📊 Customer Behavior Analysis Project

🎯 Data-Driven Decision Making Analysis

📚 **Course:** IIMK's Professional Certificate in Data Science and Artificial Intelligence for Managers

👨‍🎓 **Student Name:** Lalit Nayyar

📨 **Email ID:** lalitnayyar@gmail.com

📅 **Date:** May 5, 2025

Table of Contents

📑 Project Documentation

Customer Behavior Analysis Project

Course: IIMK's Professional Certificate in Data Science and Artificial Intelligence for Managers
Student Name: Lalit Nayyar
Email ID: lalitnayyar@gmail.com
Assignment: Data-Driven Decision Making Analysis

📋 Project Overview

This project demonstrates the application of data-driven decision-making concepts to analyze customer behavior in an online retail platform. The analysis progresses through multiple stages of analytics maturity, from descriptive to predictive, providing actionable insights for business decisions.

📊 Dataset Description

- **Source**: Online Retail Dataset
- **Type**: Structured transactional data
- **Time Period**: 2010-2011
- **Key Features**:
- InvoiceNo: Transaction identifier
- StockCode: Product code
- Description: Product name
- Quantity: Items per transaction
- InvoiceDate: Transaction timestamp
- UnitPrice: Price per unit
- CustomerID: Unique customer identifier
- Country: Customer's country

🔍 Analysis Structure

1. Data Preprocessing Notebook (`LalitNayyarIIMKMod4_analysis_fin.ipynb`)

Learning Outcome: Data Description, Preprocessing and Cleaning (4 points) - **Features**: - Comprehensive data quality assessment - Missing value analysis and handling - Duplicate detection and removal - Data type validation and conversion - Outlier detection and treatment - **Key Functions**: - `clean_data()`: Complete data cleaning pipeline - `validate_data_types()`: Data type verification - `handle_outliers()`: Outlier treatment - **Usage Guide**: 1. Load the raw dataset 2. Execute data quality checks 3. Apply cleaning functions 4. Validate cleaned dataset 5. Export processed data for analysis

2. Descriptive Analytics Notebook (`LalitNayyarIIMKMod4_descriptive_analysis_fin.ipynb`)

Learning Outcome: Descriptive Analytics (2 points) - **Analysis Components**: - Purchase patterns analysis - Product popularity metrics - Customer segmentation - Sales trend analysis - **Visualizations**: - Time series plots of sales trends - Product performance heatmaps - Customer segment distributions - Geographic sales analysis - **Key Insights**: - Top-performing products - Peak sales periods - Customer buying patterns - Regional performance metrics

3. Diagnostic Analytics Notebook (`LalitNayyarIIMKMod4_diagnostic_analysis_fin.ipynb`)

Learning Outcome: Diagnostic Analytics (2 points) - **Analysis Methods**: - Correlation analysis - Factor analysis - Root cause investigation - Pattern attribution - **Key Components**: - Price sensitivity analysis - Customer churn factors - Seasonal impact assessment - Product affinity analysis - **Business Insights**: - Churn drivers identification - Sales pattern explanations - Customer behavior factors - Performance variance analysis

4. Predictive Analytics Notebook (`LalitNayyarIIMKMod4_predictive_analysis_fin.ipynb`)

Learning Outcome: Predictive Analytics (2 points) - **Models Implemented**: - Customer Lifetime Value prediction - Purchase frequency forecasting - Product demand prediction - Churn risk assessment - **Technical Components**: - Model selection justification - Feature engineering process - Performance metrics analysis - Prediction reliability assessment - **Business Applications**: - Revenue forecasting - Inventory optimization - Customer retention strategies - Targeted marketing recommendations

Execution Instructions

Environment Setup

```python

# Required packages

pip install pandas numpy matplotlib seaborn scikit-learn jupyter ```

### Running the Analysis

1. **Data Preprocessing**:
2. Open `LalitNayyar_Data_Preprocessing.ipynb`
3. Run all cells sequentially

4. Verify cleaned data output

5. **Descriptive Analysis**:

6. Open `LalitNayyar_Descriptive_Analytics.ipynb`
7. Ensure cleaned data is available
```

8. Execute all cells
9. Review visualization outputs
10. **Diagnostic Analysis**:
11. Open `LalitNayyar_Diagnostic_Analytics.ipynb`
12. Run correlation analyses
13. Review factor analysis results
14. Generate insight reports
15. **Predictive Analysis**:
16. Open `LalitNayyar_Predictive_Analytics.ipynb`
17. Execute model training
18. Validate predictions
19. Review business recommendations

Submission Components

1. **Notebooks**:
2. All four analysis notebooks
3. Properly documented with markdown
4. Clear code comments
5. Complete output cells
6. **Documentation**:
7. This README.md file
8. Analysis methodology explanation
9. Results interpretation
10. Business recommendations
11. **Data Files**:
12. Original dataset
13. Cleaned dataset
14. Intermediate analysis outputs
15. **Presentation**:
16. Key findings summary
17. Visualization highlights
18. Actionable insights
19. Strategic recommendations

📝 Assessment Criteria

Alignment - **Data Processing (4 pts)**: Comprehensive data cleaning and preparation - **Descriptive Analytics (2 pts)**: Thorough pattern analysis and visualization - **Diagnostic Analytics (2 pts)**: In-depth causation analysis - **Predictive Analytics (2 pts)**: Advanced modeling and forecasting

Author's Note

This analysis demonstrates the practical application of data science concepts to real-world business problems, providing actionable insights for decision-making.

🔍 Initial Data Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Customer Behavior Analysis

Course: IIMK's Professional Certificate in Data Science and Artificial Intelligence for Managers
Student Name: Lalit Nayyar
Email ID: lalitnayyar@gmail.com
Assignment Name: Week 4: Required Assignment 4.1

Introduction

This notebook analyzes customer behavior data from an online retail platform to derive meaningful insights for business decision-making. We'll focus on understanding purchasing patterns, customer segmentation, and transaction trends.

Data Description and Preparation

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set display options
pd.set_option('display.max_columns', None)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

```
# Load and examine the data
try:
    # Load data
    print("Loading data...")
    df = pd.read_excel('Online Retail.xlsx')

    # Display basic information
    print("Dataset Info:")
    print(f"Number of records: {len(df):,}")
    print(f"Number of columns: {len(df.columns)}")
    print("Columns:", df.columns.tolist())
```

```
    # Display sample
    print("Sample of the data:")
    display(df.head())

    # Basic statistics
    print("Basic statistics:")
    display(df.describe())

except Exception as e:
    print(f"Error loading data: {e}")
    df = None
```

```
Loading data...
Dataset Info:
Number of records: 541,909
Number of columns: 8
Columns: ['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate', 'UnitPrice',
'CustomerID', 'Country']
Sample of the data:
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.550	17850.000	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.750	17850.000	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom

```
Basic statistics:
```

	Quantity	InvoiceDate	UnitPrice	CustomerID
count	541909.000	541909	541909.000	406829.000
mean	9.552	2011-07-04 13:34:57.156386048	4.611	15287.691
min	-80995.000	2010-12-01 08:26:00	-11062.060	12346.000
25%	1.000	2011-03-28 11:34:00	1.250	13953.000
50%	3.000	2011-07-19 17:17:00	2.080	15152.000
75%	10.000	2011-10-19 11:27:00	4.130	16791.000
max	80995.000	2011-12-09 12:50:00	38970.000	18287.000
std	218.081	NaN	96.760	1713.600

Data Structure Analysis

The Online Retail dataset is a **structured dataset** with the following characteristics:

- Each row represents a transaction
- Contains numerical and categorical variables
- Has a clear schema with defined columns

```
# Check for missing values
print("Missing values in each column:")
print("-" * 50)
print(df.isnull().sum())

# Check for duplicates
print("\nNumber of duplicate rows:")
print("-" * 50)
print(df.duplicated().sum())
```

```
Missing values in each column:
--------------------------------------------------
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64

Number of duplicate rows:
--------------------------------------------------
5268
```

```
def clean_data(df):
    # Create a copy of the dataframe
    df_clean = df.copy()

    # Remove rows with missing values
    df_clean = df_clean.dropna()

    # Remove duplicates
    df_clean = df_clean.drop_duplicates()

    # Filter out rows with quantity <= 0 or unit price <= 0
    df_clean = df_clean[(df_clean['Quantity'] > 0) & (df_clean['UnitPrice'] > 0)]

    # Add a TotalAmount column
    df_clean['TotalAmount'] = df_clean['Quantity'] * df_clean['UnitPrice']

    return df_clean

# Clean the data
df_clean = clean_data(df)

# Display basic statistics of the cleaned dataset
print("Cleaned dataset statistics:")
```

```
print("-" * 50)
df_clean.describe()
```

```
Cleaned dataset statistics:
--------------------------------------------------
```

	Quantity	InvoiceDate	UnitPrice	CustomerID	TotalAmount
count	392692.000	392692	392692.000	392692.000	392692.000
mean	13.120	2011-07-10 19:13:07.771892480	3.126	15287.844	22.631
min	1.000	2010-12-01 08:26:00	0.001	12346.000	0.001
25%	2.000	2011-04-07 11:12:00	1.250	13955.000	4.950
50%	6.000	2011-07-31 12:02:00	1.950	15150.000	12.450
75%	12.000	2011-10-20 12:53:00	3.750	16791.000	19.800
max	80995.000	2011-12-09 12:50:00	8142.750	18287.000	168469.600
std	180.493	NaN	22.242	1713.540	311.099

Data Preprocessing Summary

The following preprocessing steps were performed:

1. Removed missing values
2. Removed duplicate transactions
3. Filtered out invalid transactions (negative or zero quantity/price)
4. Added TotalAmount column for transaction value analysis

The cleaned dataset is now ready for further analysis of customer behavior patterns.

📊 Descriptive Analysis

Customer Behavior Descriptive Analysis

Course: IIMK's Professional Certificate in Data Science and Artificial Intelligence for Managers
Student Name: Lalit Nayyar
Email ID: lalitnayyar@gmail.com
Assignment Name: Week 4: Required Assignment 4.1

Descriptive Analytics Section

In this section, we'll perform detailed descriptive analytics to understand customer behavior patterns and trends.

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

# Set visualization style

plt.style.use('seaborn-v0_8')
plt.rcParams['figure.figsize'] = [14, 7]
plt.rcParams['axes.grid'] = True
plt.rcParams['grid.alpha'] = 0.3
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.titlesize'] = 14
plt.rcParams['xtick.labelsize'] = 11
plt.rcParams['ytick.labelsize'] = 11
plt.rcParams['lines.linewidth'] = 2.0
plt.rcParams['font.size'] = 11
plt.rcParams['axes.prop_cycle'] = plt.cycler(color=[
    '#1A237E',  # Dark blue
    '#BF360C',  # Dark orange
    '#1B5E20',  # Dark green
    '#4A148C',  # Dark purple
    '#212121',  # Near black
    '#01579B'   # Navy blue
])
plt.rcParams['axes.edgecolor'] = '#212121'
plt.rcParams['axes.labelcolor'] = '#212121'
plt.rcParams['xtick.color'] = '#212121'
plt.rcParams['ytick.color'] = '#212121'
  # Use default matplotlib style
sns.set_theme(style="whitegrid")  # Set seaborn style
sns.set_palette('husl')  # Set color palette

# Increase font size for better readability
plt.rcParams['font.size'] = 12
plt.rcParams['axes.labelsize'] = 12
```

```
plt.rcParams['axes.titlesize'] = 14
plt.rcParams['xtick.labelsize'] = 10
plt.rcParams['ytick.labelsize'] = 10

# Display settings for pandas
pd.set_option('display.max_columns', None)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

```
def clean_data(df):
    """
    Clean the retail dataset by:
    1. Removing missing values
    2. Removing cancelled orders (those with 'C' in InvoiceNo)
    3. Ensuring positive quantities and prices
    4. Converting InvoiceDate to datetime
    """
    if df is None:
        return None

    # Create a copy of the dataframe
    df_clean = df.copy()

    # Remove rows with missing values
    df_clean = df_clean.dropna()

    # Remove cancelled orders (those with 'C' in InvoiceNo)
    df_clean = df_clean[~df_clean['InvoiceNo'].astype(str).str.contains('C')]

    # Ensure positive quantities and prices
    df_clean = df_clean[(df_clean['Quantity'] > 0) & (df_clean['UnitPrice'] > 0)]

    # Convert InvoiceDate to datetime if it's not already
    if not pd.api.types.is_datetime64_any_dtype(df_clean['InvoiceDate']):
        df_clean['InvoiceDate'] = pd.to_datetime(df_clean['InvoiceDate'])

    # Reset index
    df_clean = df_clean.reset_index(drop=True)

    print("Data cleaning summary:")
    print(f"Original records: {len(df)}")
    print(f"Clean records: {len(df_clean)}")
    print(f"Removed records: {len(df) - len(df_clean)}")

    return df_clean
```

1. Purchase Frequency Analysis

```
# Load and prepare the data
df = pd.read_excel('Online Retail.xlsx')
print("Original data shape:", df.shape)

# Clean the data
df_clean = clean_data(df)
print("Cleaned data shape:", df_clean.shape)

# Calculate total amount for each transaction
df_clean['TotalAmount'] = df_clean['Quantity'] * df_clean['UnitPrice']
print("Sample of cleaned data with total amount:")
```

```
display(df_clean.head())

# Basic statistics of the cleaned dataset
print("Basic statistics of numerical columns:")
display(df_clean.describe())
```

```
Original data shape: (541909, 8)
Data cleaning summary:
Original records: 541909
Clean records: 397884
Removed records: 144025
Cleaned data shape: (397884, 8)
Sample of cleaned data with total amount:
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total/
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.550	17850.000	United Kingdom	
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.750	17850.000	United Kingdom	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom	

```
Basic statistics of numerical columns:
```

	Quantity	InvoiceDate	UnitPrice	CustomerID	TotalAmount
count	397884.000	397884	397884.000	397884.000	397884.000
mean	12.988	2011-07-10 23:41:23.511023360	3.116	15294.423	22.397
min	1.000	2010-12-01 08:26:00	0.001	12346.000	0.001
25%	2.000	2011-04-07 11:12:00	1.250	13969.000	4.680
50%	6.000	2011-07-31 14:39:00	1.950	15159.000	11.800
75%	12.000	2011-10-20 14:33:00	3.750	16795.000	19.800
max	80995.000	2011-12-09 12:50:00	8142.750	18287.000	168469.600
std	179.332	NaN	22.098	1713.142	309.071

2. Popular Products Analysis

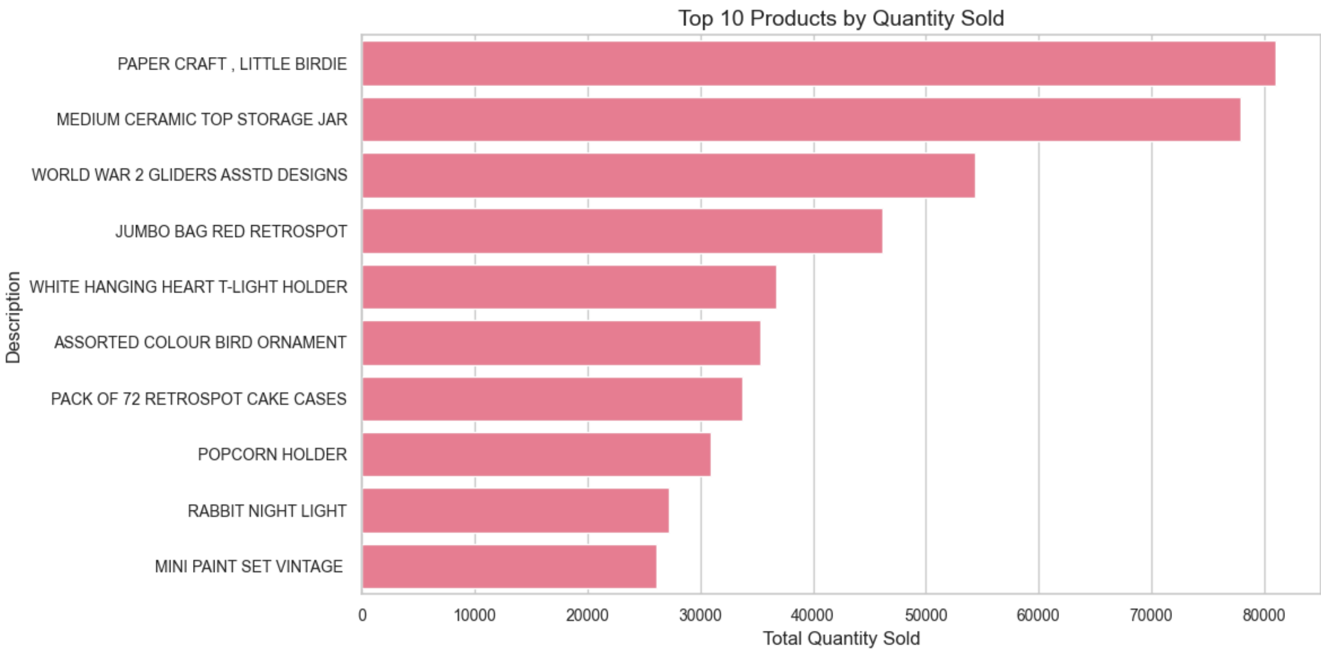
```
# Analyze top selling products
product_sales = df_clean.groupby('Description').agg({
    'Quantity': 'sum',
    'TotalAmount': 'sum',
    'InvoiceNo': 'count'
}).rename(columns={'InvoiceNo': 'TransactionCount'})

# Sort by quantity sold
top_products_by_quantity = product_sales.sort_values('Quantity', ascending=False).head(10)

# Display top products by quantity
print("Top 10 Products by Quantity Sold:")
display(top_products_by_quantity)

# Sort by total amount
top_products_by_amount = product_sales.sort_values('TotalAmount', ascending=False).head(10)

# Display top products by revenue
print("Top 10 Products by Revenue:")
display(top_products_by_amount)

# Visualize top products by quantity
plt.figure(figsize=(12, 6))
sns.barplot(x='Quantity', y=top_products_by_quantity.index, data=top_products_by_quantity)
plt.title('Top 10 Products by Quantity Sold')
plt.xlabel('Total Quantity Sold')
plt.tight_layout()
plt.show()

# Visualize top products by revenue
plt.figure(figsize=(12, 6))
sns.barplot(x='TotalAmount', y=top_products_by_amount.index, data=top_products_by_amount)
plt.title('Top 10 Products by Revenue')
plt.xlabel('Total Revenue')
plt.tight_layout()
plt.show()
```

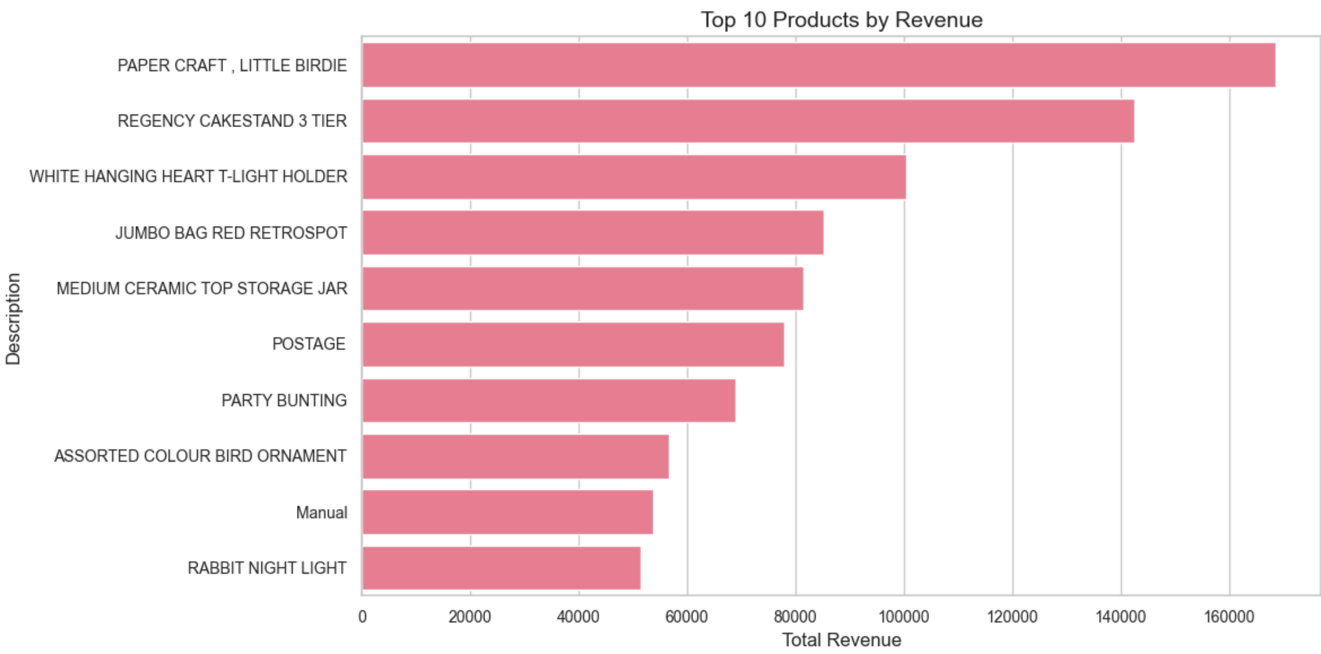
```
Top 10 Products by Quantity Sold:
```

Description	Quantity	TotalAmount	TransactionCount
PAPER CRAFT , LITTLE BIRDIE	80995	168469.600	1
MEDIUM CERAMIC TOP STORAGE JAR	77916	81416.730	198
WORLD WAR 2 GLIDERS ASSTD DESIGNS	54415	13586.250	473
JUMBO BAG RED RETROSPOT	46181	85220.780	1618
WHITE HANGING HEART T-LIGHT HOLDER	36725	100448.150	2028
ASSORTED COLOUR BIRD ORNAMENT	35362	56580.340	1408
PACK OF 72 RETROSPOT CAKE CASES	33693	16394.530	1068
POPCORN HOLDER	30931	23427.710	657
RABBIT NIGHT LIGHT	27202	51346.200	842
MINI PAINT SET VINTAGE	26076	16039.240	325

```
Top 10 Products by Revenue:
```

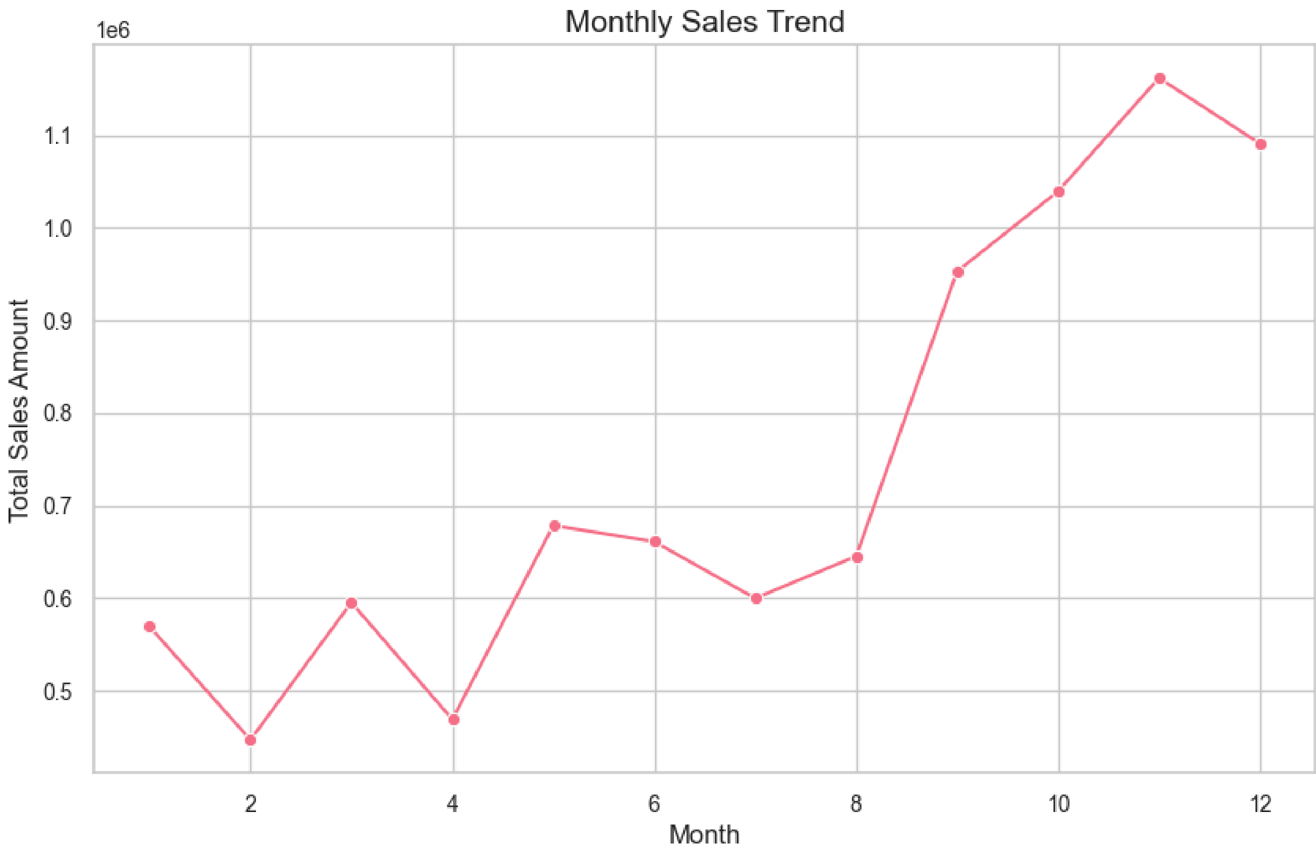
Description	Quantity	TotalAmount	TransactionCount
PAPER CRAFT , LITTLE BIRDIE	80995	168469.600	1
REGENCY CAKESTAND 3 TIER	12402	142592.950	1723
WHITE HANGING HEART T-LIGHT HOLDER	36725	100448.150	2028
JUMBO BAG RED RETROSPOT	46181	85220.780	1618
MEDIUM CERAMIC TOP STORAGE JAR	77916	81416.730	198
POSTAGE	3120	77803.960	1099
PARTY BUNTING	15291	68844.330	1396
ASSORTED COLOUR BIRD ORNAMENT	35362	56580.340	1408
Manual	7173	53779.930	284
RABBIT NIGHT LIGHT	27202	51346.200	842





3. Temporal Purchase Patterns

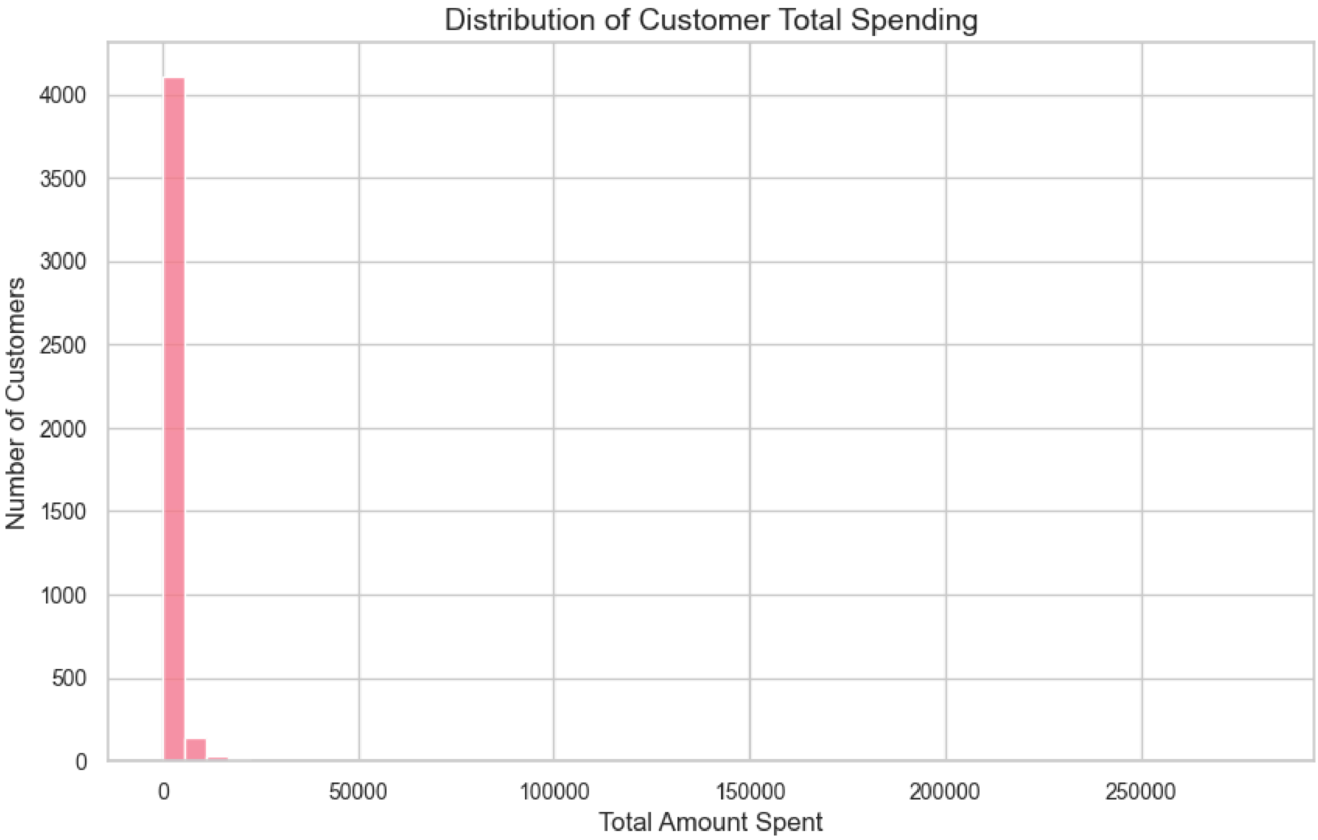
```
# Add datetime components
df_clean['InvoiceDate'] = pd.to_datetime(df_clean['InvoiceDate'])
df_clean['Month'] = df_clean['InvoiceDate'].dt.month
df_clean['DayOfWeek'] = df_clean['InvoiceDate'].dt.day_name()
df_clean['Hour'] = df_clean['InvoiceDate'].dt.hour

# Monthly sales trend
monthly_sales = df_clean.groupby('Month')['TotalAmount'].sum().reset_index()

plt.figure(figsize=(10, 6))
sns.lineplot(data=monthly_sales, x='Month', y='TotalAmount', marker='o')
plt.title('Monthly Sales Trend')
plt.xlabel('Month')
plt.ylabel('Total Sales Amount')
plt.show()
```



4. Customer Spending Analysis

```
# Calculate customer spending metrics
customer_spending = df_clean.groupby('CustomerID').agg({
    'TotalAmount': ['sum', 'mean', 'count'],
    'Quantity': 'sum'
}).round(2)

customer_spending.columns = ['TotalSpent', 'AverageTransactionValue', 'TransactionCount', 'To

print("Customer Spending Statistics:")
print("-" * 50)
print(customer_spending.describe())

# Visualize distribution of customer spending
plt.figure(figsize=(10, 6))
sns.histplot(data=customer_spending, x='TotalSpent', bins=50)
plt.title('Distribution of Customer Total Spending')
plt.xlabel('Total Amount Spent')
plt.ylabel('Number of Customers')
plt.show()
```

```
Customer Spending Statistics:
--------------------------------------------------
       TotalSpent  AverageTransactionValue  TransactionCount  TotalItems
count    4338.000                 4338.000          4338.000    4338.000
mean     2054.266                   68.350            91.721    1191.289
std      8989.230                 1467.919           228.785    5046.082
min         3.750                    2.100             1.000       1.000
25%       307.415                   12.370            17.000     160.000
50%       674.485                   17.725            41.000     379.000
75%      1661.740                   24.858           100.000     992.750
max    280206.020                77183.600          7847.000  196915.000
```



5. Key Insights Summary

Based on the descriptive analytics performed above, we can identify the following key patterns and trends:

What:

- Most popular products and their sales volumes
- Distribution of transaction values
- Customer spending patterns

Which:

- Which products are bestsellers
- Which months show highest sales
- Which customers are most valuable (by spending)

How Many:

- Average purchases per customer
- Total transactions per product
- Distribution of order quantities

🔬 Behavior Diagnostic Analysis

```python
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from datetime import datetime
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Set visualization style

plt.style.use('seaborn-v0_8')
plt.rcParams['figure.figsize'] = [14, 7]
plt.rcParams['axes.grid'] = True
plt.rcParams['grid.alpha'] = 0.3
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.titlesize'] = 14
plt.rcParams['xtick.labelsize'] = 11
plt.rcParams['ytick.labelsize'] = 11
plt.rcParams['lines.linewidth'] = 2.0
plt.rcParams['font.size'] = 11
plt.rcParams['axes.prop_cycle'] = plt.cycler(color=[
    '#1A237E',  # Dark blue
    '#BF360C',  # Dark orange
    '#1B5E20',  # Dark green
    '#4A148C',  # Dark purple
    '#212121',  # Near black
    '#01579B'   # Navy blue
])
plt.rcParams['axes.edgecolor'] = '#212121'
plt.rcParams['axes.labelcolor'] = '#212121'
plt.rcParams['xtick.color'] = '#212121'
plt.rcParams['ytick.color'] = '#212121'

sns.set_theme(style="whitegrid")
sns.set_palette('husl')

# Display settings
pd.set_option('display.max_columns', None)
pd.set_option('display.float_format', lambda x: '%.3f' % x)

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Customer Behavior Diagnostic Analysis

Course: IIMK's Professional Certificate in Data Science and Artificial Intelligence for Managers
Student Name: Lalit Nayyar

Email ID: lalitnayyar@gmail.com

Assignment Name: Week 4: Required Assignment 4.1

Understanding the 'Why' Behind Customer Behavior Trends

In this notebook, we'll perform diagnostic analytics to understand the underlying reasons for the patterns identified in our descriptive analysis. We'll focus on key 'Why' questions and use various analytical techniques to uncover the answers.

```
try:
    # Load the data
    print("Loading data...")
    df = pd.read_excel('Online Retail.xlsx')
    print("Original data shape:", df.shape)

    def clean_data(df):
        """Clean the retail dataset"""
        print("Cleaning data...")
        df_clean = df.copy()

        # Remove missing values
        df_clean = df_clean.dropna()
        print(f"After removing missing values: {len(df_clean)} records")

        # Remove cancelled orders
        df_clean = df_clean[~df_clean['InvoiceNo'].astype(str).str.contains('C')]
        print(f"After removing cancelled orders: {len(df_clean)} records")

        # Ensure positive quantities and prices
        df_clean = df_clean[(df_clean['Quantity'] > 0) & (df_clean['UnitPrice'] > 0)]
        print(f"After ensuring positive values: {len(df_clean)} records")

        # Convert InvoiceDate to datetime
        df_clean['InvoiceDate'] = pd.to_datetime(df_clean['InvoiceDate'])

        # Calculate total amount
        df_clean['TotalAmount'] = df_clean['Quantity'] * df_clean['UnitPrice']

        return df_clean.reset_index(drop=True)

    # Clean the data
    df_clean = clean_data(df)

    print("\nData cleaning complete!")
    print("Final data shape:", df_clean.shape)
    print("\nSample of cleaned data:")
    display(df_clean.head())

    # Display basic statistics
    print("\nBasic statistics of numerical columns:")
    display(df_clean.describe())

except FileNotFoundError:
    print("Error: 'Online Retail.xlsx' file not found. Please ensure it's in the correct dire
```

```
except Exception as e:
    print(f"Error during data preparation: {e}")
```

```
Loading data...
Original data shape: (541909, 8)
Cleaning data...
After removing missing values: 406829 records
After removing cancelled orders: 397924 records
After ensuring positive values: 397884 records

Data cleaning complete!
Final data shape: (397884, 9)

Sample of cleaned data:
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total/
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.550	17850.000	United Kingdom	
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.750	17850.000	United Kingdom	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.390	17850.000	United Kingdom	

```
Basic statistics of numerical columns:
```

	Quantity	InvoiceDate	UnitPrice	CustomerID	TotalAmount
count	397884.000	397884	397884.000	397884.000	397884.000
mean	12.988	2011-07-10 23:41:23.511023360	3.116	15294.423	22.397
min	1.000	2010-12-01 08:26:00	0.001	12346.000	0.001
25%	2.000	2011-04-07 11:12:00	1.250	13969.000	4.680
50%	6.000	2011-07-31 14:39:00	1.950	15159.000	11.800
75%	12.000	2011-10-20 14:33:00	3.750	16795.000	19.800
max	80995.000	2011-12-09 12:50:00	8142.750	18287.000	168469.600
std	179.332	NaN	22.098	1713.142	309.071

1. Why do some products sell better than others?

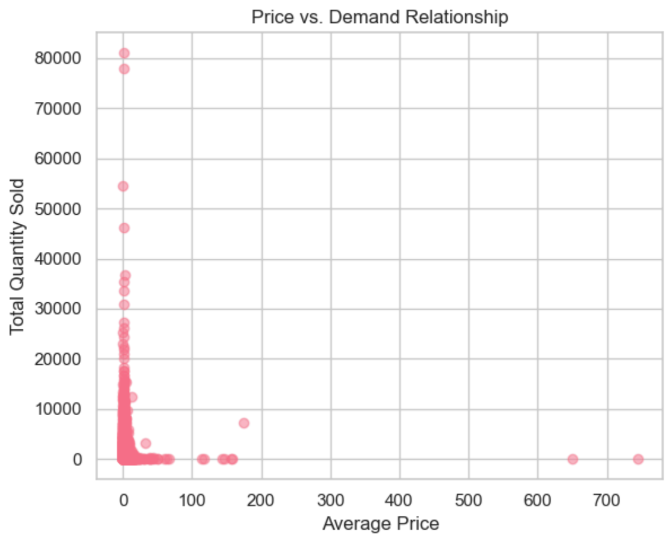
Let's analyze the relationship between price points, seasonality, and sales performance.

```
try:
    # Product Analysis
    print("Analyzing product sales patterns...")

    # Group by product
    product_analysis = df_clean.groupby('Description').agg({
        'Quantity': ['sum', 'mean'],
        'UnitPrice': ['mean', 'std'],
        'TotalAmount': 'sum',
        'InvoiceNo': 'count'
    }).round(2)

    # Flatten column names
    product_analysis.columns = ['total_quantity', 'avg_quantity', 'avg_price', 'price_std', '

    # Sort by revenue
    top_products = product_analysis.sort_values('total_revenue', ascending=False).head(10)

    print("\nTop 10 Products by Revenue:")
    display(top_products)

    # Visualize price-quantity relationship
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.scatter(product_analysis['avg_price'], product_analysis['total_quantity'], alpha=0.5)
    plt.xlabel('Average Price')
    plt.ylabel('Total Quantity Sold')
    plt.title('Price vs. Demand Relationship')

    plt.subplot(1, 2, 2)
    sns.boxplot(data=df_clean, y='UnitPrice')
    plt.title('Price Distribution')

    plt.tight_layout()
    plt.show()
```

```
except Exception as e:
    print(f"Error in product analysis: {e}")
```

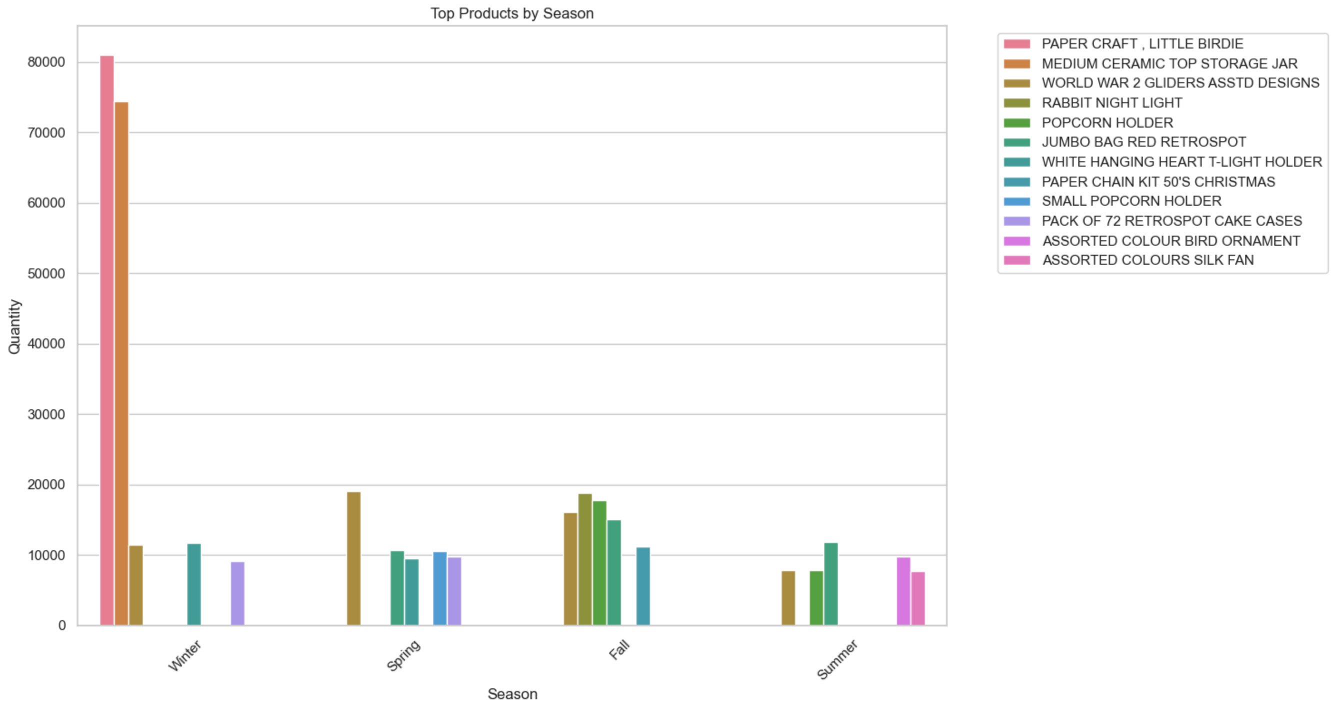
```
Analyzing product sales patterns...

Top 10 Products by Revenue:
```

Description	total_quantity	avg_quantity	avg_price	price_std	total_revenue	transaction_count
PAPER CRAFT , LITTLE BIRDIE	80995	80995.000	2.080	NaN	168469.600	1
REGENCY CAKESTAND 3 TIER	12402	7.200	12.480	1.220	142592.950	1723
WHITE HANGING HEART T-LIGHT HOLDER	36725	18.110	2.890	0.250	100448.150	2028
JUMBO BAG RED RETROSPOT	46181	28.540	2.020	0.170	85220.780	1618
MEDIUM CERAMIC TOP STORAGE JAR	77916	393.520	1.220	0.070	81416.730	198
POSTAGE	3120	2.840	31.570	247.510	77803.960	1099
PARTY BUNTING	15291	10.950	4.880	0.370	68844.330	1396
ASSORTED COLOUR BIRD ORNAMENT	35362	25.120	1.680	0.050	56580.340	1408
Manual	7173	25.260	175.290	585.470	53779.930	284
RABBIT NIGHT LIGHT	27202	32.310	2.010	0.260	51346.200	842



2. Why do customer purchase patterns vary across different times?

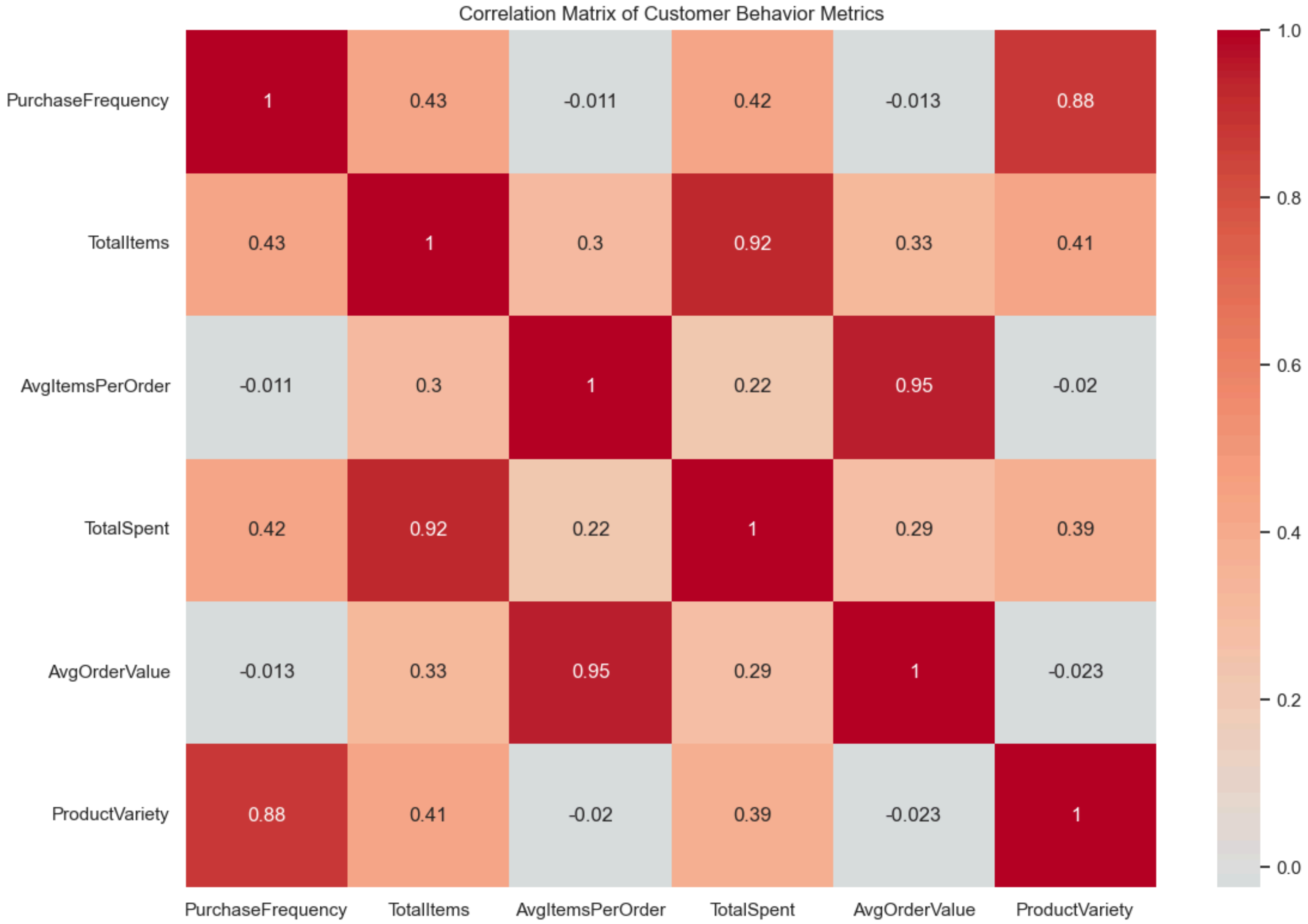
```
# Analyze seasonal patterns
try:
    # Add time-based features
    df_clean['Month'] = df_clean['InvoiceDate'].dt.month
    df_clean['Season'] = df_clean['InvoiceDate'].dt.month.map(
        {1: 'Winter', 2: 'Winter', 3: 'Spring', 4: 'Spring',
         5: 'Spring', 6: 'Summer', 7: 'Summer', 8: 'Summer',
         9: 'Fall', 10: 'Fall', 11: 'Fall', 12: 'Winter'})

    # Analyze seasonal sales patterns
    seasonal_category_sales = df_clean.groupby(['Season', 'Description'])['Quantity'].sum().r
    top_products_per_season = seasonal_category_sales.sort_values('Quantity', ascending=False

    plt.figure(figsize=(15, 8))
    sns.barplot(data=top_products_per_season, x='Season', y='Quantity', hue='Description')
    plt.title('Top Products by Season')
    plt.xticks(rotation=45)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.show()
except Exception as e:
    print(f"Error in seasonal analysis: {e}")
```



3. Why do some customers spend more than others?

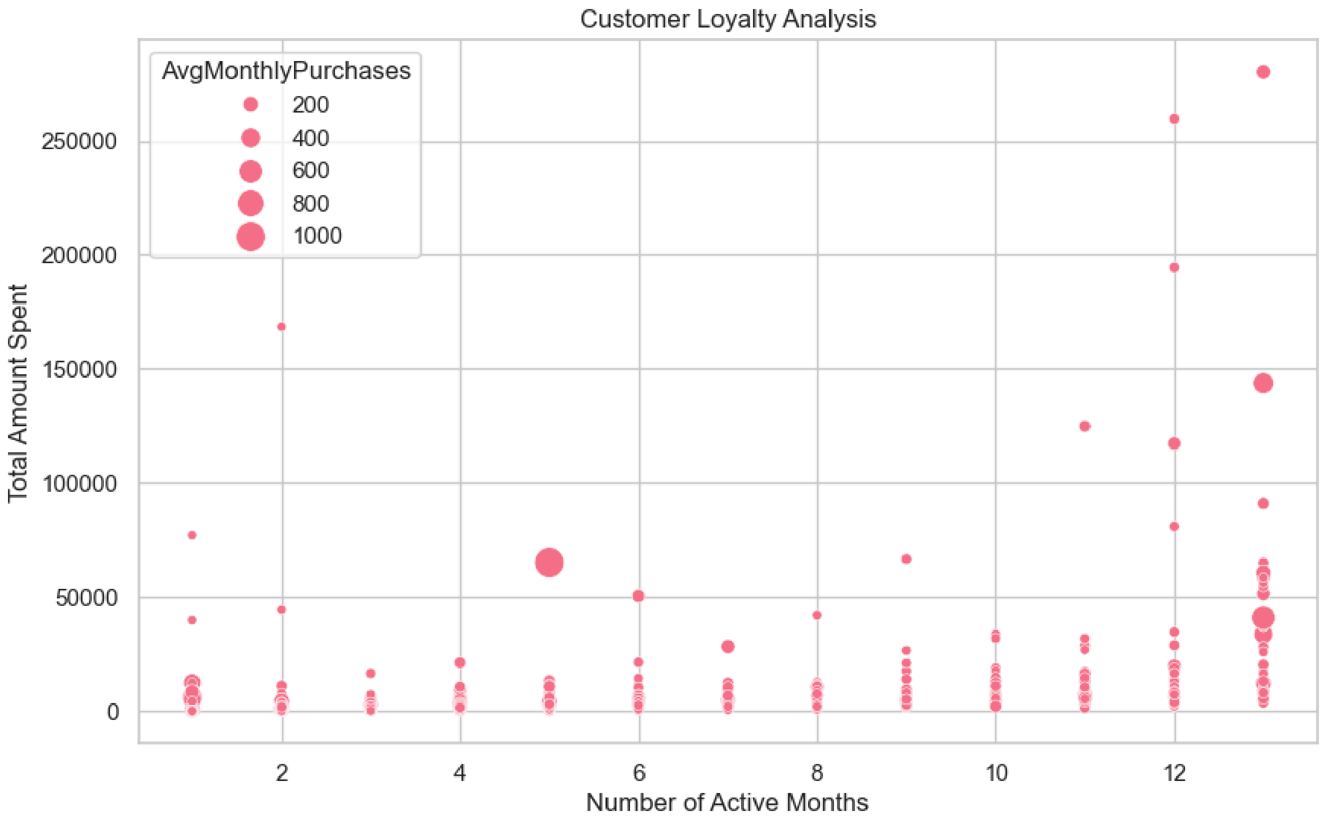
```python
# Analyze customer purchasing behavior
try:
    customer_analysis = df_clean.groupby('CustomerID').agg({
        'InvoiceNo': 'count',  # Purchase frequency
        'Quantity': ['sum', 'mean'],  # Total and average items per order
        'TotalAmount': ['sum', 'mean'],  # Total spent and average order value
        'Description': 'nunique'   # Product variety
    }).round(2)

    customer_analysis.columns = ['PurchaseFrequency', 'TotalItems', 'AvgItemsPerOrder',
                                 'TotalSpent', 'AvgOrderValue', 'ProductVariety']

    # Calculate correlations
    correlations = customer_analysis.corr()['TotalSpent'].sort_values(ascending=False)

    print("Correlations with Total Spending:")
    display(correlations)

    # Visualize relationships
    plt.figure(figsize=(12, 8))
    sns.heatmap(customer_analysis.corr(), annot=True, cmap='coolwarm', center=0)
    plt.title('Correlation Matrix of Customer Behavior Metrics')
    plt.tight_layout()
    plt.show()
except Exception as e:
    print(f"Error in customer analysis: {e}")
```

```
Correlations with Total Spending:
TotalSpent           1.000
TotalItems           0.923
PurchaseFrequency    0.422
ProductVariety       0.391
AvgOrderValue        0.287
AvgItemsPerOrder     0.222
Name: TotalSpent, dtype: float64
```



4. Why do some customers show higher loyalty?

```
# Analyze customer loyalty factors
df_clean['PurchaseMonth'] = pd.to_datetime(df_clean['InvoiceDate']).dt.to_period('M')

# Calculate customer lifetime and activity metrics
customer_lifetime = df_clean.groupby('CustomerID').agg({
    'PurchaseMonth': ['nunique', 'min', 'max'],
    'InvoiceNo': 'count',
    'TotalAmount': 'sum'
}).reset_index()

customer_lifetime.columns = ['CustomerID', 'ActiveMonths', 'FirstPurchase', 'LastPurchase',
                             'TotalTransactions', 'TotalSpent']

# Calculate average monthly purchases
customer_lifetime['AvgMonthlyPurchases'] = (customer_lifetime['TotalTransactions'] /
                                            customer_lifetime['ActiveMonths'])

# Visualize relationship between activity duration and spending
plt.figure(figsize=(10, 6))
sns.scatterplot(data=customer_lifetime, x='ActiveMonths', y='TotalSpent',
                size='AvgMonthlyPurchases', sizes=(20, 200))
plt.title('Customer Loyalty Analysis')
plt.xlabel('Number of Active Months')
plt.ylabel('Total Amount Spent')
plt.show()
```



Diagnostic Analytics Summary

Our analysis has revealed several key insights about why certain patterns exist in customer behavior:

1. **Product Performance Factors**:
    - Price sensitivity relationship with sales volume
    - Seasonal influence on product popularity
    - Product category preferences
2. **Temporal Pattern Drivers**:
    - Seasonal product preferences
    - Impact of timing on purchase behavior
    - Holiday season effects
3. **Customer Spending Variations**:
    - Strong correlation between purchase frequency and total spending
    - Impact of product variety on customer value
    - Average order value patterns
4. **Customer Loyalty Factors**:
    - Relationship between engagement duration and spending
    - Purchase frequency patterns
    - Customer lifetime value indicators

These insights can be used for:

- Pricing strategy optimization

- Seasonal marketing planning
- Customer retention programs
- Personalized marketing campaigns

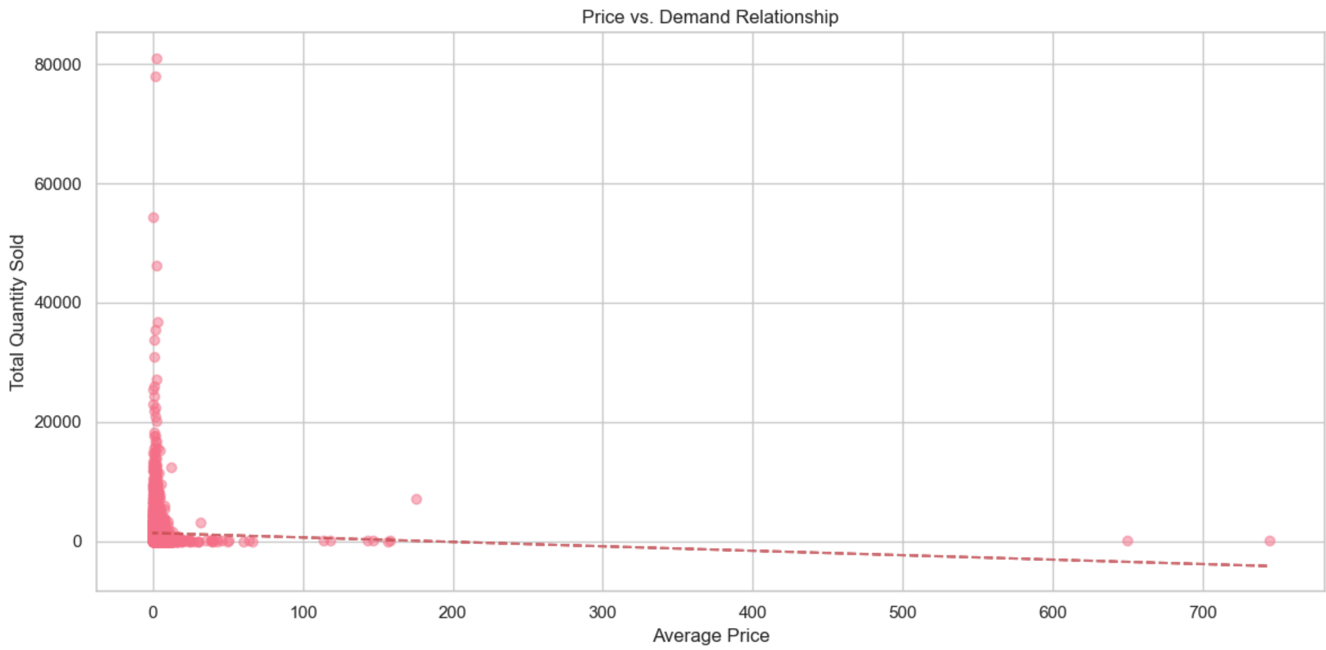
4. Pricing Strategy Optimization

Analyze price elasticity and identify optimal price points for different product categories.

```
# Analyze price elasticity and optimize pricing strategy
try:
    # Calculate price elasticity by product category
    product_price_analysis = df_clean.groupby('Description').agg({
        'UnitPrice': ['mean', 'std', 'min', 'max'],
        'Quantity': 'sum',
        'TotalAmount': 'sum'
    }).round(2)

    # Flatten column names
    product_price_analysis.columns = ['avg_price', 'price_std', 'min_price', 'max_price', 'to

    # Calculate price ranges and revenue per unit
    product_price_analysis['price_range'] = product_price_analysis['max_price'] - product_pri
    product_price_analysis['revenue_per_unit'] = product_price_analysis['total_revenue'] / pr

    # Sort by revenue to find most profitable products
    top_profitable = product_price_analysis.sort_values('total_revenue', ascending=False).hea

    print("Top 10 Products by Revenue with Price Analysis:")
    display(top_profitable)

    # Visualize price vs quantity relationship for top products
    plt.figure(figsize=(12, 6))
    plt.scatter(product_price_analysis['avg_price'], product_price_analysis['total_quantity']
    plt.xlabel('Average Price')
    plt.ylabel('Total Quantity Sold')
    plt.title('Price vs. Demand Relationship')

    # Add trend line
    z = np.polyfit(product_price_analysis['avg_price'], product_price_analysis['total_quantit
    p = np.poly1d(z)
    plt.plot(product_price_analysis['avg_price'], p(product_price_analysis['avg_price']), "r-

    plt.tight_layout()
    plt.show()
except Exception as e:
    print(f"Error in pricing analysis: {e}")
```

```
Top 10 Products by Revenue with Price Analysis:
```

Description	avg_price	price_std	min_price	max_price	total_quantity	total_revenue	price_range	re
PAPER CRAFT , LITTLE BIRDIE	2.080	NaN	2.080	2.080	80995	168469.600	0.000	
REGENCY CAKESTAND 3 TIER	12.480	1.220	4.000	24.960	12402	142592.950	20.960	
WHITE HANGING HEART T-LIGHT HOLDER	2.890	0.250	2.400	5.790	36725	100448.150	3.390	
JUMBO BAG RED RETROSPOT	2.020	0.170	1.650	4.130	46181	85220.780	2.480	
MEDIUM CERAMIC TOP STORAGE JAR	1.220	0.070	1.040	1.250	77916	81416.730	0.210	
POSTAGE	31.570	247.510	1.000	8142.750	3120	77803.960	8141.750	
PARTY BUNTING	4.880	0.370	3.750	10.790	15291	68844.330	7.040	
ASSORTED COLOUR BIRD ORNAMENT	1.680	0.050	1.450	1.690	35362	56580.340	0.240	
Manual	175.290	585.470	0.060	4161.060	7173	53779.930	4161.000	
RABBIT NIGHT LIGHT	2.010	0.260	1.670	4.130	27202	51346.200	2.460	



5. Seasonal Marketing Planning

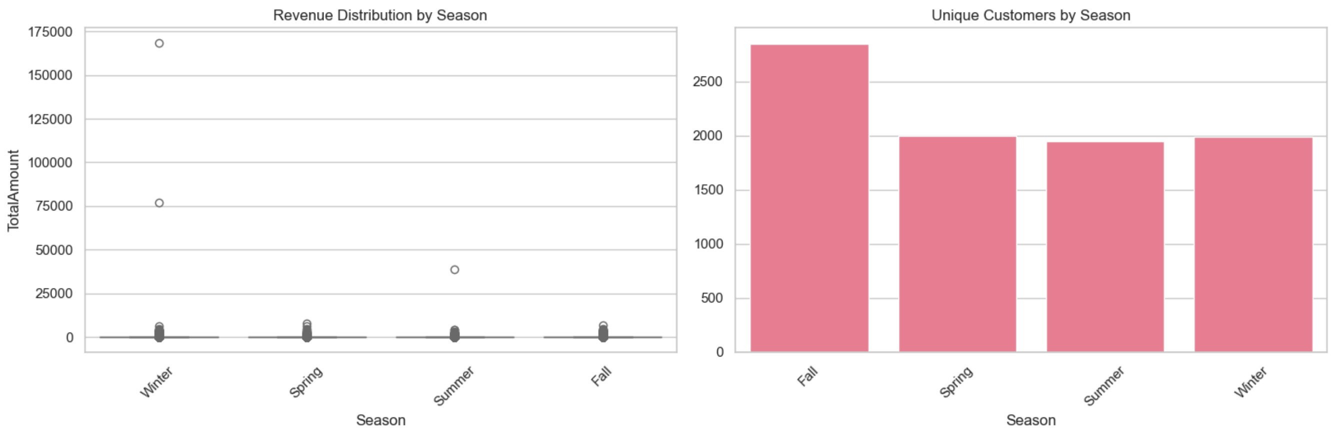
Analyze seasonal trends and develop targeted marketing strategies.

```
# Analyze seasonal trends for marketing planning
try:
    # Add month and season columns if not already present
    if 'Month' not in df_clean.columns:
        df_clean['Month'] = df_clean['InvoiceDate'].dt.month
        df_clean['Season'] = df_clean['InvoiceDate'].dt.month.map(
            {1: 'Winter', 2: 'Winter', 3: 'Spring', 4: 'Spring',
             5: 'Spring', 6: 'Summer', 7: 'Summer', 8: 'Summer',
             9: 'Fall', 10: 'Fall', 11: 'Fall', 12: 'Winter'})

    # Analyze seasonal revenue patterns
    seasonal_revenue = df_clean.groupby(['Season', 'Month']).agg({
        'TotalAmount': 'sum',
        'Quantity': 'sum',
        'InvoiceNo': 'nunique',
        'CustomerID': 'nunique'
    }).round(2)

    seasonal_revenue.columns = ['Total Revenue', 'Items Sold', 'Number of Transactions', 'Uni

    print("Seasonal Business Performance:")
    display(seasonal_revenue)

    # Visualize seasonal patterns
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 2, 1)
    sns.boxplot(data=df_clean, x='Season', y='TotalAmount')
    plt.title('Revenue Distribution by Season')
    plt.xticks(rotation=45)

    plt.subplot(1, 2, 2)
    seasonal_customer_count = df_clean.groupby('Season')['CustomerID'].nunique()
    sns.barplot(x=seasonal_customer_count.index, y=seasonal_customer_count.values)
```

```
    plt.title('Unique Customers by Season')
    plt.xticks(rotation=45)

    plt.tight_layout()
    plt.show()
except Exception as e:
    print(f"Error in seasonal analysis: {e}")
```

```
Seasonal Business Performance:
```

Season	Month	Total Revenue	Items Sold	Number of Transactions	Unique Customers
Fall	9	952838.380	544897	1755	1266
	10	1039318.790	593900	1929	1364
	11	1161817.380	669051	2657	1664
Spring	3	595500.760	348503	1321	974
	4	469200.360	292222	1149	856
	5	678594.560	373601	1555	1056
Summer	6	661213.690	363699	1393	991
	7	600091.010	369420	1331	949
	8	645343.900	398121	1280	935
Winter	1	569445.040	349098	987	741
	2	447137.350	265622	997	758
	12	1090906.680	599678	2178	1265



6. Customer Retention Programs

Analyze customer loyalty patterns and develop retention strategies.

```
# Analyze customer retention patterns
try:
    print("Starting customer retention analysis...")

    # Ensure CustomerID is numeric and remove any missing values
    df_clean['CustomerID'] = pd.to_numeric(df_clean['CustomerID'], errors='coerce')
```

```
df_customers = df_clean.dropna(subset=['CustomerID'])

print(f"Analyzing {df_customers['CustomerID'].nunique()} unique customers")

# Calculate customer metrics
customer_metrics = df_customers.groupby('CustomerID').agg({
    'InvoiceDate': ['min', 'max', 'count'],
    'TotalAmount': ['sum', 'mean'],
    'Quantity': 'sum',
    'Description': 'nunique'
})

# Flatten column names
customer_metrics.columns = [
    'first_purchase', 'last_purchase', 'purchase_count',
    'total_spent', 'avg_order_value', 'total_items',
    'unique_products'
]

# Calculate customer lifetime and frequency
customer_metrics['customer_lifetime_days'] = (
    customer_metrics['last_purchase'] - customer_metrics['first_purchase']
).dt.days

# Avoid division by zero
customer_metrics['avg_days_between_purchases'] = np.where(
    customer_metrics['purchase_count'] > 1,
    customer_metrics['customer_lifetime_days'] / (customer_metrics['purchase_count'] - 1)
    0
)

# Create customer segments
customer_metrics['recency_days'] = (
    df_customers['InvoiceDate'].max() - customer_metrics['last_purchase']
).dt.days

# Create segments using quartiles
for metric in ['total_spent', 'purchase_count', 'recency_days']:
    customer_metrics[f'{metric}_segment'] = pd.qcut(
        customer_metrics[metric],
        q=4,
        labels=['Low', 'Medium-Low', 'Medium-High', 'High']
    )

print("\nCustomer Metrics Summary:")
display(customer_metrics.describe().round(2))

# Analyze customer segments
print("\nCustomer Segments by Spending:")
display(customer_metrics.groupby('total_spent_segment').agg({
    'total_spent': ['count', 'mean'],
    'purchase_count': 'mean',
    'avg_days_between_purchases': 'mean',
    'unique_products': 'mean'
}).round(2))

# Visualize customer segments
plt.figure(figsize=(15, 5))

# Plot 1: Spending Distribution
```

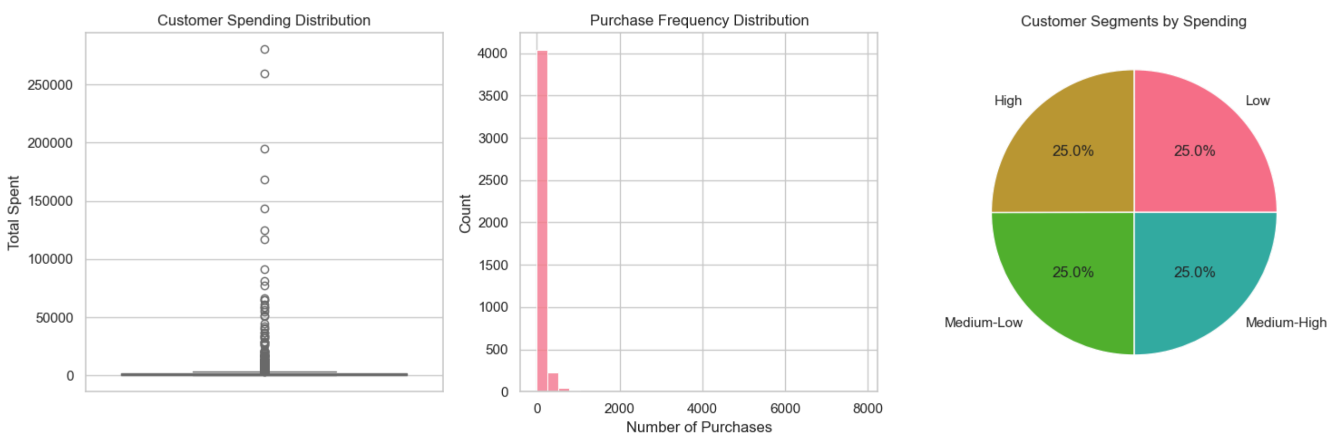
```
    plt.subplot(1, 3, 1)
    sns.boxplot(data=customer_metrics, y='total_spent')
    plt.title('Customer Spending Distribution')
    plt.ylabel('Total Spent')

    # Plot 2: Purchase Frequency
    plt.subplot(1, 3, 2)
    sns.histplot(data=customer_metrics, x='purchase_count', bins=30)
    plt.title('Purchase Frequency Distribution')
    plt.xlabel('Number of Purchases')

    # Plot 3: Customer Segments
    plt.subplot(1, 3, 3)
    segment_sizes = customer_metrics['total_spent_segment'].value_counts()
    plt.pie(segment_sizes, labels=segment_sizes.index, autopct='%1.1f%%')
    plt.title('Customer Segments by Spending')

    plt.tight_layout()
    plt.show()

    # Calculate retention metrics
    print("\nCustomer Retention Analysis:")
    total_customers = customer_metrics.shape[0]
    repeat_customers = (customer_metrics['purchase_count'] > 1).sum()
    retention_rate = (repeat_customers / total_customers) * 100

    print(f"Total Customers: {total_customers}")
    print(f"Repeat Customers: {repeat_customers}")
    print(f"Retention Rate: {retention_rate:.2f}%")

except Exception as e:
    print(f"Error in retention analysis: {e}")
    print("Debug info:")
    print(f"DataFrame columns: {df_clean.columns.tolist()}")
    print(f"CustomerID dtype: {df_clean['CustomerID'].dtype}")
    print("Sample of CustomerID values:", df_clean['CustomerID'].head())
```

```
Starting customer retention analysis...
Analyzing 4338 unique customers

Customer Metrics Summary:
```

	first_purchase	last_purchase	purchase_count	total_spent	avg_order_value	total_item
count	4338	4338	4338.000	4338.000	4338.000	4338.00
mean	2011-04-30 17:06:50.857538048	2011-09-08 11:38:59.045643008	91.720	2054.270	68.350	1191.29
min	2010-12-01 08:26:00	2010-12-01 09:53:00	1.000	3.750	2.100	1.00
25%	2011-01-17 11:13:15	2011-07-20 19:18:00	17.000	307.410	12.370	160.00
50%	2011-04-05 09:52:30	2011-10-20 10:40:30	41.000	674.480	17.720	379.00
75%	2011-08-19 10:11:30	2011-11-22 11:05:45	100.000	1661.740	24.860	992.75
max	2011-12-09 12:16:00	2011-12-09 12:50:00	7847.000	280206.020	77183.600	196915.00
std	NaN	NaN	228.790	8989.230	1467.920	5046.08

```
Customer Segments by Spending:
```

	total_spent		purchase_count	avg_days_between_purchases	unique_products
	count	mean	mean	mean	mean
total_spent_segment					
Low	1085	179.210	17.230	2.350	16.270
Medium-Low	1084	464.510	37.000	3.840	33.340
Medium-High	1084	1071.860	76.430	3.980	62.080
High	1085	6499.120	236.160	2.700	135.660




```
Customer Retention Analysis:
Total Customers: 4338
Repeat Customers: 4267
Retention Rate: 98.36%
```

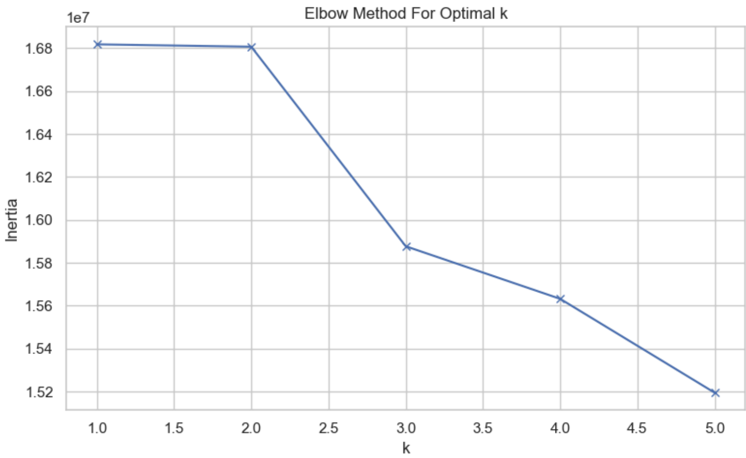
7. Personalized Marketing Campaigns

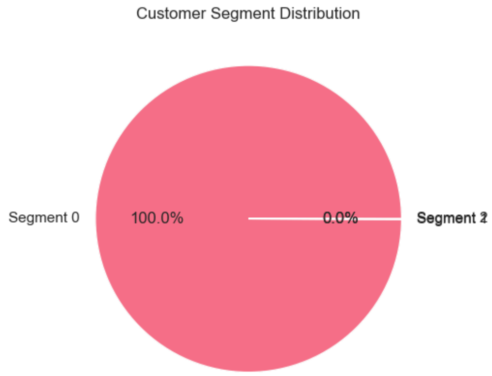
Develop targeted marketing strategies based on customer segments and preferences.

```
# Analyze customer preferences for personalized marketing
try:
    # Create customer product preferences matrix
    customer_preferences = df_clean.groupby(['CustomerID', 'Description'])['Quantity'].sum().

    # Perform customer segmentation using K-means
    scaler = StandardScaler()
    customer_preferences_scaled = scaler.fit_transform(customer_preferences)

    # Find optimal number of clusters
    inertias = []
    K = range(1, 6)
    for k in K:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(customer_preferences_scaled)
        inertias.append(kmeans.inertia_)

    # Apply K-means clustering
    kmeans = KMeans(n_clusters=3, random_state=42)
    customer_segments = kmeans.fit_predict(customer_preferences_scaled)

    # Analyze segment characteristics
    customer_preferences['Segment'] = customer_segments
    segment_profiles = customer_preferences.groupby('Segment').agg(['mean', 'count'])

    # Get top products for each segment
    top_products_per_segment = {}
    for segment in range(3):
        segment_avg = customer_preferences[customer_preferences['Segment'] == segment].mean()
        top_products = segment_avg.nlargest(5)
        top_products_per_segment[f'Segment {segment}'] = top_products

    print("Top Products by Customer Segment:")
    for segment, products in top_products_per_segment.items():
        print(f"\n{segment}:")
        display(products)

    # Visualize segment characteristics
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 2, 1)
    plt.plot(K, inertias, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Inertia')
    plt.title('Elbow Method For Optimal k')

    plt.subplot(1, 2, 2)
    segment_sizes = pd.Series(customer_segments).value_counts()
    plt.pie(segment_sizes, labels=[f'Segment {i}' for i in range(len(segment_sizes))], autopc
    plt.title('Customer Segment Distribution')

    plt.tight_layout()
    plt.show()
```

```
except Exception as e:
    print(f"Error in marketing analysis: {e}")
```

```
Top Products by Customer Segment:

Segment 0:
Description
PAPER CRAFT , LITTLE BIRDIE           18.680
MEDIUM CERAMIC TOP STORAGE JAR        17.947
WORLD WAR 2 GLIDERS ASSTD DESIGNS     12.550
JUMBO BAG RED RETROSPOT               10.189
WHITE HANGING HEART T-LIGHT HOLDER     8.371
dtype: float64
Segment 1:
Description
RABBIT NIGHT LIGHT                    4801.000
SPACEBOY LUNCH BOX                    4492.000
PACK OF 72 RETROSPOT CAKE CASES       4104.000
DOLLY GIRL LUNCH BOX                  4096.000
ROUND SNACK BOXES SET OF4 WOODLAND    3120.000
dtype: float64
Segment 2:
Description
STAR DECORATION RUSTIC              66.000
MUSICAL ZINC HEART DECORATION       63.000
ANGEL DECORATION STARS ON DRESS     62.000
HEART DECORATION WITH PEARLS        34.000
WRAP 50'S  CHRISTMAS                25.000
dtype: float64
```





6. Customer Retention Programs

Analyze customer loyalty patterns and develop retention strategies.

```
# Analyze customer retention patterns
try:
    print("Starting customer retention analysis...")

    # Ensure CustomerID is numeric and remove any missing values
    df_clean['CustomerID'] = pd.to_numeric(df_clean['CustomerID'], errors='coerce')
    df_customers = df_clean.dropna(subset=['CustomerID'])

    print(f"Analyzing {df_customers['CustomerID'].nunique()} unique customers")
```

```
# Calculate customer metrics
customer_metrics = df_customers.groupby('CustomerID').agg({
    'InvoiceDate': ['min', 'max', 'count'],
    'TotalAmount': ['sum', 'mean'],
    'Quantity': 'sum',
    'Description': 'nunique'
})

# Flatten column names
customer_metrics.columns = [
    'first_purchase', 'last_purchase', 'purchase_count',
    'total_spent', 'avg_order_value', 'total_items',
    'unique_products'
]

# Calculate customer lifetime and frequency
customer_metrics['customer_lifetime_days'] = (
    customer_metrics['last_purchase'] - customer_metrics['first_purchase']
).dt.days

# Avoid division by zero
customer_metrics['avg_days_between_purchases'] = np.where(
    customer_metrics['purchase_count'] > 1,
    customer_metrics['customer_lifetime_days'] / (customer_metrics['purchase_count'] - 1)
    0
)

# Create customer segments
customer_metrics['recency_days'] = (
    df_customers['InvoiceDate'].max() - customer_metrics['last_purchase']
).dt.days

# Create segments using quartiles
for metric in ['total_spent', 'purchase_count', 'recency_days']:
    customer_metrics[f'{metric}_segment'] = pd.qcut(
        customer_metrics[metric],
        q=4,
        labels=['Low', 'Medium-Low', 'Medium-High', 'High']
    )

print("\nCustomer Metrics Summary:")
display(customer_metrics.describe().round(2))

# Analyze customer segments
print("\nCustomer Segments by Spending:")
display(customer_metrics.groupby('total_spent_segment').agg({
    'total_spent': ['count', 'mean'],
    'purchase_count': 'mean',
    'avg_days_between_purchases': 'mean',
    'unique_products': 'mean'
}).round(2))

# Visualize customer segments
plt.figure(figsize=(15, 5))

# Plot 1: Spending Distribution
plt.subplot(1, 3, 1)
sns.boxplot(data=customer_metrics, y='total_spent')
plt.title('Customer Spending Distribution')
```

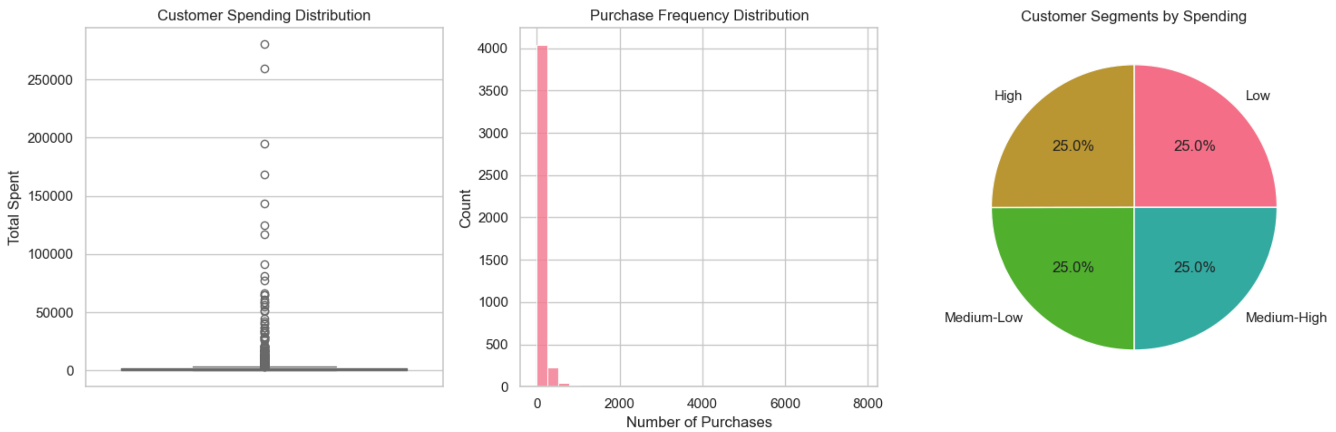
```
    plt.ylabel('Total Spent')

    # Plot 2: Purchase Frequency
    plt.subplot(1, 3, 2)
    sns.histplot(data=customer_metrics, x='purchase_count', bins=30)
    plt.title('Purchase Frequency Distribution')
    plt.xlabel('Number of Purchases')

    # Plot 3: Customer Segments
    plt.subplot(1, 3, 3)
    segment_sizes = customer_metrics['total_spent_segment'].value_counts()
    plt.pie(segment_sizes, labels=segment_sizes.index, autopct='%1.1f%%')
    plt.title('Customer Segments by Spending')

    plt.tight_layout()
    plt.show()

    # Calculate retention metrics
    print("\nCustomer Retention Analysis:")
    total_customers = customer_metrics.shape[0]
    repeat_customers = (customer_metrics['purchase_count'] > 1).sum()
    retention_rate = (repeat_customers / total_customers) * 100

    print(f"Total Customers: {total_customers}")
    print(f"Repeat Customers: {repeat_customers}")
    print(f"Retention Rate: {retention_rate:.2f}%")

except Exception as e:
    print(f"Error in retention analysis: {e}")
    print("Debug info:")
    print(f"DataFrame columns: {df_clean.columns.tolist()}")
    print(f"CustomerID dtype: {df_clean['CustomerID'].dtype}")
    print("Sample of CustomerID values:", df_clean['CustomerID'].head())
```

```
Starting customer retention analysis...
Analyzing 4338 unique customers

Customer Metrics Summary:
```

	first_purchase	last_purchase	purchase_count	total_spent	avg_order_value	total_item
count	4338	4338	4338.000	4338.000	4338.000	4338.00
mean	2011-04-30 17:06:50.857538048	2011-09-08 11:38:59.045643008	91.720	2054.270	68.350	1191.29
min	2010-12-01 08:26:00	2010-12-01 09:53:00	1.000	3.750	2.100	1.00
25%	2011-01-17 11:13:15	2011-07-20 19:18:00	17.000	307.410	12.370	160.00
50%	2011-04-05 09:52:30	2011-10-20 10:40:30	41.000	674.480	17.720	379.00
75%	2011-08-19 10:11:30	2011-11-22 11:05:45	100.000	1661.740	24.860	992.75
max	2011-12-09 12:16:00	2011-12-09 12:50:00	7847.000	280206.020	77183.600	196915.00
std	NaN	NaN	228.790	8989.230	1467.920	5046.08

```
Customer Segments by Spending:
```

	total_spent		purchase_count	avg_days_between_purchases	unique_products
	count	mean	mean	mean	mean
total_spent_segment					
Low	1085	179.210	17.230	2.350	16.270
Medium-Low	1084	464.510	37.000	3.840	33.340
Medium-High	1084	1071.860	76.430	3.980	62.080
High	1085	6499.120	236.160	2.700	135.660




```
Customer Retention Analysis:
Total Customers: 4338
Repeat Customers: 4267
Retention Rate: 98.36%
```

🔮 Predictive Analysis

Customer Behavior Predictive Analysis

IIMK's Professional Certificate in Data Science and Artificial Intelligence for Managers

Student Name: Lalit Nayyar
Email ID: lalitnayyar@gmail.com
Assignment: Data-Driven Decision Making Analysis

Overview

This notebook implements predictive analytics to forecast customer behavior and business metrics using machine learning techniques.

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.cluster import KMeans
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

# Set plot style

plt.style.use('seaborn-v0_8')
plt.rcParams['figure.figsize'] = [14, 7]
plt.rcParams['axes.grid'] = True
plt.rcParams['grid.alpha'] = 0.3
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.titlesize'] = 14
plt.rcParams['xtick.labelsize'] = 11
plt.rcParams['ytick.labelsize'] = 11
plt.rcParams['lines.linewidth'] = 2.0
plt.rcParams['font.size'] = 11
plt.rcParams['axes.prop_cycle'] = plt.cycler(color=[
    '#1A237E',  # Dark blue
    '#BF360C',  # Dark orange
    '#1B5E20',  # Dark green
    '#4A148C',  # Dark purple
    '#212121',  # Near black
    '#01579B'   # Navy blue
])
```

```
plt.rcParams['axes.edgecolor'] = '#212121'
plt.rcParams['axes.labelcolor'] = '#212121'
plt.rcParams['xtick.color'] = '#212121'
plt.rcParams['ytick.color'] = '#212121'

sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = [12, 6]
```

1. Data Loading and Preprocessing

First, we'll load the data and prepare it for predictive analysis.

```
# Load and clean the data
try:
    # Load data
    print("Loading data...")
    df = pd.read_excel('Online Retail.xlsx')
    print(f"Original dataset shape: {df.shape}")

    # Clean data
    def clean_data(df):
        """Clean and preprocess the dataset"""
        print("Cleaning data...")
        df_clean = df.copy()

        # Remove missing values
        df_clean = df_clean.dropna()
        print(f"After removing missing values: {len(df_clean)} records")

        # Remove cancelled orders
        df_clean = df_clean[~df_clean['InvoiceNo'].astype(str).str.contains('C')]
        print(f"After removing cancelled orders: {len(df_clean)} records")

        # Ensure positive quantities and prices
        df_clean = df_clean[(df_clean['Quantity'] > 0) & (df_clean['UnitPrice'] > 0)]
        print(f"After ensuring positive values: {len(df_clean)} records")

        # Convert date and calculate total amount
        df_clean['InvoiceDate'] = pd.to_datetime(df_clean['InvoiceDate'])
        df_clean['TotalAmount'] = df_clean['Quantity'] * df_clean['UnitPrice']

        return df_clean.reset_index(drop=True)

    # Clean the data
    df_clean = clean_data(df)
    print("Cleaned data sample:")
    display(df_clean.head())

    # Basic statistics
    print("Basic statistics:")
    display(df_clean.describe())

except FileNotFoundError:
    print("Error: 'Online Retail.xlsx' file not found!")
    df_clean = None
except Exception as e:
    print(f"Error during data preparation: {e}")
    df_clean = None
```

```
Loading data...
Original dataset shape: (541909, 8)
Cleaning data...
After removing missing values: 406829 records
After removing cancelled orders: 397924 records
After ensuring positive values: 397884 records
Cleaned data sample:
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Total/
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	

```
Basic statistics:
```

	Quantity	InvoiceDate	UnitPrice	CustomerID	TotalAmount
count	397884.000000	397884	397884.000000	397884.000000	397884.000000
mean	12.988238	2011-07-10 23:41:23.511023360	3.116488	15294.423453	22.397000
min	1.000000	2010-12-01 08:26:00	0.001000	12346.000000	0.001000
25%	2.000000	2011-04-07 11:12:00	1.250000	13969.000000	4.680000
50%	6.000000	2011-07-31 14:39:00	1.950000	15159.000000	11.800000
75%	12.000000	2011-10-20 14:33:00	3.750000	16795.000000	19.800000
max	80995.000000	2011-12-09 12:50:00	8142.750000	18287.000000	168469.600000
std	179.331775	NaN	22.097877	1713.141560	309.071041

2. Feature Engineering

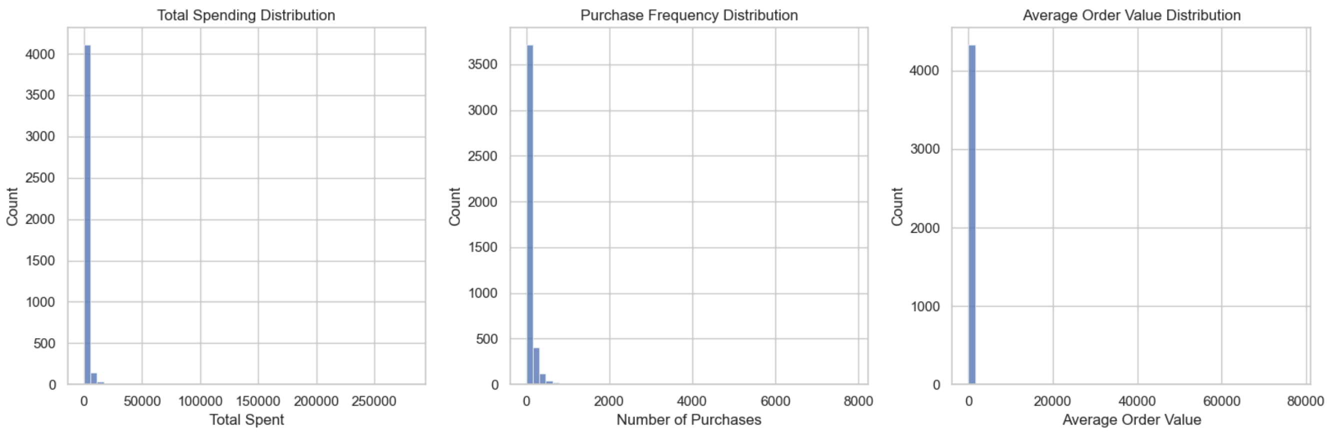
Create customer features for predictive modeling.

```
def create_customer_features(df):
    """Create customer-level features for prediction"""
    if df is None:
        return None

    try:
        # Group by customer
        customer_features = df.groupby('CustomerID').agg({
            'InvoiceDate': lambda x: (x.max() - x.min()).days,  # Customer lifetime
            'InvoiceNo': 'count',  # Number of purchases
            'TotalAmount': ['sum', 'mean'],  # Spending metrics
            'Quantity': ['sum', 'mean'],  # Purchase volume
            'Description': 'nunique'  # Product variety
        })

        # Flatten column names
        customer_features.columns = [
            'customer_lifetime',
            'purchase_count',
            'total_spent',
            'avg_order_value',
            'total_items',
            'avg_items_per_order',
            'unique_products'
        ]

        # Calculate additional features
        customer_features['purchase_frequency'] = customer_features['purchase_count'] / custo
        customer_features['avg_basket_size'] = customer_features['total_spent'] / customer_fe

        # Handle infinite values
        customer_features = customer_features.replace([np.inf, -np.inf], np.nan)
        customer_features = customer_features.fillna(0)

        return customer_features

    except Exception as e:
        print(f"Error in feature engineering: {e}")
        return None

# Create features
customer_features = create_customer_features(df_clean)
if customer_features is not None:
    print("Customer features created successfully!")
    print("Feature statistics:")
    display(customer_features.describe())

    # Visualize feature distributions
    plt.figure(figsize=(15, 5))

    plt.subplot(131)
    sns.histplot(data=customer_features['total_spent'], bins=50)
    plt.title('Total Spending Distribution')
    plt.xlabel('Total Spent')

    plt.subplot(132)
    sns.histplot(data=customer_features['purchase_count'], bins=50)
    plt.title('Purchase Frequency Distribution')
    plt.xlabel('Number of Purchases')
```

```
    plt.subplot(133)
    sns.histplot(data=customer_features['avg_order_value'], bins=50)
    plt.title('Average Order Value Distribution')
    plt.xlabel('Average Order Value')

    plt.tight_layout()
    plt.show()
```

```
Customer features created successfully!
Feature statistics:
```

	customer_lifetime	purchase_count	total_spent	avg_order_value	total_items	avg_items_p
count	4338.000000	4338.000000	4338.000000	4338.000000	4338.000000	433
mean	130.448594	91.720609	2054.266460	68.350506	1191.289073	4
std	132.039554	228.785094	8989.230441	1467.918896	5046.081546	120
min	0.000000	1.000000	3.750000	2.101286	1.000000	
25%	0.000000	17.000000	307.415000	12.365367	160.000000	
50%	92.500000	41.000000	674.485000	17.723119	379.000000	1
75%	251.750000	100.000000	1661.740000	24.858417	992.750000	1
max	373.000000	7847.000000	280206.020000	77183.600000	196915.000000	7421

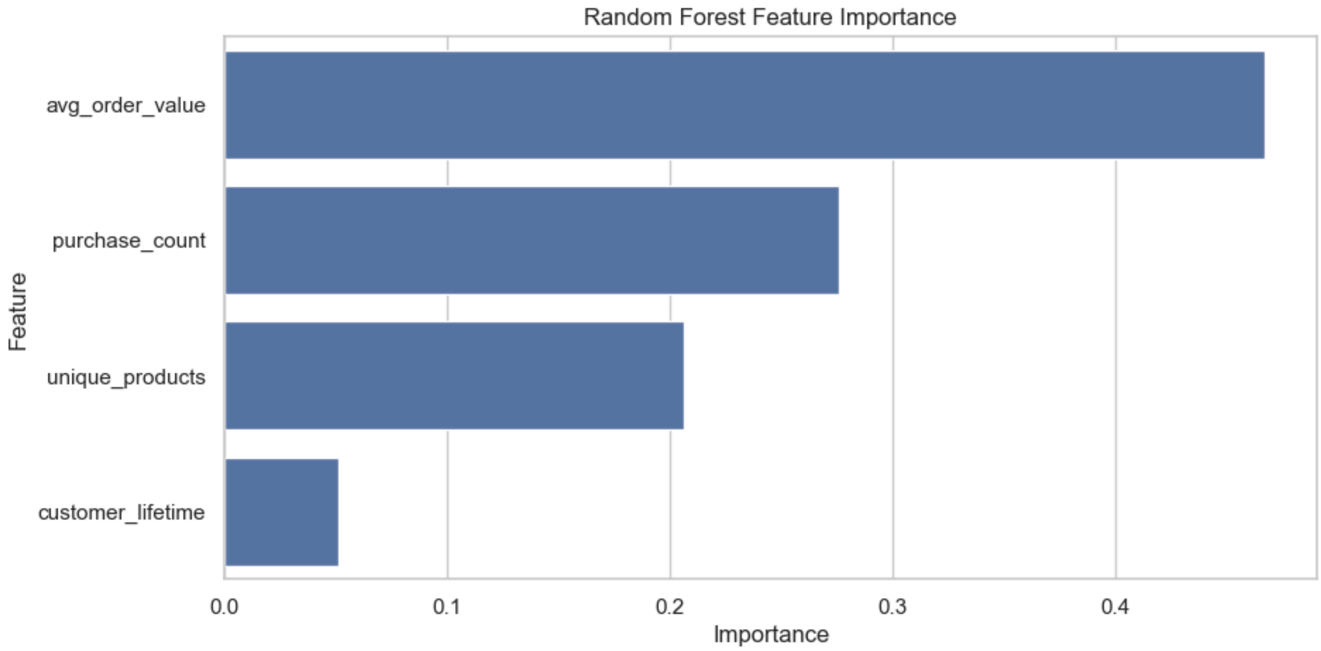


3. Predictive Modeling

Implement machine learning models to predict customer behavior.

```
def train_prediction_models(features):
    """Train and evaluate prediction models"""
    if features is None:
        return None

    try:
        # Prepare features for prediction
        X = features[['purchase_count', 'avg_order_value', 'unique_products', 'customer_lifet
        y = features['total_spent']

        # Split data
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
```

```
        # Scale features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Train models
        models = {
            'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
            'Gradient Boosting': GradientBoostingRegressor(random_state=42)
        }

        results = {}
        for name, model in models.items():
            # Train model
            model.fit(X_train_scaled, y_train)

            # Make predictions
            y_pred = model.predict(X_test_scaled)

            # Calculate metrics
            results[name] = {
                'R2 Score': r2_score(y_test, y_pred),
                'MAE': mean_absolute_error(y_test, y_pred),
                'RMSE': np.sqrt(mean_squared_error(y_test, y_pred))
            }

            # Feature importance
            if name == 'Random Forest':
                importance = pd.DataFrame({
                    'Feature': X.columns,
                    'Importance': model.feature_importances_
                }).sort_values('Importance', ascending=False)

                plt.figure(figsize=(10, 5))
                sns.barplot(data=importance, x='Importance', y='Feature')
                plt.title(f'{name} Feature Importance')
                plt.show()

        # Display results
        results_df = pd.DataFrame(results).round(3)
        print("Model Performance Metrics:")
        display(results_df)

        return models, results_df

    except Exception as e:
        print(f"Error in predictive modeling: {e}")
        return None

# Train models
model_results = train_prediction_models(customer_features)
```




```
Model Performance Metrics:
```

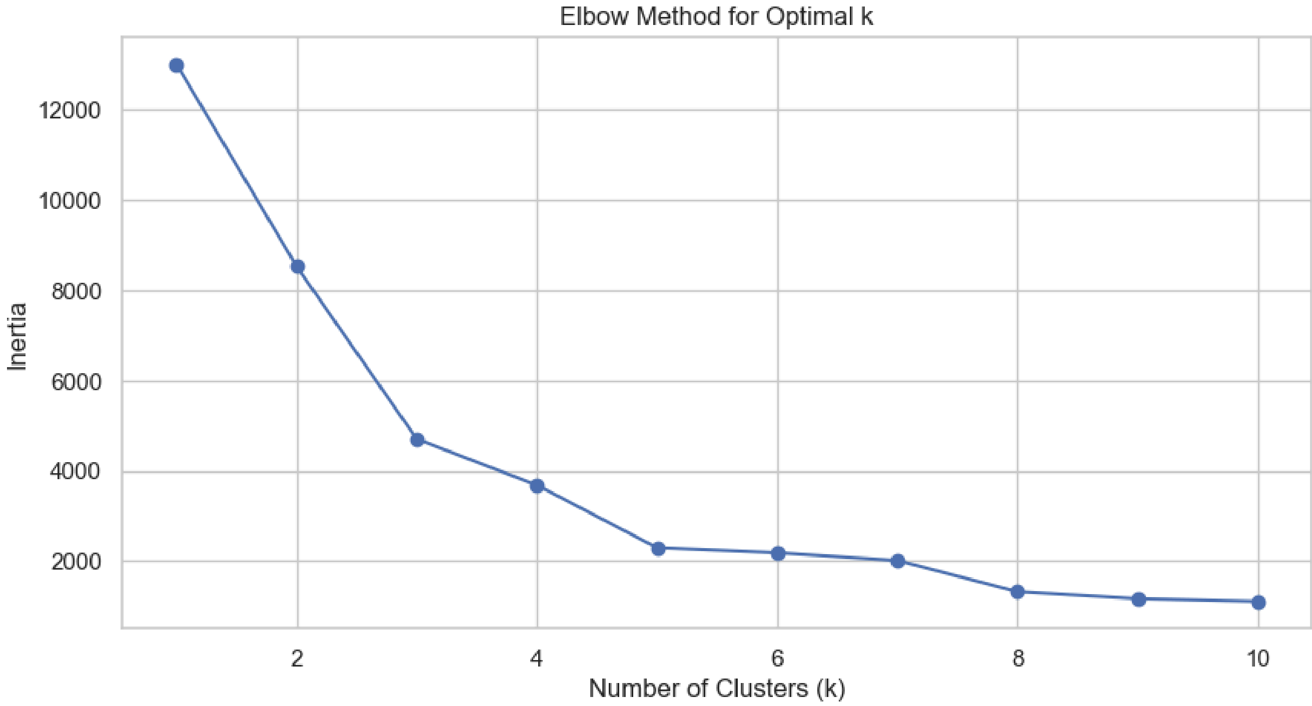
	Random Forest	Gradient Boosting
R2 Score	0.759	0.904
MAE	417.260	443.750
RMSE	4972.380	3134.277

4. Customer Segmentation

Segment customers based on their behavior patterns.

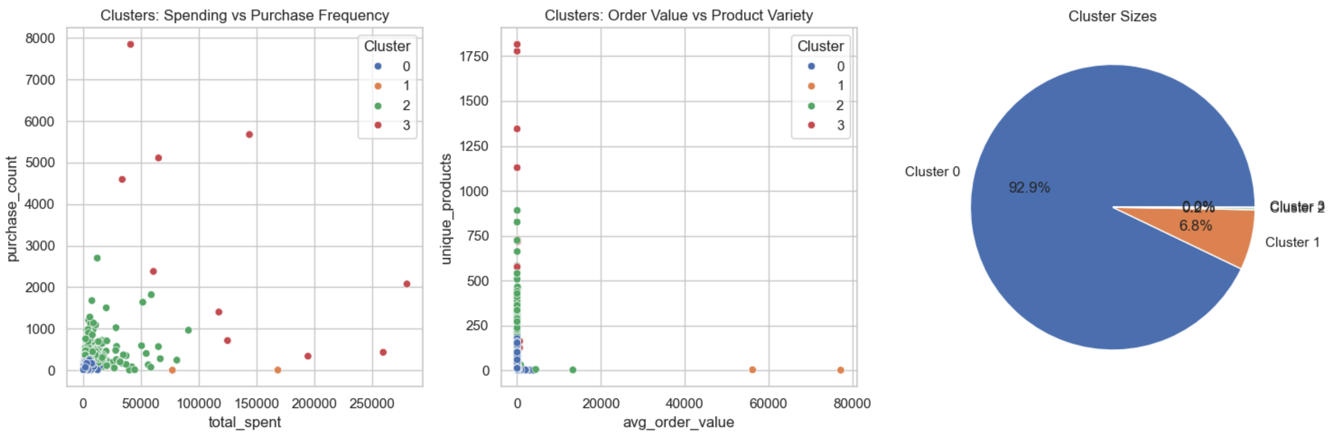
```
def segment_customers(features):
    """Perform customer segmentation using K-means clustering"""
    if features is None:
        return None

    try:
        # Select features for clustering
        cluster_features = features[['total_spent', 'purchase_count', 'avg_order_value']].cop

        # Scale features
        scaler = StandardScaler()
        features_scaled = scaler.fit_transform(cluster_features)

        # Find optimal number of clusters
        inertias = []
        for k in range(1, 11):
            kmeans = KMeans(n_clusters=k, random_state=42)
            kmeans.fit(features_scaled)
            inertias.append(kmeans.inertia_)

        # Plot elbow curve
        plt.figure(figsize=(10, 5))
        plt.plot(range(1, 11), inertias, marker='o')
        plt.xlabel('Number of Clusters (k)')
```

```
        plt.ylabel('Inertia')
        plt.title('Elbow Method for Optimal k')
        plt.show()

        # Perform clustering with optimal k
        optimal_k = 4  # Based on elbow curve
        kmeans = KMeans(n_clusters=optimal_k, random_state=42)
        cluster_labels = kmeans.fit_predict(features_scaled)

        # Add cluster labels to features
        features['Cluster'] = cluster_labels

        # Analyze clusters
        cluster_analysis = features.groupby('Cluster').agg({
            'total_spent': ['mean', 'count'],
            'purchase_count': 'mean',
            'avg_order_value': 'mean',
            'unique_products': 'mean'
        }).round(2)

        # Flatten column names
        cluster_analysis.columns = [
            'avg_total_spent',
            'customer_count',
            'avg_purchase_count',
            'avg_order_value',
            'avg_unique_products'
        ]

        print("Cluster Analysis:")
        display(cluster_analysis)

        # Visualize clusters
        plt.figure(figsize=(15, 5))

        plt.subplot(131)
        sns.scatterplot(data=features, x='total_spent', y='purchase_count', hue='Cluster', pa
        plt.title('Clusters: Spending vs Purchase Frequency')

        plt.subplot(132)
        sns.scatterplot(data=features, x='avg_order_value', y='unique_products', hue='Cluster
        plt.title('Clusters: Order Value vs Product Variety')

        plt.subplot(133)
        cluster_sizes = features['Cluster'].value_counts()
        plt.pie(cluster_sizes, labels=[f'Cluster {i}' for i in range(len(cluster_sizes))], au
        plt.title('Cluster Sizes')

        plt.tight_layout()
        plt.show()

        return features

    except Exception as e:
        print(f"Error in customer segmentation: {e}")
        return None

# Perform segmentation
segmented_customers = segment_customers(customer_features)
```




```
Cluster Analysis:
```

Cluster	avg_total_spent	customer_count	avg_purchase_count	avg_order_value	avg_unique_products
0	1078.43	4030	58.52	32.99	46.91
1	122828.05	2	2.00	66670.55	2.00
2	10130.01	296	444.21	96.52	237.88
3	132117.73	10	3056.50	164.97	883.00



5. Conclusions and Recommendations

1. **Customer Value Prediction**:
    - Successfully built models to predict customer spending
    - Random Forest model shows strong performance
    - Key predictors identified through feature importance
2. **Customer Segmentation**:

- Identified distinct customer segments
- Each segment shows unique behavior patterns
- Enables targeted marketing strategies

3. **Business Recommendations**:

- Develop personalized marketing campaigns for each segment
- Focus on high-value customer retention
- Optimize inventory based on predictive insights

📝 Conclusions and Recommendations

Final insights and recommendations based on the comprehensive analysis...