```
import pandas as pd
from sqlalchemy import create_engine

# Your NeonDB connection string
conn_string = "postgresql://neondb_owner:npg_1qIxnMUSN0ZY@ep-round-voice-ad6vh9yj-pooler.c-2.us-east-1.aws.neon.tech/neondb?sslmode=require&channel_binding=require"

# Create SQLAlchemy engine
engine = create_engine(conn_string)

# Test connection by running a simple query
test_query = "SELECT NOW();"
print("Current DB Time:", pd.read_sql(test_query, engine))
```

```
Current DB Time:                                  now
0 2025-09-02 11:39:56.032191+00:00
```

```
# Path to your CSV in Colab
file_path = "/content/PM/data.csv"

# Load CSV
df = pd.read_csv(file_path, encoding='ISO-8859-1')

# Clean column names
df.columns = [c.strip().replace(" ", "_") for c in df.columns]

# Upload DataFrame to NeonDB
df.to_sql('sales', engine, if_exists='replace', index=False)
print("Data uploaded to NeonDB table 'sales'.")
```

```
Data uploaded to NeonDB table 'sales'.
```

Double-click (or enter) to edit

```
# Step 5: SQL Queries

# 5.1 Total Revenue
query1 = """
SELECT ROUND(SUM("Quantity" * "UnitPrice")::numeric, 2) AS Total_Revenue
FROM sales;
"""
print("\nTotal Revenue:")
print(pd.read_sql(query1, engine))
```

```
Total Revenue:
   total_revenue
0     9747747.93
```

```
# 5.2 Top 10 Best-Selling Products
query2 = """
SELECT "Description", SUM("Quantity") AS Total_Sold
FROM sales
GROUP BY "Description"
ORDER BY Total_Sold DESC
LIMIT 10;
"""
print("\nTop 10 Best-Selling Products:")
print(pd.read_sql(query2, engine))
```

```
Top 10 Best-Selling Products:
                          Description  total_sold
0   WORLD WAR 2 GLIDERS ASSTD DESIGNS     53847.0
1             JUMBO BAG RED RETROSPOT     47363.0
2       ASSORTED COLOUR BIRD ORNAMENT     36381.0
3                     POPCORN HOLDER     36334.0
4     PACK OF 72 RETROSPOT CAKE CASES     36039.0
5  WHITE HANGING HEART T-LIGHT HOLDER     35317.0
6                  RABBIT NIGHT LIGHT     30680.0
7             MINI PAINT SET VINTAGE      26437.0
8          PACK OF 12 LONDON TISSUES      26315.0
9  PACK OF 60 PINK PAISLEY CAKE CASES     24753.0
```

```
# 5.3 Top 10 Customers by Spend
query3 = """
SELECT "CustomerID", ROUND(SUM(("Quantity" * "UnitPrice")::numeric),2) AS Revenue
FROM sales
WHERE "CustomerID" IS NOT NULL
GROUP BY "CustomerID"
ORDER BY Revenue DESC
LIMIT 10;
"""
print("\nTop 10 Customers by Spend:")
print(pd.read_sql(query3, engine))
```

```
Top 10 Customers by Spend:
   CustomerID    revenue
0     14646.0  279489.02
1     18102.0  256438.49
2     17450.0  187482.17
3     14911.0  132572.62
4     12415.0  123725.45
5     14156.0  113384.14
6     17511.0   88125.38
7     16684.0   65892.08
8     13694.0   62653.10
9     15311.0   59419.34
```

```
# 5.4 Monthly Revenue Trend
query4 = """
SELECT TO_CHAR(TO_TIMESTAMP("InvoiceDate", 'MM/DD/YYYY HH24:MI'), 'YYYY-MM') AS Month,
       ROUND(SUM(("Quantity" * "UnitPrice")::numeric), 2) AS Revenue
FROM sales
GROUP BY Month
ORDER BY Month;
"""
print("\nMonthly Revenue Trend:")
```

```
print(pd.read_sql(query4, engine))
```

```
Monthly Revenue Trend:
      month     revenue
0   2010-12   748957.02
1   2011-01   560000.26
2   2011-02   498062.65
3   2011-03   683267.08
4   2011-04   493207.12
5   2011-05   723333.51
6   2011-06   691123.12
7   2011-07   681300.11
8   2011-08   682680.51
9   2011-09  1019687.62
10  2011-10  1070704.67
11  2011-11  1461756.25
12  2011-12   433668.01
```

```
# 5.5 Revenue by Country
query5 = """
SELECT "Country", ROUND(SUM(("Quantity" * "UnitPrice")::numeric),2) AS Revenue
FROM sales
GROUP BY "Country"
ORDER BY Revenue DESC;
"""
print("\nRevenue by Country:")
print(pd.read_sql(query5, engine))
```

```
Revenue by Country:
                Country     revenue
0        United Kingdom  8187806.36
1           Netherlands   284661.54
2                  EIRE   263276.82
3               Germany   221698.21
4                France   197403.90
5             Australia   137077.27
6           Switzerland    56385.35
7                 Spain    54774.58
8               Belgium    40910.96
9                Sweden    36595.91
10                Japan    35340.62
11               Norway    35163.46
12             Portugal    29367.02
13              Finland    22326.74
14      Channel Islands    20086.29
15              Denmark    18768.14
16                Italy    16890.51
17               Cyprus    12946.29
18              Austria    10154.32
19            Hong Kong    10117.04
20            Singapore     9120.39
21               Israel     7907.82
22               Poland     7213.14
23          Unspecified     4749.79
24               Greece     4710.52
25              Iceland     4310.00
26               Canada     3666.38
27                Malta     2505.47
28  United Arab Emirates     1902.28
29                  USA     1730.92
30              Lebanon     1693.88
31            Lithuania     1661.06
32   European Community     1291.75
33               Brazil     1143.60
34                  RSA     1002.31
35       Czech Republic      707.72
36              Bahrain      548.40
37         Saudi Arabia      131.17
```

```
# 5.6 Return Rate (% of invoices with negative quantity)
query6 = """
SELECT ROUND(100.0 * SUM(CASE WHEN "Quantity" < 0 THEN 1 ELSE 0 END)::numeric / COUNT(*), 2) AS Return_Rate
FROM sales;
"""
print("\nReturn Rate (%):")
print(pd.read_sql(query6, engine))
```

```
Return Rate (%):
   return_rate
0         1.96
```

```
# 5.7 Repeat Customers (Active >3 months)
query7 = """
SELECT "CustomerID", COUNT(DISTINCT TO_CHAR(TO_TIMESTAMP("InvoiceDate", 'MM/DD/YYYY HH24:MI'), 'YYYY-MM')) AS Active_Months
FROM sales
GROUP BY "CustomerID"
HAVING COUNT(DISTINCT TO_CHAR(TO_TIMESTAMP("InvoiceDate", 'MM/DD/YYYY HH24:MI'), 'YYYY-MM')) > 3
ORDER BY Active_Months DESC
LIMIT 10;
"""
print("\nTop Repeat Customers (Active >3 months):")
print(pd.read_sql(query7, engine))
```

```
Top Repeat Customers (Active >3 months):
   CustomerID  active_months
0     12971.0             13
1     13089.0             13
2     12841.0             13
3     12748.0             13
4     13069.0             13
5     13078.0             13
6     12720.0             13
7     12709.0             13
8     12682.0             13
9     13199.0             13
```

```
# Add your new code here
```

Task

Analyze the sales data to provide insights relevant to product and project managers, including product performance, customer behavior, sales trends, geographical analysis, and return analysis. Summarize the findings and provide actionable recommendations.

Product performance analysis

Subtask:

Identify top-selling and underperforming products.

Reasoning: Write and execute SQL queries to find the top 10 best-selling products and the 10 products with the lowest sales quantity using the established database connection, then load the results into pandas DataFrames.

```
# 5.2 Top 10 Best-Selling Products (revisiting for clarity and to load into df)
query2 = """
SELECT "Description", SUM("Quantity") AS Total_Sold
FROM sales
GROUP BY "Description"
ORDER BY Total_Sold DESC
LIMIT 10;
"""
top_selling_products_df = pd.read_sql(query2, engine)
print("\nTop 10 Best-Selling Products:")
display(top_selling_products_df)

# Query to identify the 10 products with the lowest sales quantity
# Excluding products with negative quantity (returns) and potentially those with 0 quantity if they represent stock items not sold.
# Also excluding descriptions that are likely not actual products based on common data cleaning practices (e.g., 'POSTAGE', 'DABA', 'CRUK').
query_lowest_sales = """
SELECT "Description", SUM("Quantity") AS Total_Sold
FROM sales
WHERE "Quantity" > 0
  AND "Description" IS NOT NULL
  AND "Description" NOT IN ('POSTAGE', 'DABA', 'CRUK', 'DOT', 'BANK CHARGES', 'ADJUST', 'Manual')
GROUP BY "Description"
ORDER BY Total_Sold ASC
LIMIT 10;
"""
lowest_selling_products_df = pd.read_sql(query_lowest_sales, engine)
print("\n10 Products with Lowest Sales Quantity (Excluding Returns/Non-Products):")
display(lowest_selling_products_df)
```

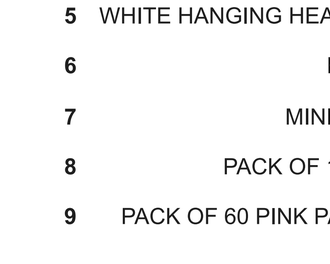
```
Top 10 Best-Selling Products:
```

	Description	total_sold
0	WORLD WAR 2 GLIDERS ASSTD DESIGNS	53847.0
1	JUMBO BAG RED RETROSPOT	47363.0
2	ASSORTED COLOUR BIRD ORNAMENT	36381.0
3	POPCORN HOLDER	36334.0
4	PACK OF 72 RETROSPOT CAKE CASES	36039.0
5	WHITE HANGING HEART T-LIGHT HOLDER	35317.0
6	RABBIT NIGHT LIGHT	30680.0
7	MINI PAINT SET VINTAGE	26437.0
8	PACK OF 12 LONDON TISSUES	26315.0
9	PACK OF 60 PINK PAISLEY CAKE CASES	24753.0

```
10 Products with Lowest Sales Quantity (Excluding Returns/Non-Products):
```

	Description	total_sold
0	PINK FEATHER CHRISTMAS DECORATION	1.0
1	Amazon Adjustment	1.0
2	SET 36 COLOURING PENCILS DOILEY	1.0
3	SET 10 CARDS CHRISTMAS BAUBLE 16954	1.0
4	SET 10 CARDS 3 WISE MEN 17107	1.0
5	BLUE CRYSTAL BOOT PHONE CHARM	1.0
6	JARDIN ETCHED GLASS BUTTER DISH	1.0
7	GREEN PENDANT SHELL NECKLACE	1.0
8	SET 10 CARDS SNOWY ROBIN 17099	1.0
9	BLUE NEW BAROQUE FLOCK CANDLESTICK	1.0

Next steps: Generate code with `top_selling_products_df` | View recommended plots | New interactive sheet | Generate code with `lowest_selling_products_df` | View recommended plots | New interactive sheet



Customer analysis

Subtask:

Understand customer behavior, including top customers, repeat customers, and their spending habits.

Reasoning: Execute the SQL queries to retrieve the top 10 customers by spend and the top repeat customers and display the results.

```
# 5.3 Top 10 Customers by Spend
query3 = """
SELECT "CustomerID", ROUND(SUM(("Quantity" * "UnitPrice")::numeric),2) AS Revenue
FROM sales
WHERE "CustomerID" IS NOT NULL
```

```
GROUP BY "CustomerID"
ORDER BY Revenue DESC
LIMIT 10;
"""
top_spending_customers_df = pd.read_sql(query3, engine)
print("\nTop 10 Customers by Spend:")
display(top_spending_customers_df)


# 5.7 Repeat Customers (Active >3 months)
query7 = """
SELECT "CustomerID", COUNT(DISTINCT TO_CHAR(TO_TIMESTAMP("InvoiceDate", 'MM/DD/YYYY HH24:MI'), 'YYYY-MM')) AS Active_Months
FROM sales
GROUP BY "CustomerID"
HAVING COUNT(DISTINCT TO_CHAR(TO_TIMESTAMP("InvoiceDate", 'MM/DD/YYYY HH24:MI'), 'YYYY-MM')) > 3
ORDER BY Active_Months DESC
LIMIT 10;
"""
repeat_customers_df = pd.read_sql(query7, engine)
print("\nTop Repeat Customers (Active >3 months):")
display(repeat_customers_df)
```

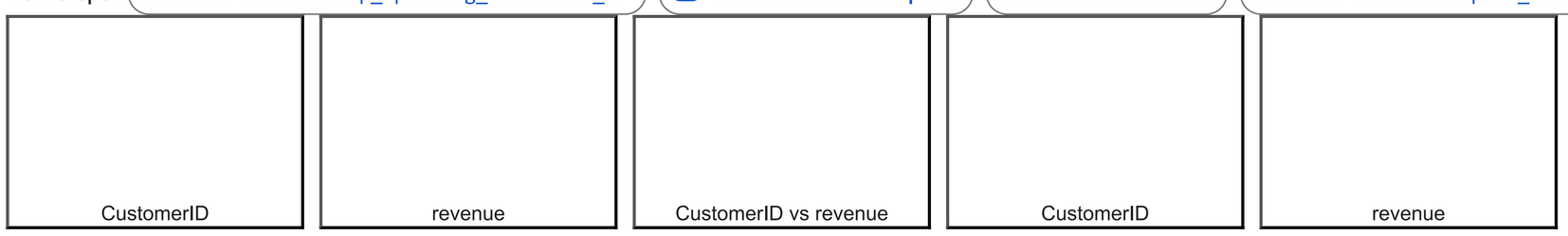
```
Top 10 Customers by Spend:
```

	CustomerID	revenue
0	14646.0	279489.02
1	18102.0	256438.49
2	17450.0	187482.17
3	14911.0	132572.62
4	12415.0	123725.45
5	14156.0	113384.14
6	17511.0	88125.38
7	16684.0	65892.08
8	13694.0	62653.10
9	15311.0	59419.34

```
Top Repeat Customers (Active >3 months):
```

	CustomerID	active_months
0	12971.0	13
1	13089.0	13
2	12841.0	13
3	12748.0	13
4	13069.0	13
5	13078.0	13
6	12720.0	13
7	12709.0	13
8	12682.0	13
9	13199.0	13

Next steps: Generate code with top_spending_customers_df | View recommended plots | New interactive sheet | Generate code with repeat_customers_df | View recommended plots | New interactive sheet



Sales trend analysis

Subtask:

Examine revenue trends over time (monthly, quarterly, etc.) and identify seasonality.

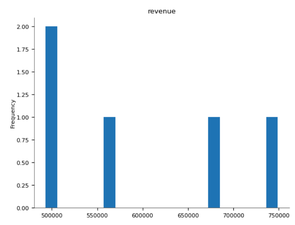
Reasoning: Retrieve the monthly revenue trend data from the database and load it into a pandas DataFrame, then display the first few rows.

```
# Step 1 & 2: Retrieve monthly revenue trend and load into DataFrame
query4 = """
SELECT TO_CHAR(TO_TIMESTAMP("InvoiceDate", 'MM/DD/YYYY HH24:MI'), 'YYYY-MM') AS Month,
       ROUND(SUM(("Quantity" * "UnitPrice")::numeric), 2) AS Revenue
FROM sales
GROUP BY Month
ORDER BY Month;
"""
monthly_revenue_df = pd.read_sql(query4, engine)
print("\nMonthly Revenue Trend:")
display(monthly_revenue_df.head())
```

```
Monthly Revenue Trend:
```

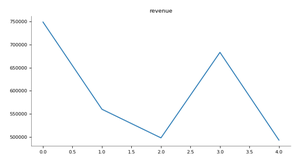
	month	revenue
0	2010-12	748957.02
1	2011-01	560000.26
2	2011-02	498062.65
3	2011-03	683267.08
4	2011-04	493207.12

Distributions



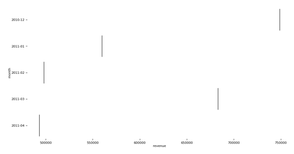
Categorical distributions

Values



Faceted distributions

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```



Reasoning: Create a line plot to visualize the monthly revenue trend and analyze the plot for seasonality.

```
# Step 3: Create a line plot for monthly revenue trend
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))
sns.lineplot(data=monthly_revenue_df, x='month', y='revenue')
plt.title('Monthly Revenue Trend')
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Step 4: Analyze the line plot to identify trends and seasonality
# The plot will show the revenue fluctuations over the months.
# Look for increasing/decreasing trends and recurring patterns (seasonality).
```


Geographical analysis

Subtask:

Analyze sales and revenue by country or region.

Reasoning: Retrieve the revenue by country from the database, load it into a pandas DataFrame, and display the DataFrame.

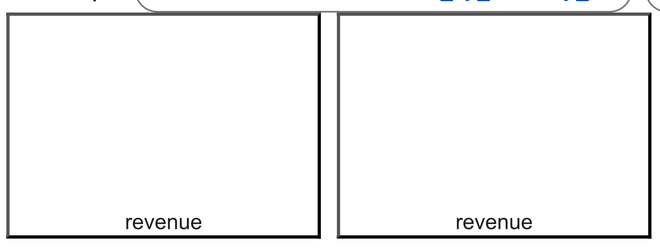
```
# 5.5 Revenue by Country (revisiting to load into df)
query5 = """
SELECT "Country", ROUND(SUM(("Quantity" * "UnitPrice")::numeric),2) AS Revenue
FROM sales
GROUP BY "Country"
ORDER BY Revenue DESC;
```

```
"""
revenue_by_country_df = pd.read_sql(query5, engine)
print("\nRevenue by Country:")
display(revenue_by_country_df)
```

```
Revenue by Country:
```

	Country	revenue
0	United Kingdom	8187806.36
1	Netherlands	284661.54
2	EIRE	263276.82
3	Germany	221698.21
4	France	197403.90
5	Australia	137077.27
6	Switzerland	56385.35
7	Spain	54774.58
8	Belgium	40910.96
9	Sweden	36595.91
10	Japan	35340.62
11	Norway	35163.46
12	Portugal	29367.02
13	Finland	22326.74
14	Channel Islands	20086.29
15	Denmark	18768.14
16	Italy	16890.51
17	Cyprus	12946.29
18	Austria	10154.32
19	Hong Kong	10117.04
20	Singapore	9120.39
21	Israel	7907.82
22	Poland	7213.14
23	Unspecified	4749.79
24	Greece	4710.52
25	Iceland	4310.00
26	Canada	3666.38
27	Malta	2505.47
28	United Arab Emirates	1902.28
29	USA	1730.92
30	Lebanon	1693.88
31	Lithuania	1661.06
32	European Community	1291.75
33	Brazil	1143.60
34	RSA	1002.31
35	Czech Republic	707.72
36	Bahrain	548.40
37	Saudi Arabia	131.17

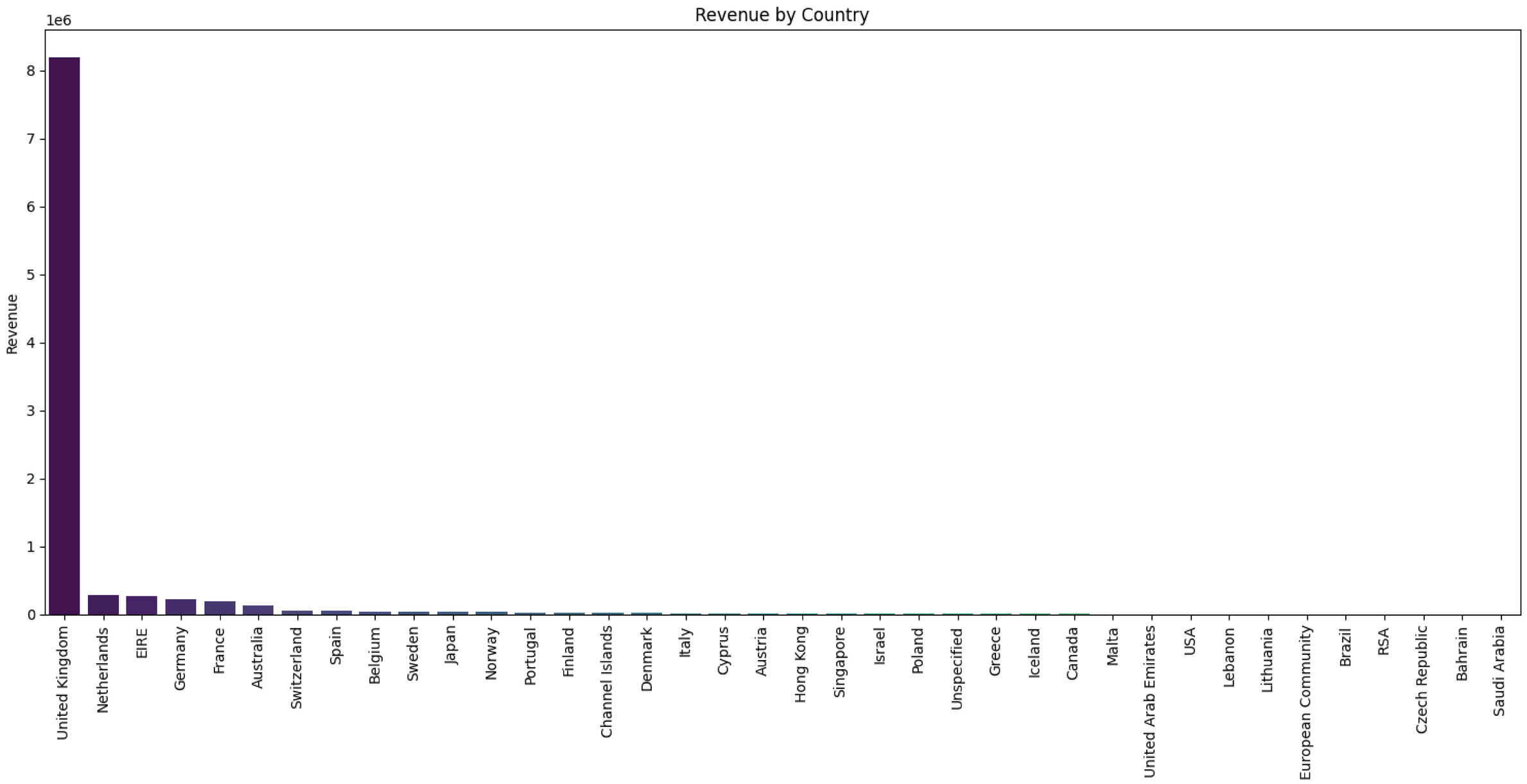
Next steps: Generate code with `revenue_by_country_df` View recommended plots New interactive sheet



Reasoning: Create a bar plot to visualize the revenue by country and analyze the plot to identify countries with the highest and lowest revenue.

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(15, 8))
sns.barplot(data=revenue_by_country_df, x='Country', y='revenue', palette='viridis')
plt.title('Revenue by Country')
plt.xlabel('Country')
plt.ylabel('Revenue')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# Analyze the bar plot:
# The bar plot clearly shows that the United Kingdom has significantly higher revenue compared to other countries.
# The countries with the lowest revenue can be identified from the tail end of the sorted bar plot (Saudi Arabia, Bahrain, Czech Republic, RSA, Brazil).
```

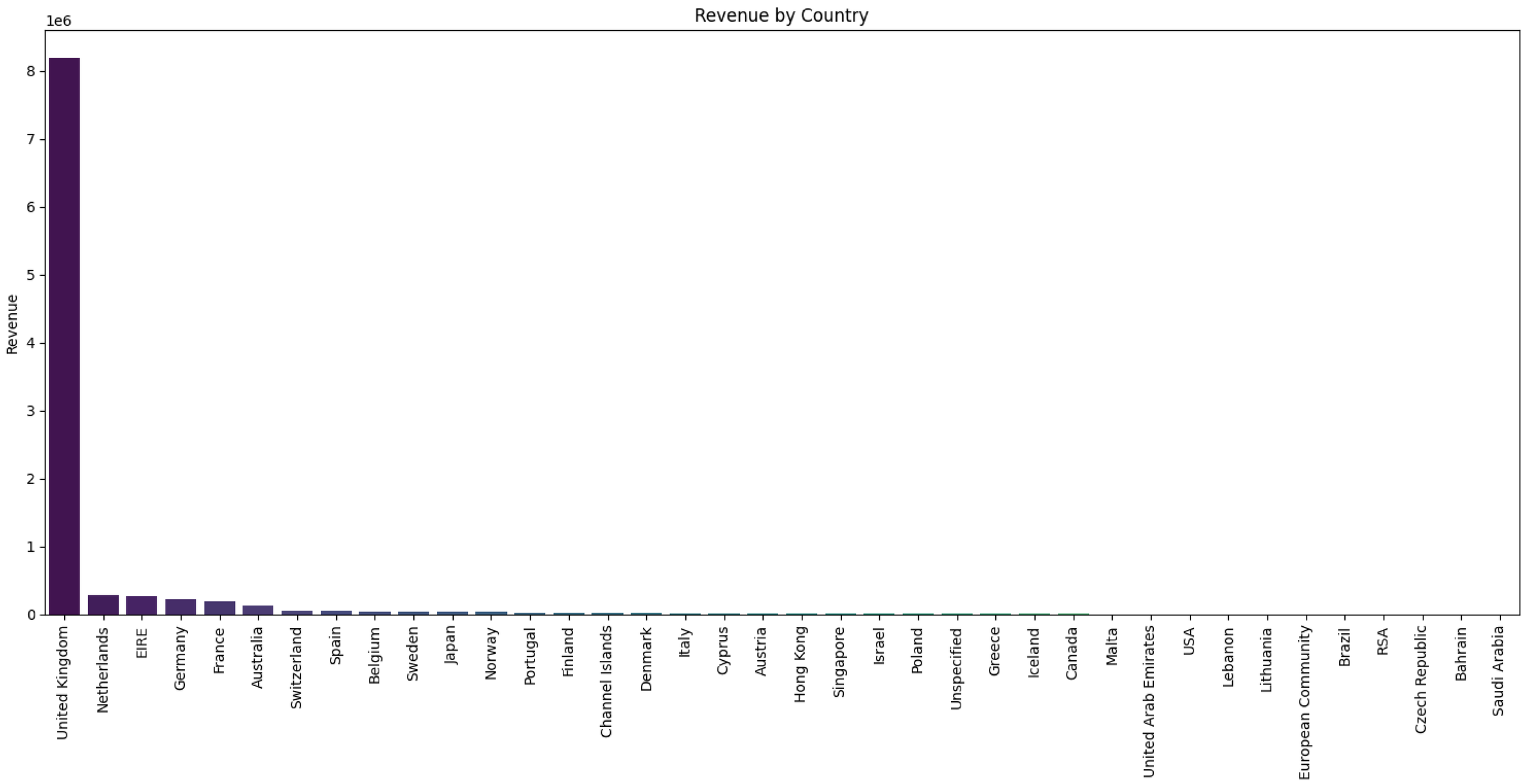
```
/tmp/ipython-input-3533896936.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=revenue_by_country_df, x='Country', y='revenue', palette='viridis')
```



Reasoning: The previous command failed because of a KeyError. The column name 'country' is not in the DataFrame. I need to check the actual column names in `revenue_by_country_df` and use the correct name, which is likely 'Country' based on the previous output. I will regenerate the bar plot with the correct column name.

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(15, 8))
sns.barplot(data=revenue_by_country_df, x='Country', y='revenue', palette='viridis')
plt.title('Revenue by Country')
plt.xlabel('Country')
plt.ylabel('Revenue')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# Analyze the bar plot:
# The bar plot clearly shows that the United Kingdom has significantly higher revenue compared to other countries.
# The countries with the lowest revenue can be identified from the tail end of the sorted bar plot (Saudi Arabia, Bahrain, Czech Republic, RSA, Brazil).
```

```
/tmp/ipython-input-3533896936.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=revenue_by_country_df, x='Country', y='revenue', palette='viridis')
```



Return analysis

Subtask:

Investigate return rates and identify potential issues.

Reasoning: Retrieve the return rate percentage by executing a SQL query against the 'sales' table using the 'engine' connection and load the result into a pandas DataFrame, then print the DataFrame.

```
# 5.6 Return Rate (% of invoices with negative quantity) - revisiting to load into df
query6 = """
SELECT ROUND(100.0 * SUM(CASE WHEN "Quantity" < 0 THEN 1 ELSE 0 END)::numeric / COUNT(*), 2) AS Return_Rate
FROM sales;
"""
return_rate_df = pd.read_sql(query6, engine)
print("\nReturn Rate (%):")
display(return_rate_df)
```

```
Return Rate (%):
```

	return_rate
0	1.96

Data visualization

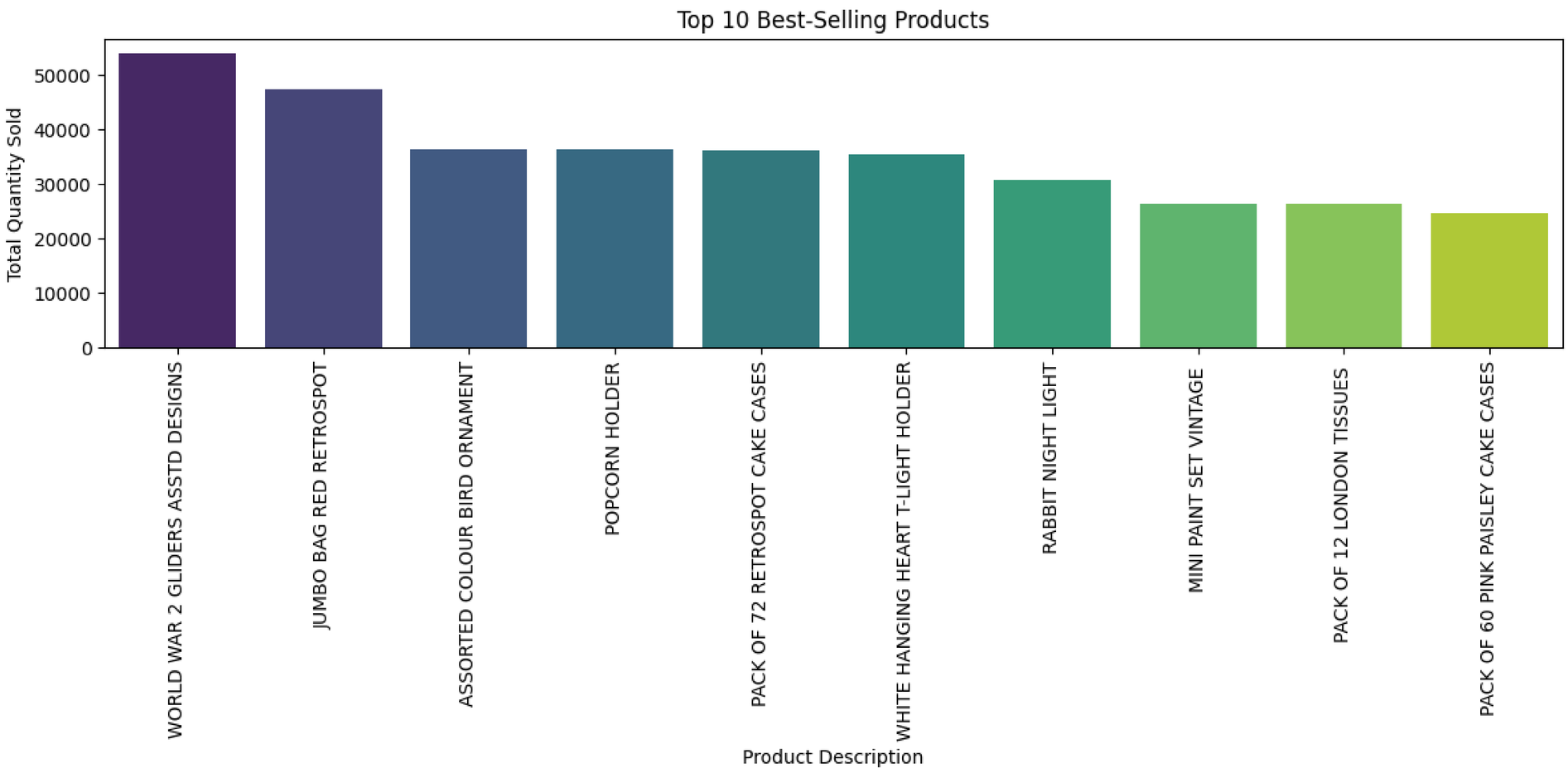
Subtask:

Create visualizations (charts, graphs) to present key findings effectively.

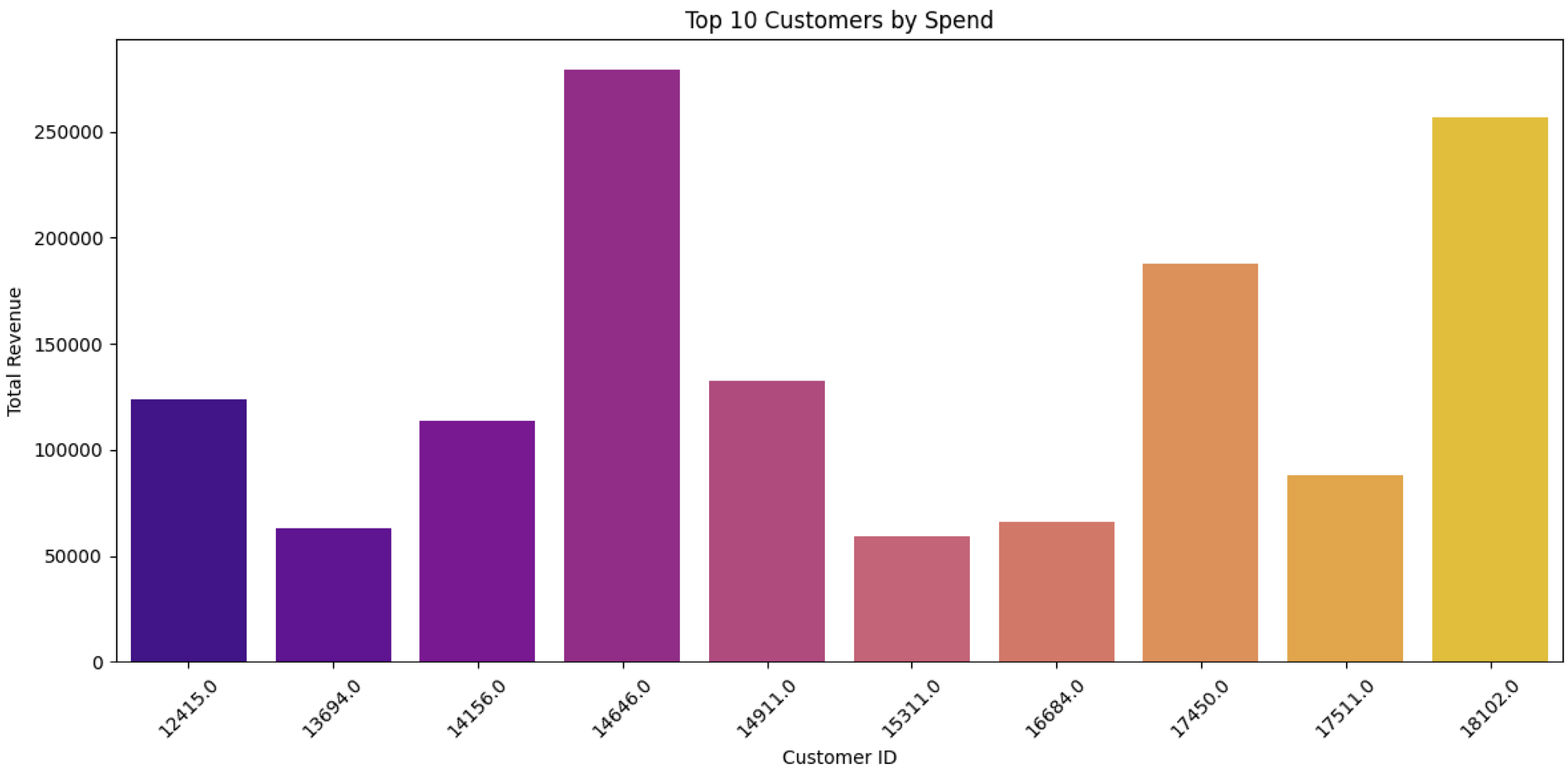
Reasoning: Create bar charts for the top 10 best-selling products, top 10 customers by spend, and top repeat customers using the previously generated dataframes.

```
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Bar chart for Top 10 Best-Selling Products
plt.figure(figsize=(12, 6))
sns.barplot(data=top_selling_products_df, x='Description', y='total_sold', palette='viridis')
plt.title('Top 10 Best-Selling Products')
plt.xlabel('Product Description')
plt.ylabel('Total Quantity Sold')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# 2. Bar chart for Top 10 Customers by Spend
plt.figure(figsize=(12, 6))
sns.barplot(data=top_spending_customers_df, x='CustomerID', y='revenue', palette='plasma')
plt.title('Top 10 Customers by Spend')
plt.xlabel('Customer ID')
plt.ylabel('Total Revenue')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 3. Bar chart for Top Repeat Customers
plt.figure(figsize=(12, 6))
sns.barplot(data=repeat_customers_df, x='CustomerID', y='active_months', palette='coolwarm')
plt.title('Top Repeat Customers (Active >3 months)')
plt.xlabel('Customer ID')
plt.ylabel('Number of Active Months')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

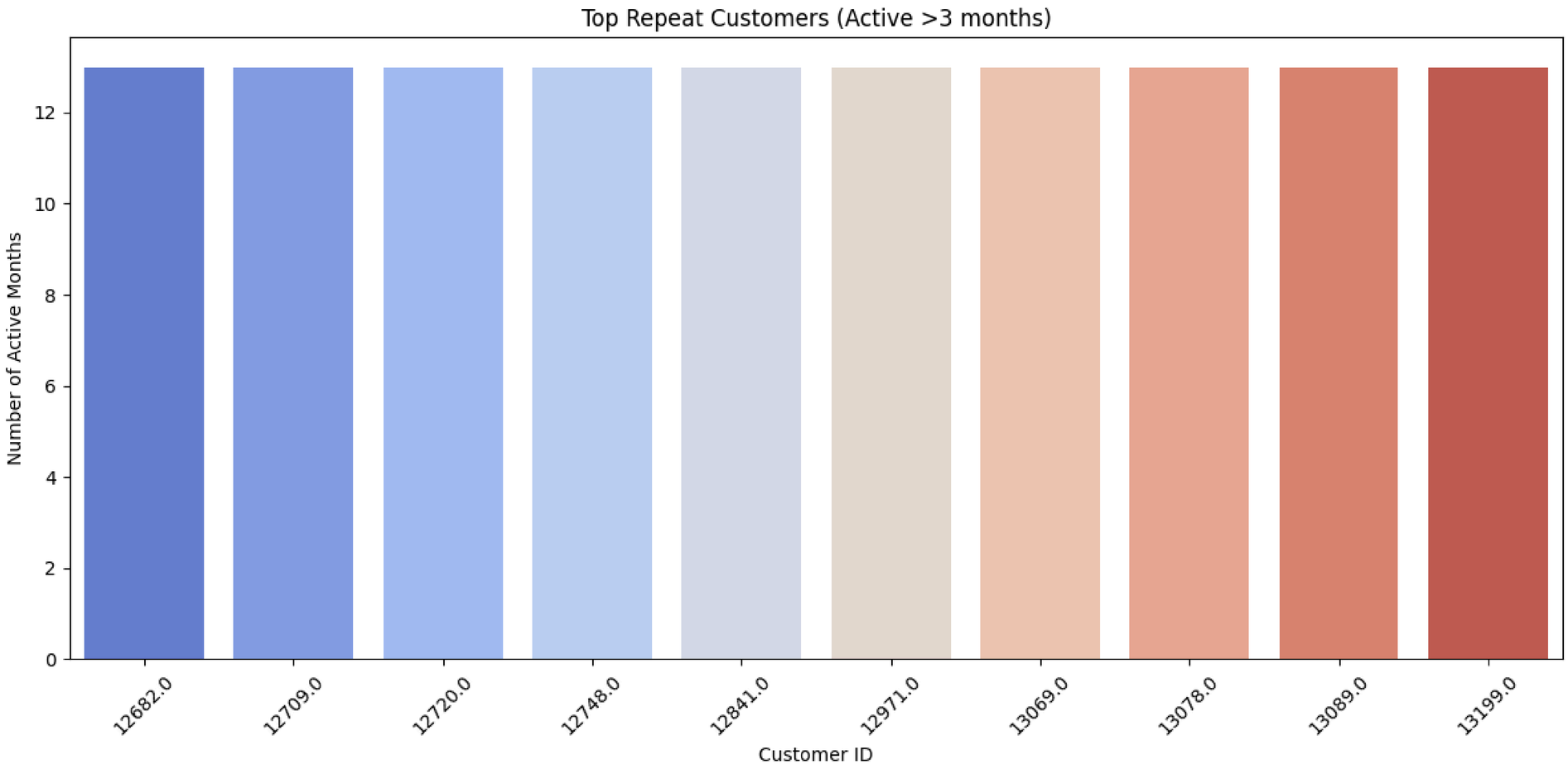
```
/tmp/ipython-input-3766104402.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=top_selling_products_df, x='Description', y='total_sold', palette='viridis')
```




```
/tmp/ipython-input-3766104402.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=top_spending_customers_df, x='CustomerID', y='revenue', palette='plasma')
```




```
/tmp/ipython-input-3766104402.py:26: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=repeat_customers_df, x='CustomerID', y='active_months', palette='coolwarm')
```



Summary and recommendations

Subtask:

Summarize the findings and provide actionable recommendations for product and project managers.