_df = seasonal_sales[seasonal_sales['Season'] == season]` filters the `seasonal_sales` DataFrame to include only rows for the current season.
3. **Top Products Identification**:
    - `top_products = season_df.sort_values(by='Quantity', ascending=False).head(10)` sorts the products by the total `Quantity` sold in descending order and selects the top 10 products for each season.
    - `print(f"Top Products for {season}")` and `print(top_products)` display the top 10 products for each season, making it easy to see which items performed best seasonally.

Visualizing the result

```python
# Get unique seasons
unique_seasons = seasonal_sales['Season'].unique()

# Set up the subplots
num_seasons = len(unique_seasons)
fig, axes = plt.subplots(num_seasons, 1, figsize=(10, 5 * num_seasons))

# Iterate over each season and display top products in separate plots
for ax, season in zip(axes, unique_seasons):
    season_df = seasonal_sales[seasonal_sales['Season'] == season]
    top_products = season_df.sort_values(by='Quantity', ascending=False).head(10)

    # Plotting
    ax.bar(top_products['StockCode'], top_products['Quantity'], color='steelblue')
    ax.set_title(f'Top Products for {season}')
    ax.set_xlabel('Products')
    ax.set_ylabel('Quantity Sold')
    ax.tick_params(axis='x', rotation=45)

# Adjust Layout
plt.tight_layout()

# Show the plots
plt.show()
```

Documentation:

1. **Set Up Unique Seasons and Subplots**:
    - `unique_seasons = seasonal_sales['Season'].unique()` retrieves the unique seasons from the `seasonal_sales` DataFrame.
    - `fig, axes = plt.subplots(num_seasons, 1, figsize=(10, 5 * num_seasons))` initializes a subplot for each season, setting the figure height to accommodate all plots.
2. **Plotting Top Products for Each Season**:
    - `for ax, season in zip(axes, unique_seasons):` iterates over each subplot axis and season.
    - `season_df = seasonal_sales[seasonal_sales['Season'] == season]` filters the DataFrame to include only records for the current season.
    - `top_products = season_df.sort_values(by='Quantity', ascending=False).head(10)` sorts products by quantity and selects the top 10 for the current season.
3. **Creating Individual Bar Plots**:
    - `ax.bar(top_products['StockCode'], top_products['Quantity'], color='steelblue')` plots the top products for the current season as a bar plot, with `StockCode` on the x-axis and `Quantity` on the y-axis.
    - `ax.set_title(f'Top Products for {season}')` sets the title of each subplot to reflect the season.
    - `ax.set_xlabel('Products')` and `ax.set_ylabel('Quantity Sold')` label the x-axis and y-axis, respectively.
    - `ax.tick_params(axis='x', rotation=45)` rotates the x-axis labels for better readability.
4. **Final Adjustments and Display**:
    - `plt.tight_layout()` adjusts the layout to prevent overlapping of elements in the figure.
    - `plt.show()` displays the final set of bar plots.