Business Questions:

1. Where do profits come from?

Which products/ categories and Countries drive revenue vs. margin after accounting for returns/ cancellations?

Category: Lightning & Fixtures
Products: Stock Code M and 22423
Country: United Kingdom

2. Who are our best customers and How do we keep them?

Build RFM Segments (Recency-Frequency-Monetary) and derive actions (e.g. "win-back", "VIP nurture", "new growth")

Best Customer: Customer ID 18102

3. When do we sell the most and what should we stock?

Identify seasonality & trends in monthly revenue; quantify the impact of returns and large-order outliers. Use a simple baseline forecast to inform inventory planning.

Highest Daily Revenue: 01/03/2009

Highest Weekly Revenue: 03/01/2009

Highest Monthly Revenue: Mar 2009

Python Coding below:

```
import polars as pl

import pandas as pd

import matplotlib.pyplot as plt


df = pl.read_csv(
    "online_retail_II.csv",
    ignore_errors=True,         # skip bad lines
    infer_schema_length=0,      # scan all rows for correct dtypes
    low_memory=True,            # chunked parsing
    try_parse_dates=True        # parsing dates
)
```

[1]: shape: (1_048_575, 8)

Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
str	str	str	str	str	str	str	str
"489434"	"85048"	"15CM CHRISTMAS GLASS BALL 20 L...	"12"	"01/12/2009 7:45"	"6.95"	"13085"	"United Kingdom"
"489434"	"79323P"	"PINK CHERRY LIGHTS"	"12"	"01/12/2009 7:45"	"6.75"	"13085"	"United Kingdom"
"489434"	"79323W"	" WHITE CHERRY LIGHTS"	"12"	"01/12/2009 7:45"	"6.75"	"13085"	"United Kingdom"
"489434"	"22041"	"RECORD FRAME 7" SINGLE SIZE "	"48"	"01/12/2009 7:45"	"2.1"	"13085"	"United Kingdom"
"489434"	"21232"	"STRAWBERRY CERAMIC TRINKET BOX"	"24"	"01/12/2009 7:45"	"1.25"	"13085"	"United Kingdom"
...	...	...	...	...	...	...	...
"580501"	"23284"	"DOORMAT KEEP CALM AND COME IN"	"2"	"04/12/2011 13:00"	"8.25"	"14546"	"United Kingdom"
"580501"	"22507"	"MEMO BOARD RETROSPOT DESIGN"	"3"	"04/12/2011 13:00"	"4.95"	"14546"	"United Kingdom"
"580502"	"22469"	"HEART OF WICKER SMALL"	"3"	"04/12/2011 13:15"	"1.65"	"16931"	"United Kingdom"
"580502"	"23489"	"VINTAGE BELLS GARLAND"	"2"	"04/12/2011 13:15"	"2.89"	"16931"	"United Kingdom"
"580502"	"23046"	"PAPER LANTERN 9 POINT DELUXE S...	"1"	"04/12/2011 13:15"	"6.65"	"16931"	"United Kingdom"

Clean invoice date change from str to date only

Parsing only Invoice Date column then reuse

```
df = df.with_columns([
    pl.col("InvoiceDate")
      .str.strptime(pl.Datetime, "%m/%d/%Y %H:%M", strict=False)
      .alias("Invoice_DateTime")
])
```

```
df = df.with_columns([
    pl.col("Invoice_DateTime").dt.date().alias("Invoice_Date"),
    pl.col("Invoice_DateTime").dt.strftime("%b %Y").alias("Month-YYYY"),  # MMM YYYY Format only
    # pl.col("Invoice_DateTime").dt.strftime("%b").alias("Month"),   # MMM Format only

    # Parse as MM-01-YYYY → convert back to Date type properly
    pl.col("Invoice_DateTime")
        .dt.strftime("%m-01-%Y")              # create string like "01-01-2011"
        .str.strptime(pl.Date, "%m-%d-%Y", strict=False)  # convert string to Date
        .alias("Month-MM-dd-YYYY"),
```

```
    pl.col("Invoice_DateTime")
        .dt.truncate("1w")      # start of the week (Monday)
        .dt.date()              # convert to date only
        .alias("Week")
])

# Clean space description
df = df.with_columns([
    pl.col("Description").str.strip_chars(" ,.;+").alias("Description") # removes spaces, commas, periods, semicolons, plus
])

# Define categorization function
def categorize(desc: str) -> str:
    desc = str(desc).lower()
    if "christmas" in desc or "xmas" in desc or "bells" in desc or "decoration" in desc:
        return "Christmas Decorations"
    elif "light" in desc or "lamp" in desc or "sign" in desc or "hook" in desc or "door" in desc or "storage" in desc \
    or "kit" in desc or "box" in desc or "block" in desc or "set" in desc or "mat" in desc or "gardening" in desc \
    or "metal" in desc or "frame" in desc or "stool" in desc or "stick" in desc or "wall" in desc:
        return "Lighting & Fixtures"
    elif "cup" in desc or "mug" in desc or "tea" in desc or "plate" in desc or "lunch" in desc or "cutlery" in desc \
    or "bottle" in desc or "coaster" in desc or "pot" in desc or "ceramic" in desc or "glass" in desc or "goblet" in desc \
    or "basket" in desc or "cases" in desc or "stand" in desc or "apron" in desc or "holder" in desc  \
    or "apron" in desc or "torch" in desc or "bowl" in desc or "straw" in desc or "jar" in desc:
        return "Kitchenware"
    elif "bag" in desc or "bracelet" in desc or "slides" in desc or "comb" in desc or "brooch" in desc or "pin" in desc \
    or "ring" in desc or "patches" in desc or "sign" in desc or "hanger" in desc or "ornament" in desc or "knit" in desc \
    or "umbrella" in desc or "earrings" in desc or "button" in desc or "muff" in desc or "key" in desc or "" in desc:
        return "Bags & Accessories"
    elif "card" in desc or "wrap" in desc or "toys" in desc or "sticker" in desc or "memo" in desc or "doll" in desc \
    or "cover" in desc or "clip" in desc or "match" in desc or "marbles" in desc or "puzzles" in desc  \
```

```
    or "game" in desc or "box" in desc or "ribbons" in desc or "car" in desc or "notebook" in desc \
    or "book" in desc or "tag" in desc:
        return "Stationery & Gifts"
    elif "toilet" in desc:
        return "Toiletries"
    else:
        return "Other"

# Apply category mapping (vectorized via .map_elements)
df = df.with_columns([
    pl.col("Description")
      .map_elements(categorize, return_dtype=pl.Utf8)
      .alias("Category")
])

# Clean and cast numeric columns before computation
df = df.with_columns([
    pl.col("Quantity").cast(pl.Float64).alias("Quantity"),
    pl.col("Price").cast(pl.Utf8).str.replace_all(",", "").cast(pl.Float64).alias("Price_Clean"),
])

# Compute Revenue and flag returns
df = df.with_columns([
    (pl.col("Quantity") * pl.col("Price_Clean")).alias("Revenue"),
    ((pl.col("Invoice").cast(pl.Utf8).str.starts_with("C")) | (pl.col("Quantity") < 0))
        .alias("IsReturn")
])

# checking if invoice is return
df = df.with_columns([
    pl.when(pl.col("IsReturn")).then(0).otherwise(pl.col("Revenue")).alias("Net_Revenue"),
```

```
    pl.when(pl.col("IsReturn")).then(pl.col("Revenue")).otherwise(0).alias("Returns"),
])
```

```
print(df.dtypes)

print(df["Invoice","Invoice_Date","Week","Month-MM-dd-YYYY","Month-YYYY","Quantity","Revenue","Net_Revenue","Returns"].head(10))
```

```
[String, String, String, Float64, String, String, String, String, Datetime(time_unit='us', time_zone=None), Date, String, Date, Date, Strin
g, Float64, Float64, Boolean, Float64, Float64]
shape: (10, 9)
```

Invoice	Invoice_Date	Week	Month-MM-dd-YYYY	…	Quantity	Revenue	Net_Revenue	Returns
str	date	date	date		f64	f64	f64	f64
489434	2009-01-12	2009-01-12	2009-01-01	…	12.0	83.4	83.4	0.0
489434	2009-01-12	2009-01-12	2009-01-01	…	12.0	81.0	81.0	0.0
489434	2009-01-12	2009-01-12	2009-01-01	…	12.0	81.0	81.0	0.0
489434	2009-01-12	2009-01-12	2009-01-01	…	48.0	100.8	100.8	0.0
489434	2009-01-12	2009-01-12	2009-01-01	…	24.0	30.0	30.0	0.0
489434	2009-01-12	2009-01-12	2009-01-01	…	24.0	39.6	39.6	0.0
489434	2009-01-12	2009-01-12	2009-01-01	…	24.0	30.0	30.0	0.0
489434	2009-01-12	2009-01-12	2009-01-01	…	10.0	59.5	59.5	0.0
489435	2009-01-12	2009-01-12	2009-01-01	…	12.0	30.6	30.6	0.0
489435	2009-01-12	2009-01-12	2009-01-01	…	12.0	45.0	45.0	0.0

```
# Compute category summary
category_summary = (
    df.group_by("Category")
      .agg([
        pl.sum("Revenue").alias("Gross_Revenue"),
        pl.sum("Net_Revenue"),
        pl.sum("Returns")
      ])
      .sort("Net_Revenue", descending=True)
)

category_summary = category_summary.drop_nulls()
```

```
print(category_summary.head(10))
```

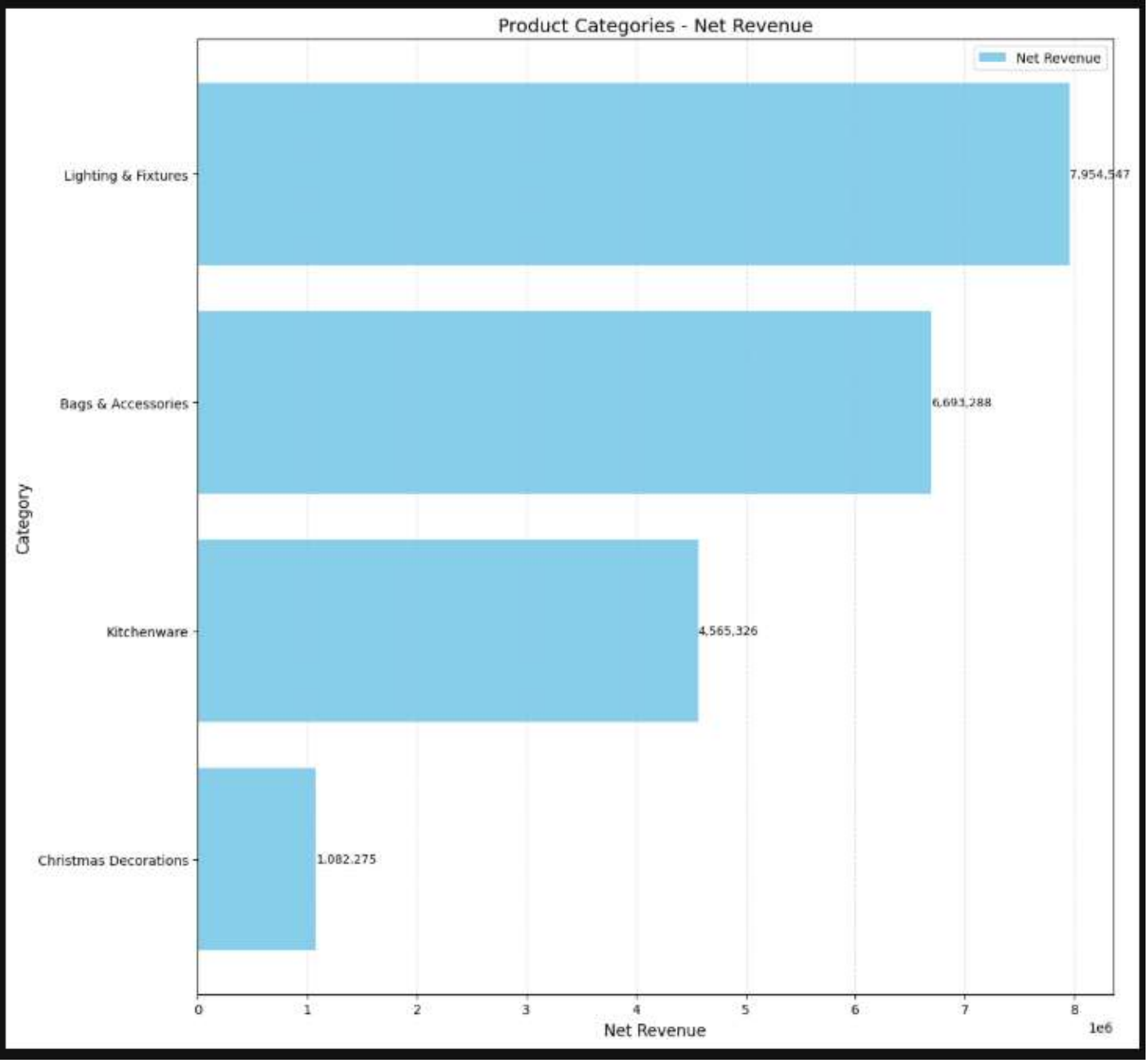
```
shape: (4, 4)
```

Category	Gross_Revenue	Net_Revenue	Returns
str	f64	f64	f64
Lighting & Fixtures	7.6673e6	7.9545e6	-287283.93
Bags & Accessories	5.7859e6	6.6933e6	-907353.05
Kitchenware	4.4546e6	4.5653e6	-110698.04
Christmas Decorations	1.0647e6	1.0823e6	-17607.88

```
# Horizontal Bar Chart - Comparing Product Categories
# Convert Polars DataFrame to pandas manually
category_pd = pd.DataFrame(category_summary.to_dicts())

# Sort by Net_Revenue descending (highest first)
category_pd = category_pd.sort_values(by="Net_Revenue", ascending=True)

plt.figure(figsize=(12, 11))  # Reasonable figure size
plt.barh(category_pd["Category"], category_pd["Net_Revenue"], color='skyblue', label="Net Revenue")

# Adding Labels, title, and legend
plt.title("Product Categories - Net Revenue", fontsize=14)
plt.xlabel("Net Revenue", fontsize=12)
plt.ylabel("Category", fontsize=12)
plt.grid(axis="x", linestyle="--", alpha=0.5)

# Labels on bars
for i, v in enumerate(category_pd["Net_Revenue"]):
    plt.text(v, i, f"{v:,.0f}", va="center", ha="left", fontsize=9, color="black")

plt.tight_layout()
plt.legend()

# Show the summary in table
print(category_summary.head(10))

# Show Chart
plt.show()
```

shape: (4, 4)

Category	Gross_Revenue	Net_Revenue	Returns
---	---	---	---
str	f64	f64	f64
Lighting & Fixtures	7.6673e6	7.9545e6	-287283.93
Bags & Accessories	5.7859e6	6.6933e6	-907353.05
Kitchenware	4.4546e6	4.5653e6	-110698.04
Christmas Decorations	1.0647e6	1.0823e6	-17607.88




```
# Stock Code Summary
stock_code_summary = (
    df.group_by("StockCode")
      .agg([
          pl.sum("Revenue").alias("Gross_Revenue"),
```

```
        pl.sum("Net_Revenue"),
        pl.sum("Returns")
    ])
    .sort("Net_Revenue", descending=True)
)

stock_code_summary = stock_code_summary.drop_nulls()
```

```
print(stock_code_summary.head(10))
```

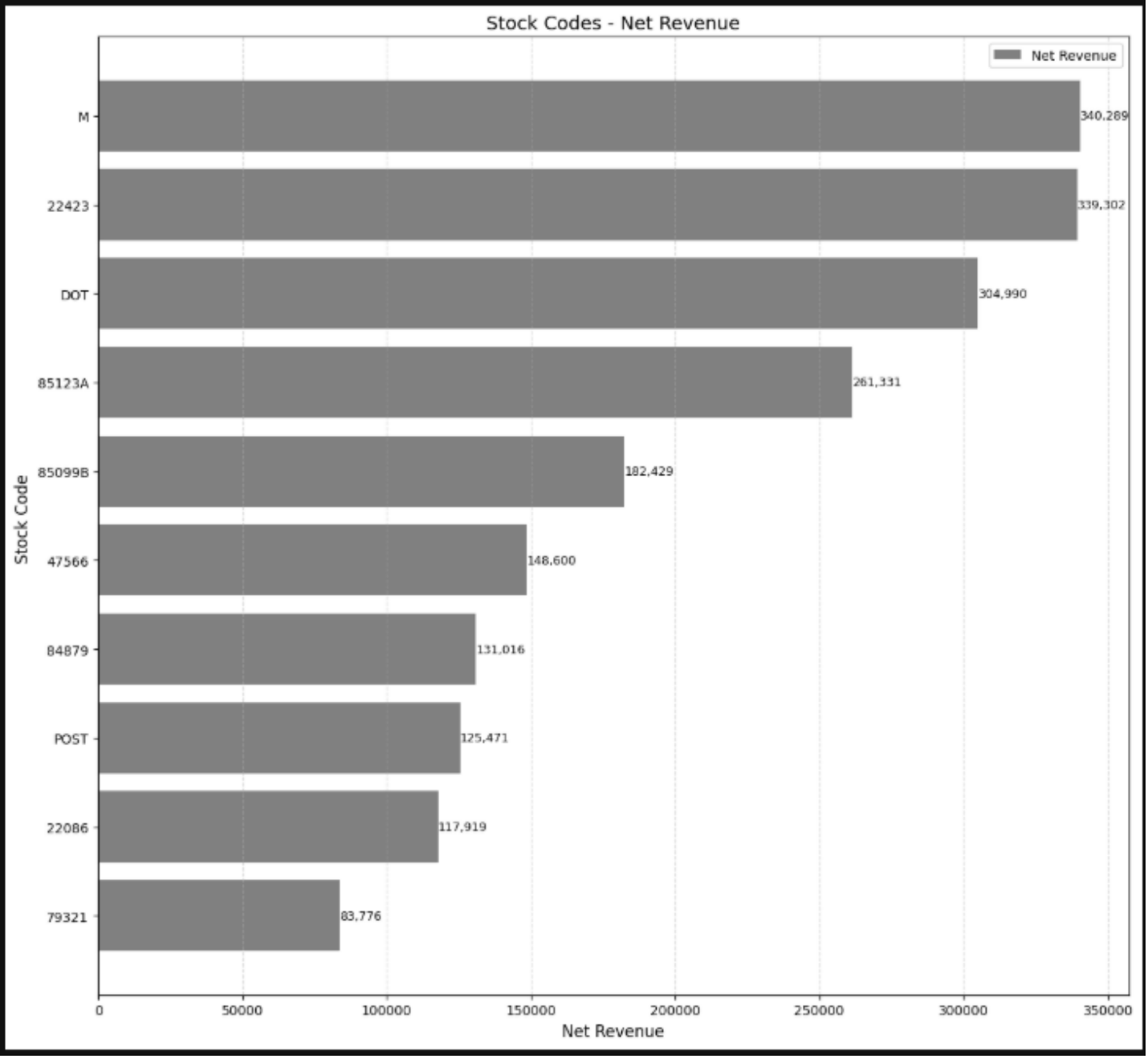
```
shape: (10, 4)
┌─────────────┬───────────────┬─────────────┬─────────┐
│ StockCode   ┆ Gross_Revenue ┆ Net_Revenue ┆ Returns │
│ ---         ┆ ---           ┆ ---         ┆ ---     │
│ str         ┆ f64           ┆ f64         ┆ f64     │
╞═════════════╪═══════════════╪═════════════╪═════════╡
│ M           ┆ -82862.89     ┆ 340288.82   ┆ -423151.71 │
│ 22423       ┆ 322643.15     ┆ 339301.55   ┆ -16658.4 │
│ DOT         ┆ 304979.93     ┆ 304989.94   ┆ -10.01  │
│ 85123A      ┆ 251944.69     ┆ 261331.39   ┆ -9386.7 │
│ 85099B      ┆ 180281.34     ┆ 182428.54   ┆ -2147.2 │
│ 47566       ┆ 147361.56     ┆ 148600.11   ┆ -1238.55 │
│ 84879       ┆ 130242.39     ┆ 131016.46   ┆ -774.07 │
│ POST        ┆ 110219.21     ┆ 125471.13   ┆ -15251.92 │
│ 22086       ┆ 116439.35     ┆ 117918.75   ┆ -1479.4 │
│ 79321       ┆ 83145.65      ┆ 83775.65    ┆ -630.0  │
└─────────────┴───────────────┴─────────────┴─────────┘
```

```
# Horizontal Bar Chart - Comparing Stock Code Categories

# Convert Polars DataFrame to pandas manually
stock_code_pd = pd.DataFrame(stock_code_summary.head(10).to_dicts())

# Sort by Net_Revenue descending (highest first)
stock_code_pd = stock_code_pd.sort_values(by="Net_Revenue", ascending=True)

plt.figure(figsize=(12, 11))  # Reasonable figure size
plt.barh(stock_code_pd["StockCode"], stock_code_pd["Net_Revenue"], color='gray', label="Net Revenue")

# Adding Labels, title, and legend
plt.title("Stock Codes - Net Revenue", fontsize=14)
```

```
plt.xlabel("Net Revenue", fontsize=12)
plt.ylabel("Stock Code ", fontsize=12)
plt.grid(axis="x", linestyle="--", alpha=0.5)

# Labels on bars
for i, v in enumerate(stock_code_pd["Net_Revenue"]):
    plt.text(v, i, f"{v:,.0f}", va="center", ha="left", fontsize=9, color="black")

plt.tight_layout()
plt.legend()

# Show the table summary
print(stock_code_summary.head(10))

# Show Chart
plt.show()
```

shape: (10, 4)

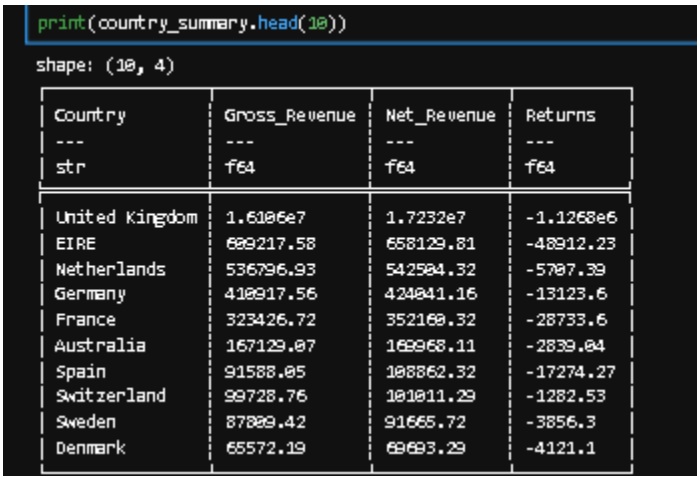
StockCode	Gross_Revenue	Net_Revenue	Returns
str	f64	f64	f64
M	-82862.89	340288.82	-423151.71
22423	322643.15	339301.55	-16658.4
DOT	304979.93	304989.94	-10.01
85123A	251944.69	261331.39	-9386.7
85099B	180281.34	182428.54	-2147.2
47566	147361.56	148600.11	-1238.55
84879	130242.39	131016.46	-774.07
POST	110219.21	125471.13	-15251.92
22086	116439.35	117918.75	-1479.4
79321	83145.65	83775.65	-630.0



Country Summary

country_summary = (

```
    df.group_by("Country")
      .agg([
          pl.sum("Revenue").alias("Gross_Revenue"),
          pl.sum("Net_Revenue"),
          pl.sum("Returns")
      ])
      .sort("Net_Revenue", descending=True)
)

country_summary = country_summary.drop_nulls()
```



```
print(country_summary.head(10))
shape: (10, 4)
```

Country	Gross_Revenue	Net_Revenue	Returns
str	f64	f64	f64
United Kingdom	1.6106e7	1.7232e7	-1.1268e6
EIRE	609217.58	658129.81	-48912.23
Netherlands	536796.93	542504.32	-5707.39
Germany	410917.56	424041.16	-13123.6
France	323426.72	352160.32	-28733.6
Australia	167129.07	169968.11	-2839.04
Spain	91588.05	108862.32	-17274.27
Switzerland	99728.76	101011.29	-1282.53
Sweden	87809.42	91665.72	-3856.3
Denmark	65572.19	69693.29	-4121.1

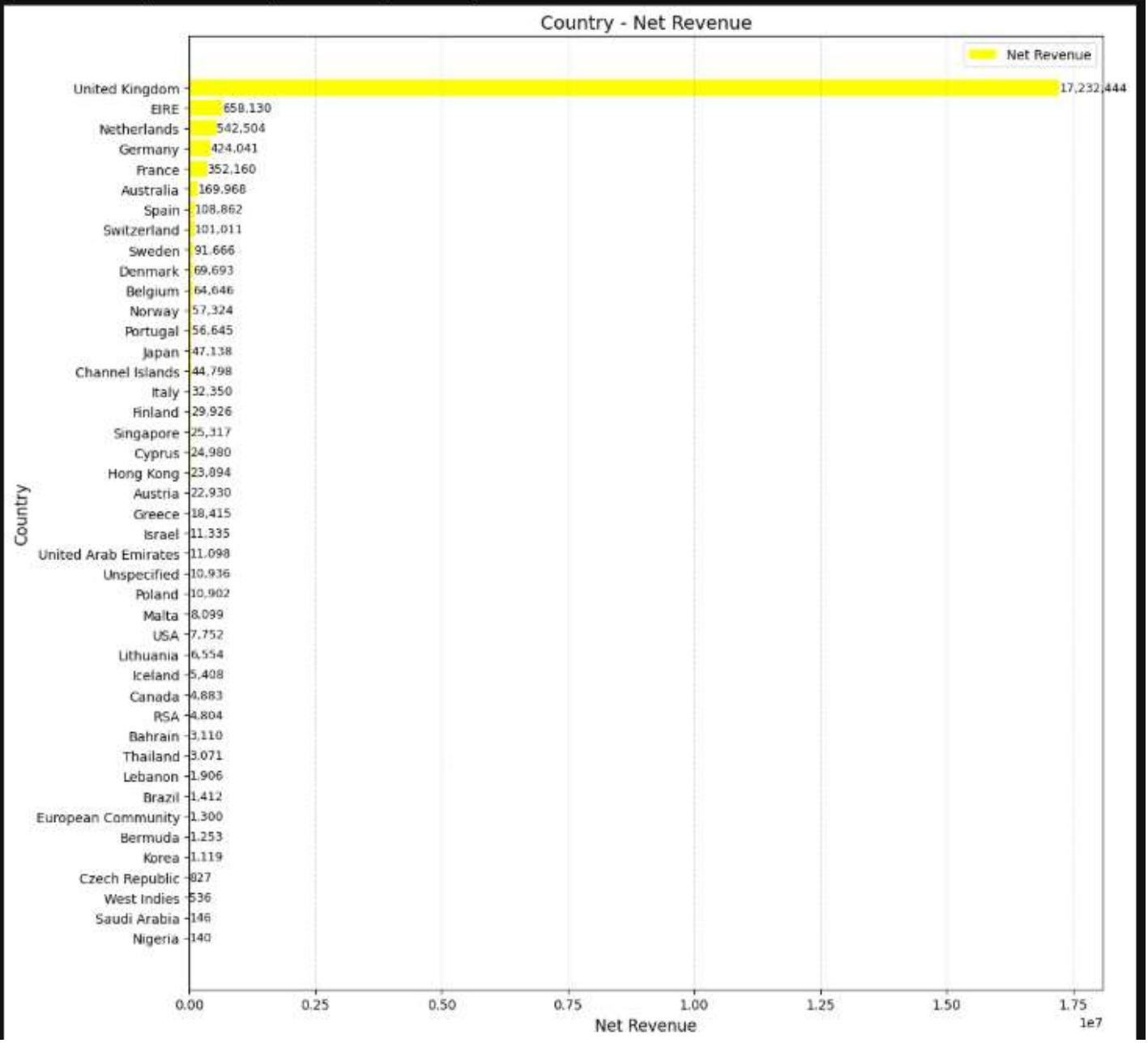
```
# Horizontal Bar Chart - Comparing Country Categories

# Convert Polars DataFrame to pandas manually
country_pd = pd.DataFrame(country_summary.to_dicts())

# Sort by Net_Revenue descending (highest first)
country_pd = country_pd.sort_values(by="Net_Revenue", ascending=True)

plt.figure(figsize=(12, 11))  # Reasonable figure size
plt.barh(country_pd["Country"], country_pd["Net_Revenue"], color='yellow', label="Net Revenue")

# Adding Labels, title, and legend
```

```
plt.title("Country - Net Revenue", fontsize=14)
plt.xlabel("Net Revenue", fontsize=12)
plt.ylabel("Country", fontsize=12)
plt.grid(axis="x", linestyle="--", alpha=0.5)

# Labels on bars
for i, v in enumerate(country_pd["Net_Revenue"]):
    plt.text(v, i, f"{v:,.0f}", va="center", ha="left", fontsize=9, color="black")

plt.tight_layout()
plt.legend()

# Show Summary in table
print(country_summary.head(10))

# Show Chart
plt.show()
```

```
plt.show()
```

```
shape: (10, 4)
```

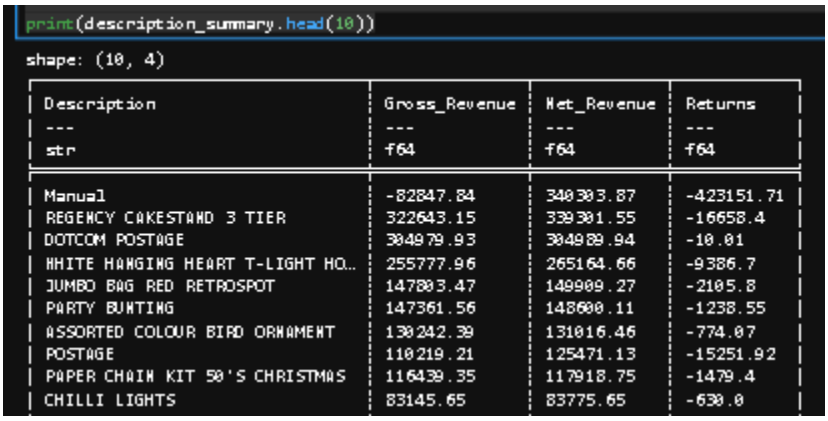
Country	Gross_Revenue	Net_Revenue	Returns
str	f64	f64	f64
United Kingdom	1.6106e7	1.7232e7	-1.1268e6
EIRE	609217.58	658129.81	-48912.23
Netherlands	536796.93	542504.32	-5707.39
Germany	410917.56	424041.16	-13123.6
France	323426.72	352160.32	-28733.6
Australia	167129.07	169968.11	-2839.04
Spain	91588.05	108862.32	-17274.27
Switzerland	99728.76	101011.29	-1282.53
Sweden	87809.42	91665.72	-3856.3
Denmark	65572.19	69693.29	-4121.1



Description Summary

description_summary = (

```
    df.group_by("Description")
      .agg([
         pl.sum("Revenue").alias("Gross_Revenue"),
         pl.sum("Net_Revenue"),
         pl.sum("Returns")
      ])
      .sort("Net_Revenue", descending=True)
)

description_summary = description_summary.drop_nulls()
```



```
print(description_summary.head(10))
shape: (10, 4)
┌──────────────────────────────┬───────────────┬─────────────┬─────────┐
│ Description                  ┆ Gross_Revenue ┆ Net_Revenue ┆ Returns │
│ ---                          ┆ ---           ┆ ---         ┆ ---     │
│ str                          ┆ f64           ┆ f64         ┆ f64     │
╞══════════════════════════════╪═══════════════╪═════════════╪═════════╡
│ Manual                       ┆ -82847.84     ┆ 340303.87   ┆ -423151.71 │
│ REGENCY CAKESTAND 3 TIER     ┆ 322643.15     ┆ 339301.55   ┆ -16658.4 │
│ DOTCOM POSTAGE               ┆ 304979.93     ┆ 304989.94   ┆ -10.01  │
│ WHITE HANGING HEART T-LIGHT HO… ┆ 255777.96     ┆ 265164.66   ┆ -9386.7 │
│ JUMBO BAG RED RETROSPOT      ┆ 147803.47     ┆ 149909.27   ┆ -2105.8 │
│ PARTY BUNTING                ┆ 147361.56     ┆ 148600.11   ┆ -1238.55 │
│ ASSORTED COLOUR BIRD ORNAMENT ┆ 130242.39     ┆ 131016.46   ┆ -774.07 │
│ POSTAGE                      ┆ 110219.21     ┆ 125471.13   ┆ -15251.92 │
│ PAPER CHAIN KIT 50'S CHRISTMAS ┆ 116439.35     ┆ 117918.75   ┆ -1479.4 │
│ CHILLI LIGHTS                ┆ 83145.65      ┆ 83775.65    ┆ -630.0  │
```

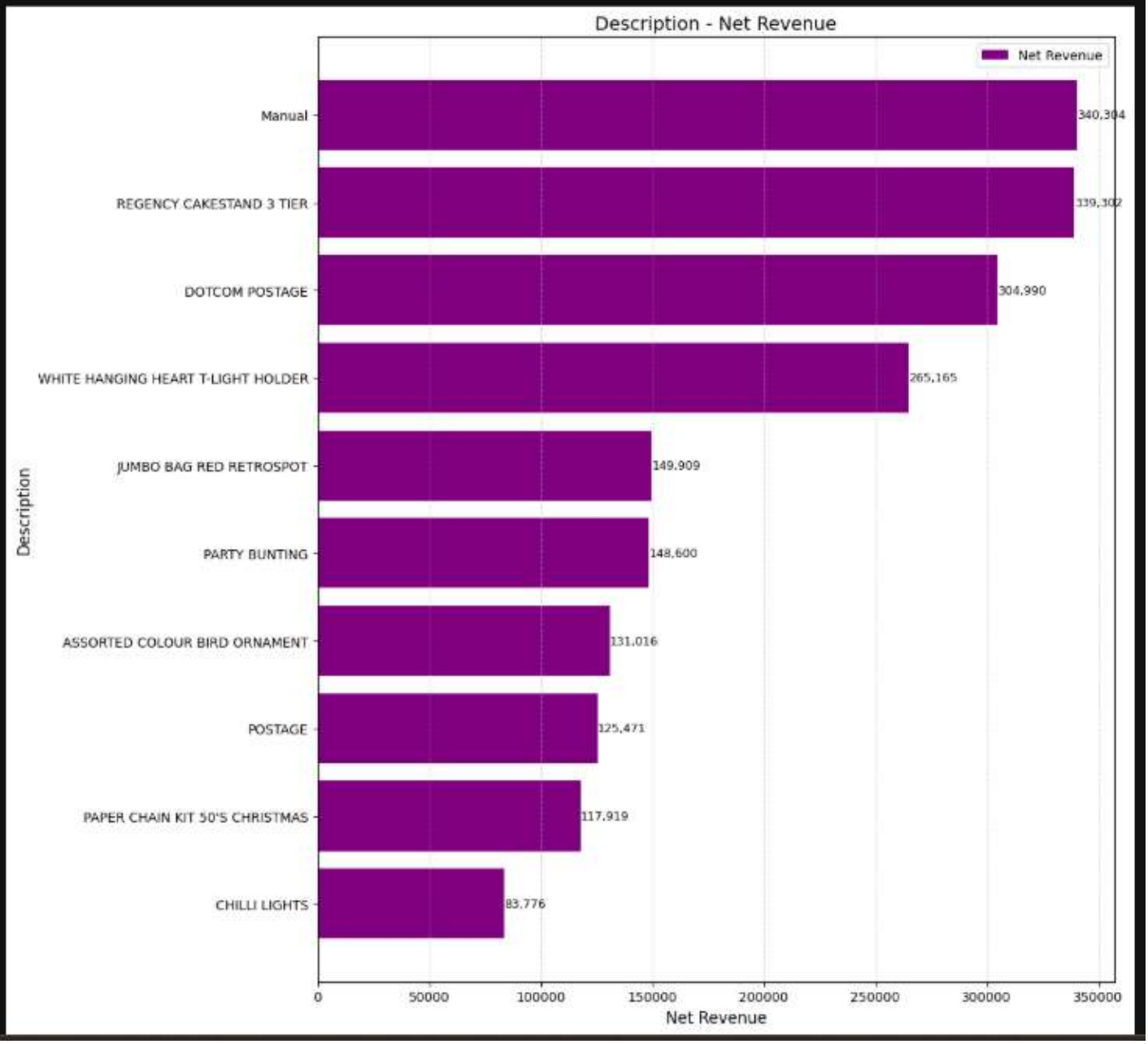
```
# Horizontal Bar Chart - Comparing Description Categories

# Convert Polars DataFrame to pandas manually

description_pd = pd.DataFrame(description_summary.head(10).to_dicts())


# Sort by Net_Revenue descending (highest first)

description_pd = description_pd.sort_values(by="Net_Revenue", ascending=True)


plt.figure(figsize=(12, 11))  # Reasonable figure size

plt.barh(description_pd["Description"], description_pd["Net_Revenue"], color='purple', label="Net Revenue")


# Adding Labels, title, and legend

plt.title("Description - Net Revenue", fontsize=14)

plt.xlabel("Net Revenue", fontsize=12)
```

```
plt.ylabel("Description", fontsize=12)
plt.grid(axis="x", linestyle="--", alpha=0.5)

# Labels on bars
for i, v in enumerate(description_pd["Net_Revenue"]):
    plt.text(v, i, f"{v:,.0f}", va="center", ha="left", fontsize=9, color="black")

plt.tight_layout()
plt.legend()

# Show Summary in Table
print(description_summary.head(10))

# Show Chart
plt.show()
```

```
plt.show()
```

shape: (10, 4)

Description	Gross_Revenue	Net_Revenue	Returns
str	f64	f64	f64
Manual	-82847.84	340303.87	-423151.71
REGENCY CAKESTAND 3 TIER	322643.15	339301.55	-16658.4
DOTCOM POSTAGE	304979.93	304989.94	-10.01
WHITE HANGING HEART T-LIGHT HO…	255777.96	265164.66	-9386.7
JUMBO BAG RED RETROSPOT	147803.47	149909.27	-2105.8
PARTY BUNTING	147361.56	148600.11	-1238.55
ASSORTED COLOUR BIRD ORNAMENT	130242.39	131016.46	-774.07
POSTAGE	110219.21	125471.13	-15251.92
PAPER CHAIN KIT 50'S CHRISTMAS	116439.35	117918.75	-1479.4
CHILLI LIGHTS	83145.65	83775.65	-630.0



Customer Summary

customer_summary = (

```
    df.group_by("Customer ID")
      .agg([
         pl.sum("Revenue").alias("Gross_Revenue"),
         pl.sum("Net_Revenue"),
         pl.sum("Returns")
      ])
      .sort("Net_Revenue", descending=True)
)

customer_summary = customer_summary.drop_nulls()
```

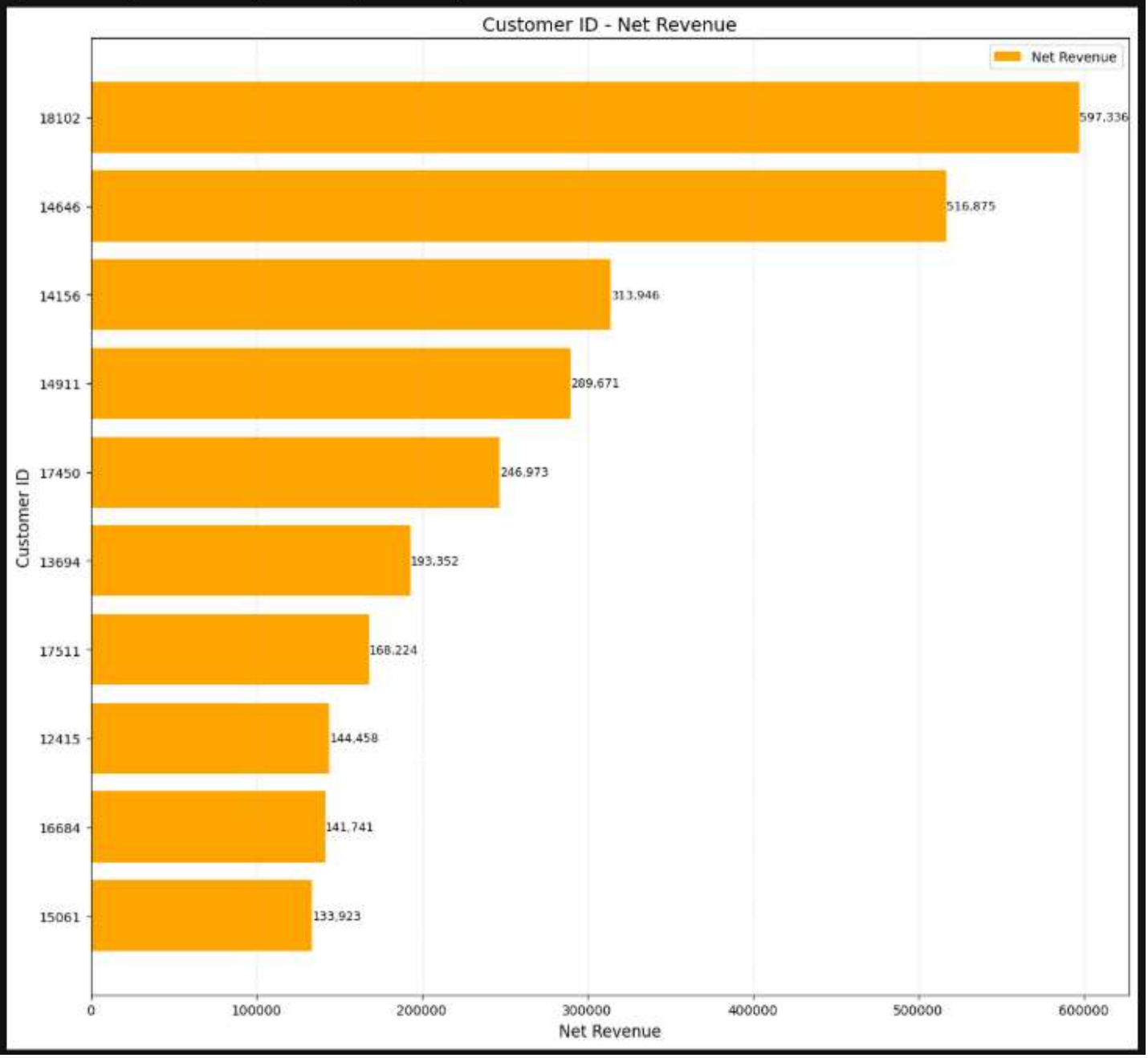
```
print(customer_summary.head(10))
shape: (10, 4)
┌─────────────┬───────────────┬─────────────┬─────────┐
│ Customer ID ┆ Gross_Revenue ┆ Net_Revenue ┆ Returns │
│ ---         ┆ ---           ┆ ---         ┆ ---     │
│ str         ┆ f64           ┆ f64         ┆ f64     │
╞═════════════╪═══════════════╪═════════════╪═════════╡
│ 18102       ┆ 586729.68     ┆ 597336.11   ┆ -10606.43 │
│ 14646       ┆ 511614.05     ┆ 516874.5    ┆ -5260.45  │
│ 14156       ┆ 296564.69     ┆ 313946.37   ┆ -17381.68 │
│ 14911       ┆ 263946.56     ┆ 289670.66   ┆ -25724.1  │
│ 17450       ┆ 233579.39     ┆ 246973.09   ┆ -13393.7  │
│ 13694       ┆ 187694.36     ┆ 193351.65   ┆ -5657.29  │
│ 17511       ┆ 164506.66     ┆ 168224.23   ┆ -3717.57  │
│ 12415       ┆ 143269.29     ┆ 144458.37   ┆ -1189.08  │
│ 16684       ┆ 136100.27     ┆ 141740.79   ┆ -5640.52  │
│ 15061       ┆ 132495.62     ┆ 133922.66   ┆ -1427.04  │
└─────────────┴───────────────┴─────────────┴─────────┘
```

```
# Horizontal Bar Chart - Comparing Customer Categories

# Convert Polars DataFrame to pandas manually
customer_pd = pd.DataFrame(customer_summary.head(10).to_dicts())

# Sort by Net_Revenue descending (highest first)
customer_pd = customer_pd.sort_values(by="Net_Revenue", ascending=True)

plt.figure(figsize=(12, 11))  # Reasonable figure size
plt.barh(customer_pd["Customer ID"], customer_pd["Net_Revenue"], color='orange', label="Net Revenue")

# Adding Labels, title, and legend
plt.title("Customer ID - Net Revenue", fontsize=14)
```

```
plt.xlabel("Net Revenue", fontsize=12)
plt.ylabel("Customer ID", fontsize=12)
plt.grid(axis="x", linestyle="--", alpha=0.5)

# Labels on bars
for i, v in enumerate(customer_pd["Net_Revenue"]):
    plt.text(v, i, f"{v:,.0f}", va="center", ha="left", fontsize=9, color="black")

plt.tight_layout()
plt.legend()

# Show Summary in Table
print(customer_summary.head(10))

# Show Chart
plt.show()
```

shape: (10, 4)

Customer ID	Gross_Revenue	Net_Revenue	Returns
str	f64	f64	f64
18102	586729.68	597336.11	-10606.43
14646	511614.05	516874.5	-5260.45
14156	296564.69	313946.37	-17381.68
14911	263946.56	289670.66	-25724.1
17450	233579.39	246973.09	-13393.7
13694	187694.36	193351.65	-5657.29
17511	164506.66	168224.23	-3717.57
12415	143269.29	144458.37	-1189.08
16684	136100.27	141740.79	-5640.52
15061	132495.62	133922.66	-1427.04



Daily Summary

daily_summary = (

df.group_by("Invoice_Date")

```
    .agg([
        pl.sum("Revenue").round(2).alias("Gross_Revenue"),
        pl.sum("Net_Revenue").round(2),
        pl.sum("Returns").round(2)
    ])
    .with_columns([
        pl.col("Invoice_Date")
          .dt.strftime("%m-%d-%Y")             # format as MM-dd-yyyy
          .str.strptime(pl.Date, "%m-%d-%Y", strict=False)  # convert back to Date type
          .alias("Invoice_Date")
    ])
    .sort("Invoice_Date", descending=False)
)

daily_summary = daily_summary.drop_nulls()
```

```
print(daily_summary.head(10))
shape: (10, 4)
┌──────────────┬───────────────┬─────────────┬─────────┐
│ Invoice_Date ┆ Gross_Revenue ┆ Net_Revenue ┆ Returns │
│ ---          ┆ ---           ┆ ---         ┆ ---     │
│ date         ┆ f64           ┆ f64         ┆ f64     │
╞══════════════╪═══════════════╪═════════════╪═════════╡
│ 2009-01-12   ┆ 53173.03      ┆ 54513.5     ┆ -1340.47 │
│ 2009-02-12   ┆ 62763.59      ┆ 63352.51    ┆ -588.92  │
│ 2009-03-12   ┆ 68093.05      ┆ 74037.91    ┆ -5944.86 │
│ 2009-04-12   ┆ 40346.4       ┆ 40732.92    ┆ -386.52  │
│ 2009-05-12   ┆ 9803.05       ┆ 9803.05     ┆ 0.0      │
│ 2009-06-12   ┆ 24317.2       ┆ 24613.64    ┆ -296.44  │
│ 2009-07-12   ┆ 44337.02      ┆ 45083.35    ┆ -746.33  │
│ 2009-08-12   ┆ 43743.14      ┆ 49517.23    ┆ -5774.09 │
│ 2009-09-12   ┆ 40396.55      ┆ 40616.09    ┆ -219.54  │
│ 2009-10-12   ┆ 43342.48      ┆ 44442.11    ┆ -1099.63 │
└──────────────┴───────────────┴─────────────┴─────────┘
```

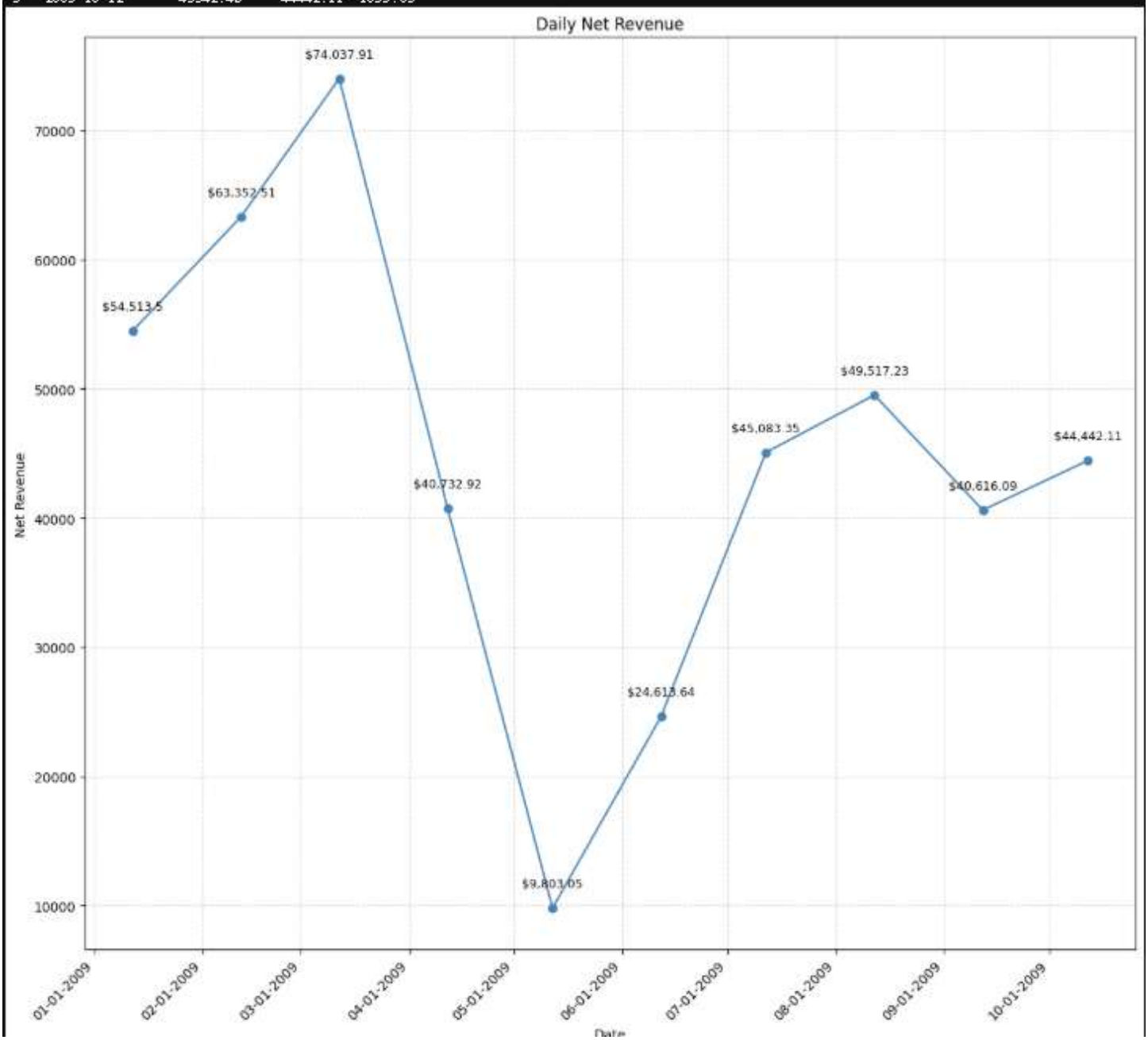
```
# Line Graph for Daily Net Revenue

import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import DateFormatter

# Convert Polars DataFrame to pandas manually
daily_pd = pd.DataFrame(daily_summary.head(10).to_dicts())
```

```
# Parse 'Invoice_Date' as datetime
daily_pd["Invoice_Date"] = pd.to_datetime(daily_pd["Invoice_Date"], format="%m-%d-%Y", errors="coerce")

# Drop nulls if any failed to parse
daily_pd = daily_pd.dropna(subset=["Invoice_Date"])

# Sort by actual datetime
daily_pd = daily_pd.sort_values(by="Invoice_Date", ascending=True)

# --- Plot ---
fig, ax = plt.subplots(figsize=(12,11))
ax.plot(daily_pd["Invoice_Date"], daily_pd["Net_Revenue"], marker='o', color='steelblue')

# Add labels on each point
ymin, ymax = ax.get_ylim()
yoff = 0.02 * (ymax-ymin)

for x,y in zip(daily_pd["Invoice_Date"], daily_pd["Net_Revenue"]):
    ax.text(x,y + yoff, f"${y:,}", ha="center", va="bottom", fontsize=9)

# Title and labels
ax.set_title("Daily Net Revenue")
ax.set_xlabel("Date")
ax.set_ylabel("Net Revenue")

# Format x-axis date as MM-dd-YYYY
date_format = DateFormatter("%m-%d-%Y")
ax.xaxis.set_major_formatter(date_format)
fig.autofmt_xdate(rotation=45)  # rotate labels for readability
```

```
# Grid and layout
ax.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()

# Show Summary in table
print(daily_summary.head(10)) # polars dataFrame
print(daily_pd.head(10))      # pandas dataFrame

plt.show()
```

2009-03-12	68093.05	74037.91	-5944.86
2009-04-12	40346.4	40732.92	-386.52
2009-05-12	9803.05	9803.05	0.0
2009-06-12	24317.2	24613.64	-296.44
2009-07-12	44337.02	45083.35	-746.33
2009-08-12	43743.14	49517.23	-5774.09
2009-09-12	40396.55	40616.09	-219.54
2009-10-12	43342.48	44442.11	-1099.63

	Invoice_Date	Gross_Revenue	Net_Revenue	Returns
0	2009-01-12	53173.03	54513.50	-1340.47
1	2009-02-12	62763.59	63352.51	-588.92
2	2009-03-12	68093.05	74037.91	-5944.86
3	2009-04-12	40346.40	40732.92	-386.52
4	2009-05-12	9803.05	9803.05	0.00
5	2009-06-12	24317.20	24613.64	-296.44
6	2009-07-12	44337.02	45083.35	-746.33
7	2009-08-12	43743.14	49517.23	-5774.09
8	2009-09-12	40396.55	40616.09	-219.54
9	2009-10-12	43342.48	44442.11	-1099.63



Weekly Summary

```
# Weekly Summary
weekly_summary = (
   df.group_by("Week")
     .agg([
        pl.sum("Revenue").round(2).alias("Gross_Revenue"),
        pl.sum("Net_Revenue").round(2),
        pl.sum("Returns").round(2)
     ])
     .with_columns([
        pl.col("Week")
          .dt.strftime("%m-%d-%Y")             # format as MM-dd-yyyy
          .str.strptime(pl.Date, "%m-%d-%Y", strict=False)  # convert back to Date type
          .alias("Invoice_Date")
     ])
     .sort("Week", descending=False)
)
weekly_summary = weekly_summary.drop_nulls()
```

```
print(weekly_summary.head(10))
shape: (10, 5)
┌────────────┬───────────────┬─────────────┬─────────┬──────────────┐
│ Week       ┆ Gross_Revenue ┆ Net_Revenue ┆ Returns ┆ Invoice_Date │
│ ---        ┆ ---           ┆ ---         ┆ ---     ┆ ---          │
│ date       ┆ f64           ┆ f64         ┆ f64     ┆ date         │
╞════════════╪═══════════════╪═════════════╪═════════╪══════════════╡
│ 2009-01-12 ┆ 53173.03      ┆ 54513.5     ┆ -1340.47┆ 2009-01-12   │
│ 2009-02-09 ┆ 62763.59      ┆ 63352.51    ┆ -588.92 ┆ 2009-02-09   │
│ 2009-03-09 ┆ 68093.05      ┆ 74037.91    ┆ -5944.86┆ 2009-03-09   │
│ 2009-04-06 ┆ 40346.4       ┆ 40732.92    ┆ -386.52 ┆ 2009-04-06   │
│ 2009-05-11 ┆ 9803.05       ┆ 9803.05     ┆ 0.0     ┆ 2009-05-11   │
│ 2009-06-08 ┆ 24317.2       ┆ 24613.64    ┆ -296.44 ┆ 2009-06-08   │
│ 2009-07-06 ┆ 44337.02      ┆ 45083.35    ┆ -746.33 ┆ 2009-07-06   │
│ 2009-08-10 ┆ 43743.14      ┆ 49517.23    ┆ -5774.09┆ 2009-08-10   │
│ 2009-09-07 ┆ 40396.55      ┆ 40616.09    ┆ -219.54 ┆ 2009-09-07   │
│ 2009-10-12 ┆ 43342.48      ┆ 44442.11    ┆ -1099.63┆ 2009-10-12   │
└────────────┴───────────────┴─────────────┴─────────┴──────────────┘
```

Line Graph for Weekly Net Revenue

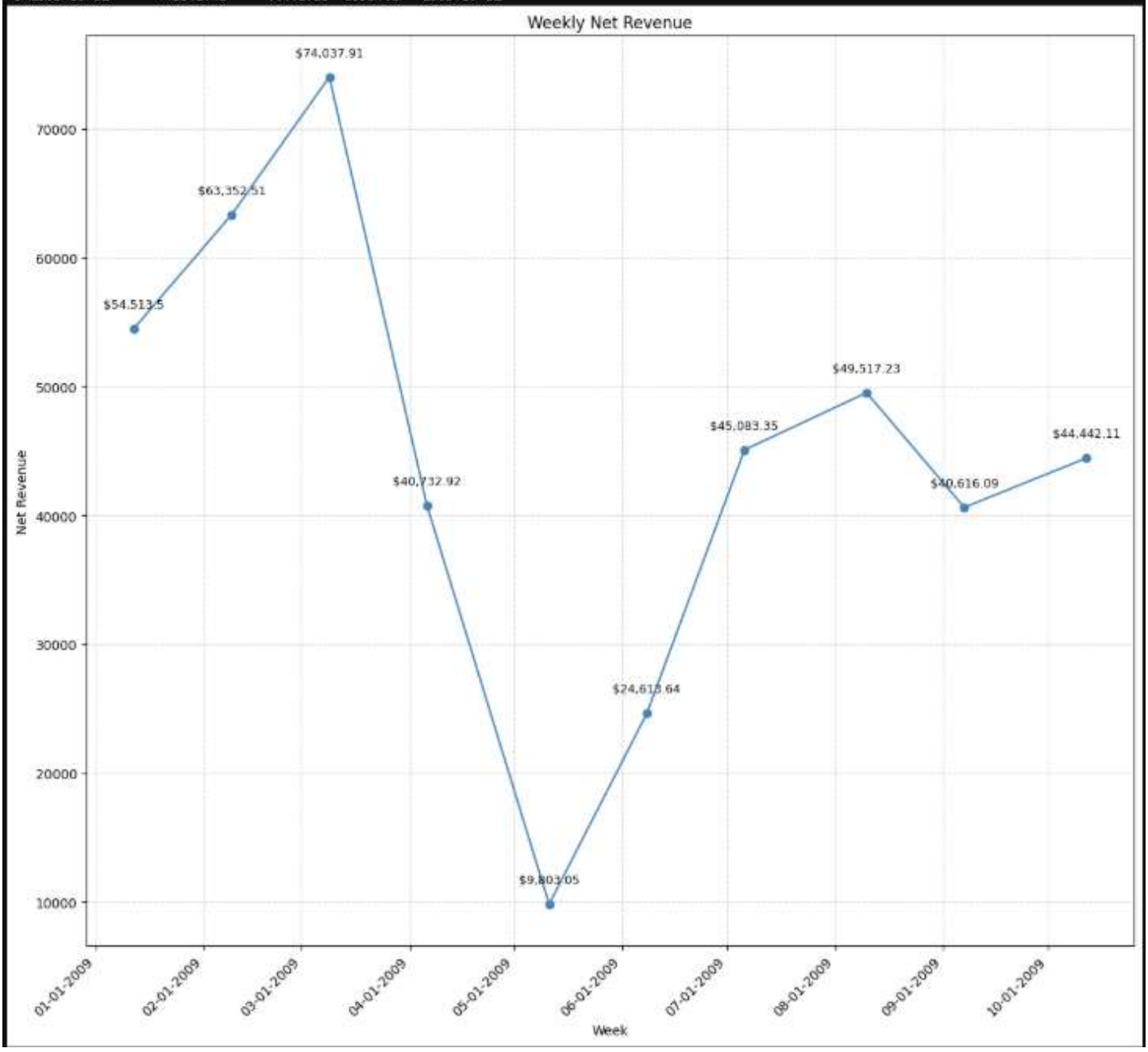
```
import matplotlib.pyplot as plt
import pandas as pd
```

```
from matplotlib.dates import DateFormatter

# Convert Polars DataFrame to pandas manually
weekly_pd = pd.DataFrame(weekly_summary.head(10).to_dicts())

# Parse 'Invoice_Date' as datetime
weekly_pd["Week"] = pd.to_datetime(weekly_pd["Week"], format="%m-%d-%Y", errors="coerce")

# Drop nulls if any failed to parse
weekly_pd = weekly_pd.dropna(subset=["Week"])

# Sort by actual datetime
weekly_pd = weekly_pd.sort_values(by="Week", ascending=True)

# --- Plot ---
fig, ax = plt.subplots(figsize=(12,11))
ax.plot(weekly_pd["Week"], weekly_pd["Net_Revenue"], marker='o', color='steelblue')

# Add labels on each point
ymin, ymax = ax.get_ylim()
yoff = 0.02 * (ymax-ymin)

for x,y in zip(weekly_pd["Week"], weekly_pd["Net_Revenue"]):
    ax.text(x,y + yoff, f"${y:,}", ha="center", va="bottom", fontsize=9)

# Title and labels
ax.set_title("Weekly Net Revenue")
ax.set_xlabel("Week")
ax.set_ylabel("Net Revenue")

# Format x-axis date as MM-dd-YYYY
```

```
date_format = DateFormatter("%m-%d-%Y")
ax.xaxis.set_major_formatter(date_format)
fig.autofmt_xdate(rotation=45)  # rotate labels for readability

# Grid and layout
ax.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()

# Show Summary in table
print(weekly_summary.head(10)) # polars dataFrame
print(weekly_pd.head(10))      # pandas dataFrame

plt.show()
```

```
| 2009-07-06 | 44337.02 | 45083.35 | -746.33 | 2009-07-06 |
| 2009-08-10 | 43743.14 | 49517.23 | -5774.09 | 2009-08-10 |
| 2009-09-07 | 40396.55 | 40616.09 | -219.54 | 2009-09-07 |
| 2009-10-12 | 43342.48 | 44442.11 | -1099.63 | 2009-10-12 |
```

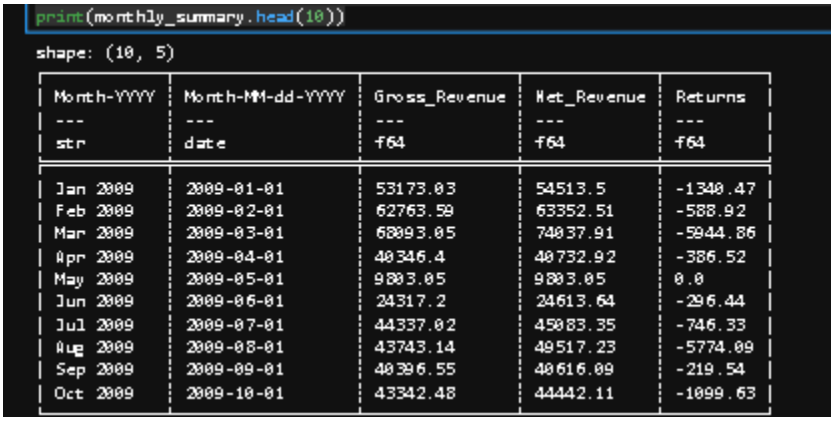
	Week	Gross_Revenue	Net_Revenue	Returns	Invoice_Date
0	2009-01-12	53173.03	54513.50	-1340.47	2009-01-12
1	2009-02-09	62763.59	63352.51	-588.92	2009-02-09
2	2009-03-09	68093.05	74037.91	-5944.86	2009-03-09
3	2009-04-06	40346.40	40732.92	-386.52	2009-04-06
4	2009-05-11	9803.05	9803.05	0.00	2009-05-11
5	2009-06-08	24317.20	24613.64	-296.44	2009-06-08
6	2009-07-06	44337.02	45083.35	-746.33	2009-07-06
7	2009-08-10	43743.14	49517.23	-5774.09	2009-08-10
8	2009-09-07	40396.55	40616.09	-219.54	2009-09-07
9	2009-10-12	43342.48	44442.11	-1099.63	2009-10-12



Monthly Summary

monthly_summary = (

```
    df.group_by(["Month-YYYY", "Month-MM-dd-YYYY"])
      .agg([
         pl.sum("Revenue").alias("Gross_Revenue"),
         pl.sum("Net_Revenue").round(2),
         pl.sum("Returns").round(2)
      ])
      .sort("Month-MM-dd-YYYY", descending=False) # sort by actual date
)

# Drop nulls for safety
monthly_summary = monthly_summary.drop_nulls()
```



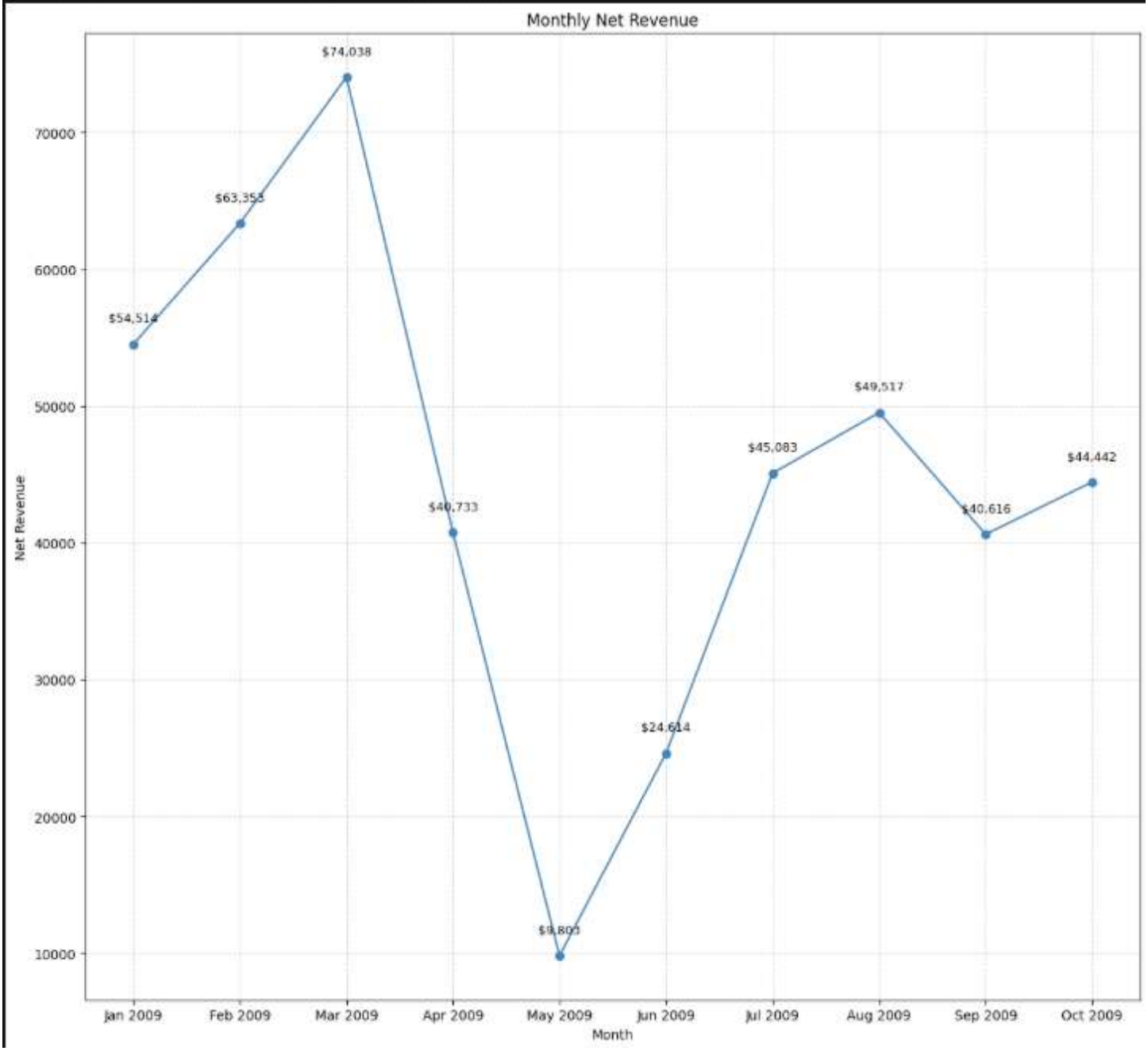
```
print(monthly_summary.head(10))
shape: (10, 5)
┌──────────────┬──────────────────┬───────────────┬─────────────┬─────────┐
│ Month-YYYY   ┆ Month-MM-dd-YYYY ┆ Gross_Revenue ┆ Net_Revenue ┆ Returns │
│ ---          ┆ ---              ┆ ---           ┆ ---         ┆ ---     │
│ str          ┆ date             ┆ f64           ┆ f64         ┆ f64     │
╞══════════════╪══════════════════╪═══════════════╪═════════════╪═════════╡
│ Jan 2009     ┆ 2009-01-01       ┆ 53173.03      ┆ 54513.5     ┆ -1340.47 │
│ Feb 2009     ┆ 2009-02-01       ┆ 62763.59      ┆ 63352.51    ┆ -588.92  │
│ Mar 2009     ┆ 2009-03-01       ┆ 68093.05      ┆ 74037.91    ┆ -5944.86 │
│ Apr 2009     ┆ 2009-04-01       ┆ 40346.4       ┆ 40732.92    ┆ -386.52  │
│ May 2009     ┆ 2009-05-01       ┆ 9803.05       ┆ 9803.05     ┆ 0.0      │
│ Jun 2009     ┆ 2009-06-01       ┆ 24317.2       ┆ 24613.64    ┆ -296.44  │
│ Jul 2009     ┆ 2009-07-01       ┆ 44337.02      ┆ 45083.35    ┆ -746.33  │
│ Aug 2009     ┆ 2009-08-01       ┆ 43743.14      ┆ 49517.23    ┆ -5774.09 │
│ Sep 2009     ┆ 2009-09-01       ┆ 40396.55      ┆ 40616.09    ┆ -219.54  │
│ Oct 2009     ┆ 2009-10-01       ┆ 43342.48      ┆ 44442.11    ┆ -1099.63 │
└──────────────┴──────────────────┴───────────────┴─────────────┴─────────┘
```

```
# Line Graph for Monthly Net Revenue

import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import DateFormatter

# Convert Polars DataFrame to pandas manually
monthly_pd = pd.DataFrame(monthly_summary.head(10).to_dicts())

# Ensure Month-MM-dd-YYYY is a datetime for sorting and plotting
monthly_pd["Month-MM-dd-YYYY"] = pd.to_datetime(monthly_pd["Month-MM-dd-YYYY"], errors="coerce")
```

```
# Sort chronologically (ascending)
monthly_pd = monthly_pd.sort_values(by="Month-MM-dd-YYYY", ascending=True)

# --- Line Graph for Monthly Net Revenue ---
fig, ax = plt.subplots(figsize=(12,11))
ax.plot(monthly_pd["Month-YYYY"], monthly_pd["Net_Revenue"], marker='o', color='steelblue')

# Add labels above points
ymin, ymax = ax.get_ylim()
yoff = 0.02 * (ymax-ymin)
for x, y in zip(monthly_pd["Month-YYYY"], monthly_pd["Net_Revenue"]):
    ax.text(x, y + yoff, f"${y:,.0f}", ha="center", va="bottom", fontsize=9)

# Title, labels, grid
ax.set_title("Monthly Net Revenue")
ax.set_xlabel("Month")
ax.set_ylabel("Net Revenue")
ax.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()

print(monthly_summary.head(10))
print(monthly_pd.head(10))

plt.show()
```

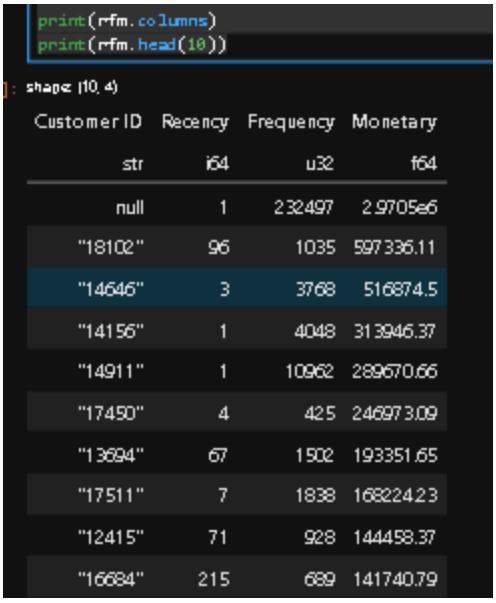
May 2009	2009-05-01	9803.05	9803.05	0.0
Jun 2009	2009-06-01	24317.2	24613.64	-296.44
Jul 2009	2009-07-01	44337.02	45083.35	-746.33
Aug 2009	2009-08-01	43743.14	49517.23	-5774.09
Sep 2009	2009-09-01	40396.55	40616.09	-219.54
Oct 2009	2009-10-01	43342.48	44442.11	-1099.63

	Month-YYYY	Month-MM-dd-YYYY	Gross_Revenue	Net_Revenue	Returns
0	Jan 2009	2009-01-01	53173.03	54513.50	-1340.47
1	Feb 2009	2009-02-01	62763.59	63352.51	-588.92
2	Mar 2009	2009-03-01	68093.05	74037.91	-5944.86
3	Apr 2009	2009-04-01	40346.40	40732.92	-386.52
4	May 2009	2009-05-01	9803.05	9803.05	0.00
5	Jun 2009	2009-06-01	24317.20	24613.64	-296.44
6	Jul 2009	2009-07-01	44337.02	45083.35	-746.33
7	Aug 2009	2009-08-01	43743.14	49517.23	-5774.09
8	Sep 2009	2009-09-01	40396.55	40616.09	-219.54
9	Oct 2009	2009-10-01	43342.48	44442.11	-1099.63



Filter only valid (non-return) transactions

```
df_valid = df.filter(~pl.col("IsReturn"))

# Building Recency-Frequency-Monetary segments
# Define reference date (max invoice date + 1 day)
max_date = df_valid["Invoice_Date"].max()
reference_date = max_date + pl.duration(days=1)

# Compute RFM metrics
rfm = (
    df_valid.group_by("Customer ID")
    .agg([
        # Recency: days since last purchase
        (reference_date - pl.col("Invoice_Date").max())
            .dt.total_days()
            .alias("Recency"),

        # Frequency: number of invoices
        pl.col("InvoiceDate").count().alias("Frequency"),

        # Monetary: total revenue
        pl.col("Revenue").sum().alias("Monetary")
    ])
    .sort("Monetary", descending=True)
)

print(rfm.columns)
print(rfm.head(10))
```



```
print(rfm.columns)
print(rfm.head(10))
```

shape: (10, 4)

Customer ID	Recency	Frequency	Monetary
str	i64	u32	f64
null	1	232497	2.9705e6
"18102"	96	1035	597336.11
"14646"	3	3768	516874.5
"14156"	1	4048	313946.37
"14911"	1	10962	289670.66
"17450"	4	425	246973.09
"13694"	67	1502	193351.65
"17511"	7	1838	168224.23
"12415"	71	928	144458.37
"16684"	215	689	141740.79

```
# Compute quantile thresholds
q = rfm.select([
    pl.col("Recency").quantile(0.2).alias("R_20"),
    pl.col("Recency").quantile(0.4).alias("R_40"),
    pl.col("Recency").quantile(0.6).alias("R_60"),
    pl.col("Recency").quantile(0.8).alias("R_80"),
    pl.col("Frequency").quantile(0.2).alias("F_20"),
    pl.col("Frequency").quantile(0.4).alias("F_40"),
    pl.col("Frequency").quantile(0.6).alias("F_60"),
    pl.col("Frequency").quantile(0.8).alias("F_80"),
    pl.col("Monetary").quantile(0.2).alias("M_20"),
    pl.col("Monetary").quantile(0.4).alias("M_40"),
    pl.col("Monetary").quantile(0.6).alias("M_60"),
    pl.col("Monetary").quantile(0.8).alias("M_80"),
]).to_dict(as_series=False)
rfm = rfm.with_columns([
    # Recency (lower is better)
    pl.when(pl.col("Recency") <= q["R_20"][0]).then(5)
     .when(pl.col("Recency") <= q["R_40"][0]).then(4)
     .when(pl.col("Recency") <= q["R_60"][0]).then(3)
```

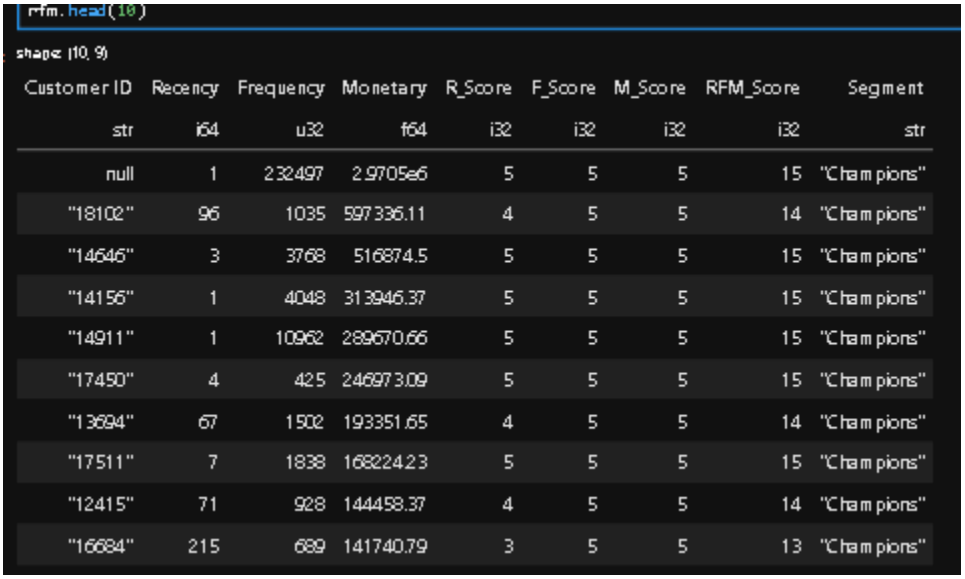
```
    .when(pl.col("Recency") <= q["R_80"][0]).then(2)
    .otherwise(1).alias("R_Score"),

   # Frequency (higher is better)
   pl.when(pl.col("Frequency") <= q["F_20"][0]).then(1)
    .when(pl.col("Frequency") <= q["F_40"][0]).then(2)
    .when(pl.col("Frequency") <= q["F_60"][0]).then(3)
    .when(pl.col("Frequency") <= q["F_80"][0]).then(4)
    .otherwise(5).alias("F_Score"),

   # Monetary (higher is better)
   pl.when(pl.col("Monetary") <= q["M_20"][0]).then(1)
    .when(pl.col("Monetary") <= q["M_40"][0]).then(2)
    .when(pl.col("Monetary") <= q["M_60"][0]).then(3)
    .when(pl.col("Monetary") <= q["M_80"][0]).then(4)
    .otherwise(5).alias("M_Score"),
])

print(rfm.columns)
print(rfm.head(10))
```

```
print(rfm.columns)
print(rfm.head(10))
```

shape: (10, 7)

Customer ID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score
str	i64	u32	f64	i32	i32	i32
null	1	232497	2.9705e6	5	5	5
"18102"	96	1035	597336.11	4	5	5
"14646"	3	3768	516874.5	5	5	5
"14156"	1	4048	313946.37	5	5	5
"14911"	1	10962	289670.66	5	5	5
"17450"	4	425	246973.09	5	5	5
"13694"	67	1502	193351.65	4	5	5
"17511"	7	1838	168224.23	5	5	5
"12415"	71	928	144458.37	4	5	5
"16684"	215	689	141740.79	3	5	5

```
# Define customer segments
rfm = rfm.with_columns(
    pl.when(pl.col("RFM_Score") >= 13).then(pl.lit("Champions"))
     .when(pl.col("RFM_Score") >= 10).then(pl.lit("Loyal Customers"))
     .when(pl.col("RFM_Score") >= 7).then(pl.lit("Potential Loyalist"))
     .when(pl.col("RFM_Score") >= 4).then(pl.lit("At Risk"))
     .otherwise(pl.lit("Lost"))
     .alias("Segment")
)
```



rfm.head(10)

shape: (10, 9)

Customer ID	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	Segment
str	i64	u32	f64	i32	i32	i32	i32	str
null	1	232497	2.9705e6	5	5	5	15	"Champions"
"18102"	96	1035	597336.11	4	5	5	14	"Champions"
"14646"	3	3768	516874.5	5	5	5	15	"Champions"
"14156"	1	4048	313946.37	5	5	5	15	"Champions"
"14911"	1	10962	289670.66	5	5	5	15	"Champions"
"17450"	4	425	246973.09	5	5	5	15	"Champions"
"13694"	67	1502	193351.65	4	5	5	14	"Champions"
"17511"	7	1838	168224.23	5	5	5	15	"Champions"
"12415"	71	928	144458.37	4	5	5	14	"Champions"
"16684"	215	689	141740.79	3	5	5	13	"Champions"

```
# sort RFM via customers
rfm = (
   rfm.group_by("Segment")
      .agg([
         pl.count().alias("Customers"),
         pl.col("Monetary").mean().alias("Avg_Monetary")
      ])
      .sort("Customers", descending=True)
)
```

shape: (5, 3)

Segment	Customers	Avg_Monetary
str	u32	f64
"Potential Loyalist"	1546	1142.358596
"At Risk"	1478	390.735886
"Loyal Customers"	1152	2764.034019
"Champions"	1105	13286.677285
"Lost"	583	147.333533

Bar Chart - Comparing RFM via Customers

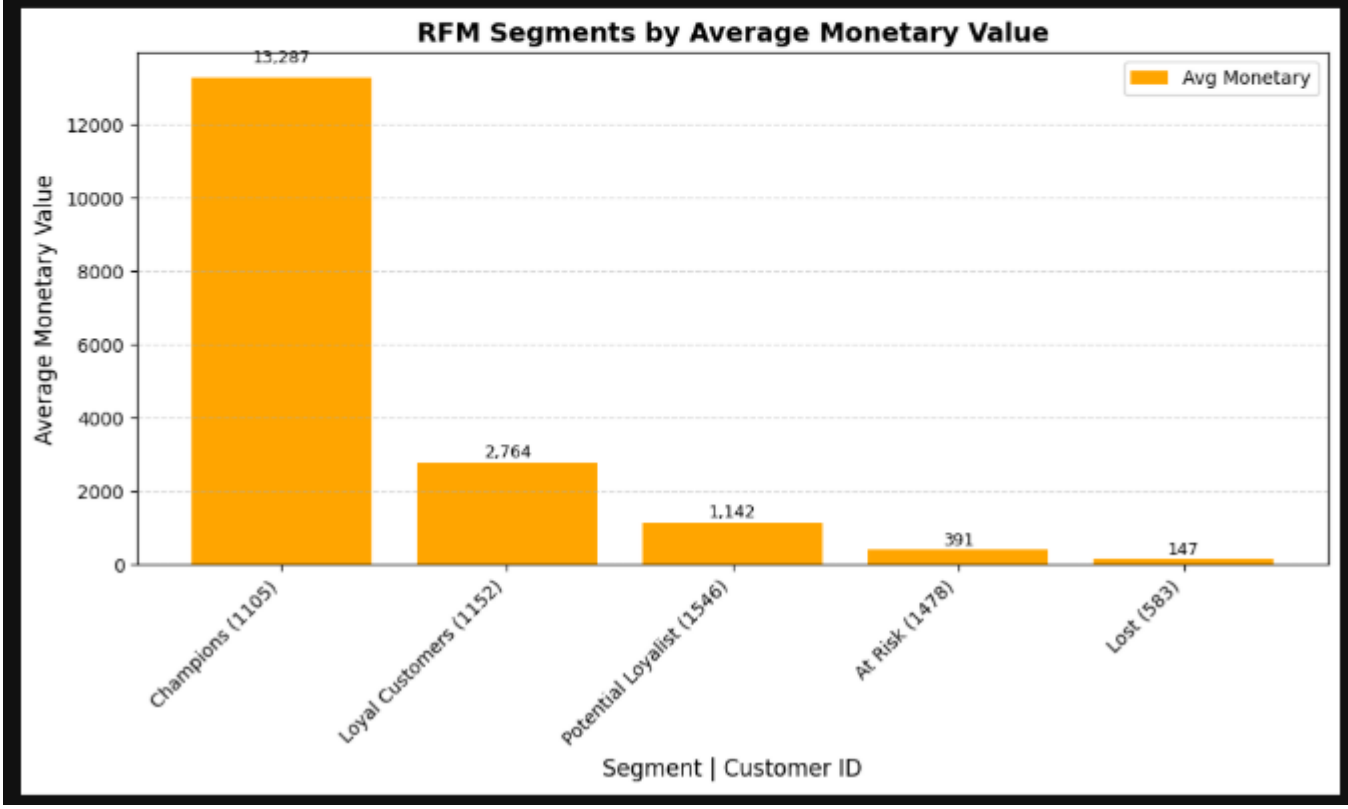
```
import polars as pl
import pandas as pd
import matplotlib.pyplot as plt

# Convert Polars DataFrame to pandas manually
rfm_pd = pd.DataFrame(rfm.head(10).to_dicts())

# Create combined label for X-axis
rfm_pd["Segment_Customers"] = rfm_pd["Segment"] + " (" + rfm_pd["Customers"].astype(str) + ")"

# Sort by Avg_Monetary descending for better visualization
rfm_pd = rfm_pd.sort_values(by="Avg_Monetary", ascending=False)

# Plot Vertical Bar Chart
plt.figure(figsize=(10, 6))
bars = plt.bar(rfm_pd["Segment_Customers"], rfm_pd["Avg_Monetary"], color='orange', label="Avg Monetary")

# Add Titles and Labels
plt.title("RFM Segments by Average Monetary Value", fontsize=14, weight='bold')
plt.xlabel("Segment | Customer ID", fontsize=12)
plt.ylabel("Average Monetary Value", fontsize=12)
plt.grid(axis="y", linestyle="--", alpha=0.5)
```

```
# Add value labels above each bar
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + (height*0.02),
             f"{height:,.0f}", ha="center", va="bottom", fontsize=9, color="black")

# Rotate x labels for readability
plt.xticks(rotation=45, ha="right")

plt.tight_layout()
plt.legend()

# Show Summary in Table
print(rfm_pd.head(10)) # Pandas DataFrame
print(rfm.head(10))    # Polars DataFrame

# Show Chart
plt.show()
```

```
plt.show()
```

```
             Segment  Customers  Avg_Monetary          Segment_Customers
3          Champions       1105  13286.677285          Champions (1105)
2     Loyal Customers       1152   2764.034019     Loyal Customers (1152)
0  Potential Loyalist       1546   1142.358596  Potential Loyalist (1546)
1             At Risk       1478    390.735886             At Risk (1478)
4                Lost        583    147.333533                Lost (583)
shape: (5, 3)
```

Segment	Customers	Avg_Monetary
str	u32	f64
Potential Loyalist	1546	1142.358596
At Risk	1478	390.735886
Loyal Customers	1152	2764.034019
Champions	1105	13286.677285
Lost	583	147.333533



detects large-order outliers (IQR method and top-percentile method) and measures their share of units/revenue

import polars as pl

import math

Example defaults for inventory calc

lead_time_weeks = 2.0 # lead time in weeks (tunable)

service_z = 1.645 # z-value for ~95% service level (tunable)

forecast_weeks = 4 # used only if you implement rolling mean later (we use avg_weekly baseline)

outlier_pct = 0.99 # top percentile threshold for outlier detection (alternative method)

iqr_multiplier = 1.5 # IQR multiplier for classical outlier detection

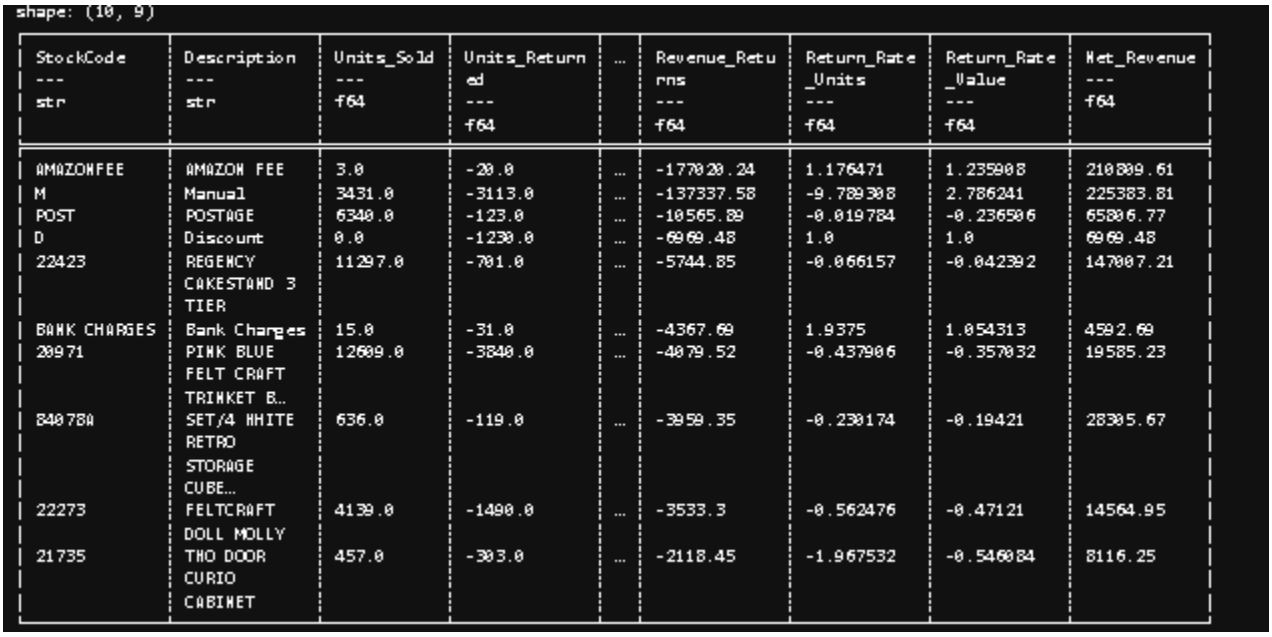
```
# Ensure Invoice_Date exists (date)
if "Invoice_Date" not in df.columns and "Invoice_DateTime" in df.columns:
    df = df.with_columns(pl.col("Invoice_DateTime").dt.date().alias("Invoice_Date"))


# Filter only non-null Invoice_Date for time aggregations
df = df.filter(pl.col("Invoice_Date").is_not_null())


# Convert Quantity to numeric (use absolute for units sold accounting)
df = df.with_columns(pl.col("Quantity").cast(pl.Float64).alias("Quantity"))


print(df.head(10))
```

```
shape: (10, 19)
┌─────────┬───────────┬──────────────────────────────┬──────────┬─────┬─────────┬──────────┬────────────┬─────────┐
│ Invoice ┆ StockCode ┆ Description                  ┆ Quantity ┆ …   ┆ Revenue ┆ IsReturn ┆ Net_Revenue ┆ Returns │
│ ---     ┆ ---       ┆ ---                          ┆ ---      ┆     ┆ ---     ┆ ---      ┆ ---        ┆ ---     │
│ str     ┆ str       ┆ str                          ┆ f64      ┆     ┆ f64     ┆ bool     ┆ f64        ┆ f64     │
╞═════════╪═══════════╪══════════════════════════════╪══════════╪═════╪═════════╪══════════╪════════════╪═════════╡
│ 489434  ┆ 85048     ┆ 15CM CHRISTMAS GLASS BALL 20 L… ┆ 12.0     ┆ …   ┆ 83.4    ┆ false    ┆ 83.4       ┆ 0.0     │
│ 489434  ┆ 79323P    ┆ PINK CHERRY LIGHTS           ┆ 12.0     ┆ …   ┆ 81.0    ┆ false    ┆ 81.0       ┆ 0.0     │
│ 489434  ┆ 79323H    ┆ WHITE CHERRY LIGHTS          ┆ 12.0     ┆ …   ┆ 81.0    ┆ false    ┆ 81.0       ┆ 0.0     │
│ 489434  ┆ 22041     ┆ RECORD FRAME 7" SINGLE SIZE  ┆ 48.0     ┆ …   ┆ 100.8   ┆ false    ┆ 100.8      ┆ 0.0     │
│ 489434  ┆ 21232     ┆ STRAWBERRY CERAMIC TRINKET BOX ┆ 24.0     ┆ …   ┆ 30.0    ┆ false    ┆ 30.0       ┆ 0.0     │
│ 489434  ┆ 22064     ┆ PINK DOUGHNUT TRINKET POT    ┆ 24.0     ┆ …   ┆ 39.6    ┆ false    ┆ 39.6       ┆ 0.0     │
│ 489434  ┆ 21871     ┆ SAVE THE PLANET MUG          ┆ 24.0     ┆ …   ┆ 30.0    ┆ false    ┆ 30.0       ┆ 0.0     │
│ 489434  ┆ 21523     ┆ FANCY FONT HOME SWEET HOME DOO… ┆ 10.0     ┆ …   ┆ 59.5    ┆ false    ┆ 59.5       ┆ 0.0     │
│ 489435  ┆ 22350     ┆ CAT BOWL                     ┆ 12.0     ┆ …   ┆ 30.6    ┆ false    ┆ 30.6       ┆ 0.0     │
│ 489435  ┆ 22349     ┆ DOG BOWL , CHASING BALL DESIGN ┆ 12.0     ┆ …   ┆ 45.0    ┆ false    ┆ 45.0       ┆ 0.0     │
└─────────┴───────────┴──────────────────────────────┴──────────┴─────┴─────────┴──────────┴────────────┴─────────┘
```

```
# Quantify returns impact


# Per product (StockCode) compute sold units, returned units, gross revenue, revenue lost to returns
product_returns = (
    df.group_by("StockCode")
      .agg([
        # Get the first description for each product
        pl.col("Description").first().alias("Description"),

        # --- Compute Sold & Returned Units ---
        pl.when(pl.col("IsReturn") == False)
          .then(pl.col("Quantity"))
```

```
        .otherwise(0)
        .sum()
        .alias("Units_Sold"),

    pl.when(pl.col("IsReturn") == True)
        .then(pl.col("Quantity"))
        .otherwise(0)
        .sum()
        .alias("Units_Returned"),

    # --- Compute Revenues ---
    pl.when(pl.col("IsReturn") == False)
        .then(pl.col("Revenue"))
        .otherwise(0)
        .sum()
        .alias("Gross_Revenue"),

    pl.when(pl.col("IsReturn") == True)
        .then(pl.col("Revenue"))
        .otherwise(0)
        .sum()
        .alias("Revenue_Returns"),
])
# --- Derived Metrics ---
.with_columns([
    (pl.col("Units_Returned") /
     (pl.col("Units_Sold") + pl.col("Units_Returned")))
     .fill_null(0)
     .alias("Return_Rate_Units"),

    (pl.col("Revenue_Returns") /
```

```
        (pl.col("Gross_Revenue") + pl.col("Revenue_Returns")))
        .fill_null(0)
        .alias("Return_Rate_Value"),

       (pl.col("Gross_Revenue") - pl.col("Revenue_Returns"))
        .alias("Net_Revenue"),
    ])
    .sort("Revenue_Returns", descending=False)
)

# remove null values
product_returns = product_returns.drop_nulls()

# Top products by revenue lost to returns
top_products_return_impact = product_returns.head(10)

print(top_products_return_impact.head(10))
```



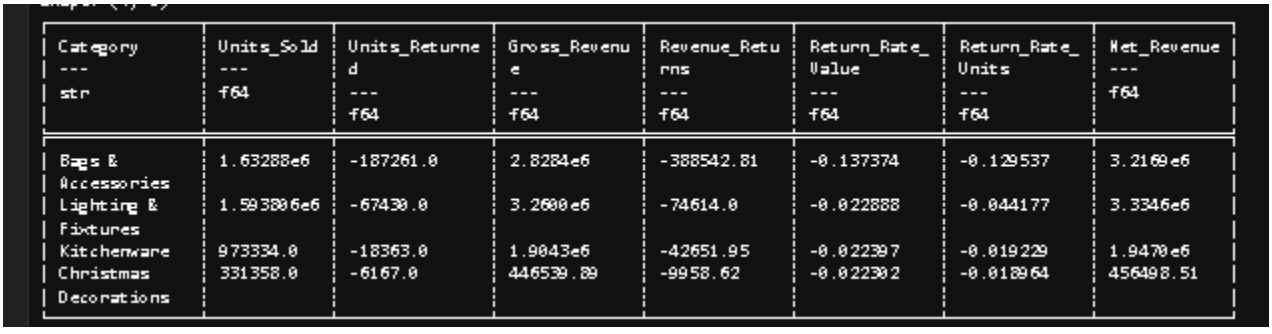
shape: (10, 9)

StockCode	Description	Units_Sold	Units_Returned	…	Revenue_Returns	Return_Rate_Units	Return_Rate_Value	Net_Revenue
str	str	f64	f64		f64	f64	f64	f64
AMAZONFEE	AMAZON FEE	3.0	-20.0	…	-177020.24	1.176471	1.235908	210809.61
M	Manual	3431.0	-3113.0	…	-137337.58	-9.789308	2.786241	225383.81
POST	POSTAGE	6340.0	-123.0	…	-10565.89	-0.019784	-0.236506	65806.77
D	Discount	0.0	-1230.0	…	-6969.48	1.0	1.0	6969.48
22423	REGENCY CAKESTAND 3 TIER	11297.0	-701.0	…	-5744.85	-0.066157	-0.042392	147007.21
BANK CHARGES	Bank Charges	15.0	-31.0	…	-4367.69	1.9375	1.054313	4592.69
20971	PINK BLUE FELT CRAFT TRINKET B…	12609.0	-3840.0	…	-4079.52	-0.437906	-0.357032	19585.23
84078A	SET/4 WHITE RETRO STORAGE CUBE…	636.0	-119.0	…	-3959.35	-0.230174	-0.19421	28305.67
22273	FELTCRAFT DOLL MOLLY	4139.0	-1490.0	…	-3533.3	-0.562476	-0.47121	14564.95
21735	TWO DOOR CURIO CABINET	457.0	-303.0	…	-2118.45	-1.967532	-0.546084	8116.25

```
# Summarize at category level for Revenue Returns
if "Category" in df.columns:
    category_returns = (
```

```
    df.group_by("Category")
      .agg([
        # --- Units Sold ---
        pl.when(pl.col("IsReturn") == False)
          .then(pl.col("Quantity"))
          .otherwise(0)
          .sum()
          .alias("Units_Sold"),

        # --- Units Returned ---
        pl.when(pl.col("IsReturn") == True)
          .then(pl.col("Quantity"))
          .otherwise(0)
          .sum()
          .alias("Units_Returned"),

        # --- Gross Revenue (all sales, positive only) ---
        pl.when(pl.col("IsReturn") == False)
          .then(pl.col("Revenue"))
          .otherwise(0)
          .sum()
          .alias("Gross_Revenue"),

        # --- Revenue Lost to Returns ---
        pl.when(pl.col("IsReturn") == True)
          .then(pl.col("Revenue"))
          .otherwise(0)
          .sum()
          .alias("Revenue_Returns"),
      ])
      .with_columns([
```

```
            # --- Derived Ratios ---
            (pl.col("Revenue_Returns") / pl.col("Gross_Revenue"))
              .fill_null(0)
              .alias("Return_Rate_Value"),

            (pl.col("Units_Returned") /
             (pl.col("Units_Sold") + pl.col("Units_Returned")))
             .fill_null(0)
             .alias("Return_Rate_Units"),

            (pl.col("Gross_Revenue") - pl.col("Revenue_Returns"))
              .alias("Net_Revenue"),
        ])
        .sort("Revenue_Returns", descending=False)
    )
else:
    category_returns = None

# remove null values
category_returns = category_returns.drop_nulls()

print(category_returns.head(10))
```



Category --- str	Units_Sold --- f64	Units_Returned --- f64	Gross_Revenue --- f64	Revenue_Returns --- f64	Return_Rate_Value --- f64	Return_Rate_Units --- f64	Net_Revenue --- f64
Bags & Accessories	1.63288e6	-187261.0	2.8284e6	-388542.81	-0.137374	-0.129537	3.2169e6
Lighting & Fixtures	1.593806e6	-67430.0	3.2600e6	-74614.0	-0.022888	-0.044177	3.3346e6
Kitchenware	973334.0	-18363.0	1.9043e6	-42651.95	-0.022397	-0.019229	1.9470e6
Christmas Decorations	331358.0	-6167.0	446539.89	-9958.62	-0.022302	-0.018964	456498.51

Summarize at country level for Revenue Return

if "Country" in df.columns:

```
country_returns = (
    df.group_by("Country")
      .agg([
          # --- Units Sold (non-return transactions) ---
          pl.when(pl.col("IsReturn") == False)
            .then(pl.col("Quantity"))
            .otherwise(0)
            .sum()
            .alias("Units_Sold"),

          # --- Units Returned ---
          pl.when(pl.col("IsReturn") == True)
            .then(pl.col("Quantity"))
            .otherwise(0)
            .sum()
            .alias("Units_Returned"),

          # --- Gross Revenue (sales only) ---
          pl.when(pl.col("IsReturn") == False)
            .then(pl.col("Revenue"))
            .otherwise(0)
            .sum()
            .alias("Gross_Revenue"),

          # --- Revenue Lost to Returns ---
          pl.when(pl.col("IsReturn") == True)
            .then(pl.col("Revenue"))
            .otherwise(0)
            .sum()
            .alias("Revenue_Returns"),
      ])
```

```
        .with_columns([
            # --- Return Value Rate (% of revenue lost) ---
            (pl.col("Revenue_Returns") / pl.col("Gross_Revenue"))
              .fill_null(0)
              .alias("Return_Rate_Value"),

            # --- Return Unit Rate (% of units returned) ---
            (pl.col("Units_Returned") /
             (pl.col("Units_Sold") + pl.col("Units_Returned")))
              .fill_null(0)
              .alias("Return_Rate_Units"),

            # --- Net Revenue after returns ---
            (pl.col("Gross_Revenue") - pl.col("Revenue_Returns"))
              .alias("Net_Revenue"),
        ])
        .sort("Revenue_Returns", descending=False)
    )
else:
    country_returns = None

# remove null values
country_returns = country_returns.drop_nulls()

print(country_returns.head(10))
```

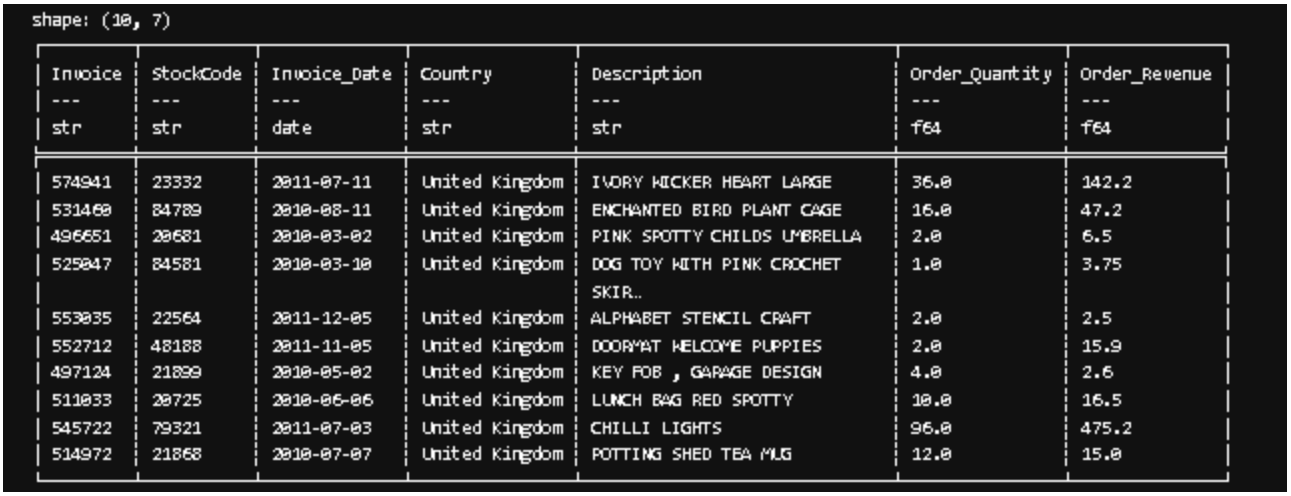
shape: (10, 8)

Country --- str	Units_Sold --- f64	Units_Returned --- f64	Gross_Revenue --- f64	Revenue_Returns --- f64	Return_Rate_Value --- f64	Return_Rate_Units --- f64	Net_Revenue --- f64
United Kingdom	3.873942e6	-363900.0	7.2717e6	-457263.18	-0.062882	-0.103674	7.7290e6
EIRE	140221.0	-3246.0	265849.3	-15891.68	-0.059777	-0.023698	281740.98
Germany	96855.0	-1481.0	180433.89	-7277.7	-0.040334	-0.015528	187711.59
Singapore	2633.0	-5.0	10994.4	-6090.84	-0.553995	-0.001903	17085.24
Spain	18218.0	-2349.0	42589.53	-5626.95	-0.132121	-0.148824	48216.48
France	173877.0	-1214.0	163637.63	-5231.22	-0.031968	-0.007031	168868.85
Japan	11631.0	-940.0	23321.6	-2260.61	-0.096932	-0.087924	25582.21
Australia	30472.0	-390.0	54616.34	-2064.08	-0.037792	-0.012965	56680.42
Hong Kong	733.0	-2.0	3423.29	-1894.6	-0.553444	-0.002736	5317.89
USA	2651.0	-1228.0	3371.2	-1579.51	-0.46853	-0.862966	4950.71

Detect large-order outliers and quantify their impact

Creating per-order (invoice x stockcode) order quantities and revenue (exclude returns)

```
orders = (
   df.filter(~pl.col("IsReturn"))
     .group_by(["Invoice", "StockCode"])
     .agg([
        pl.first("Invoice_Date").alias("Invoice_Date"),
        pl.first("Country").alias("Country"),
        pl.first("Description").alias("Description"),
        pl.sum("Quantity").alias("Order_Quantity"),
        pl.sum("Revenue").alias("Order_Revenue")
     ])
)

orders = orders.drop_nulls()

print(orders.head(10))
```

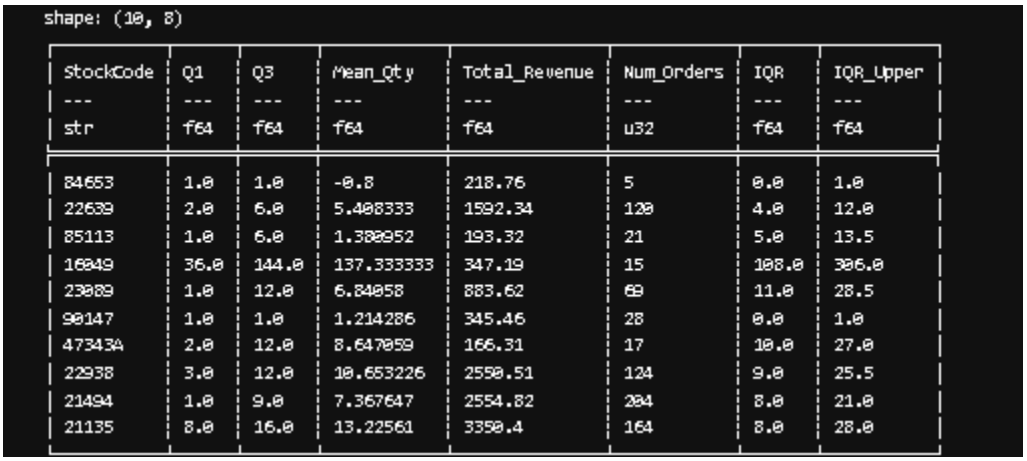


shape: (10, 7)

Invoice	StockCode	Invoice_Date	Country	Description	Order_Quantity	Order_Revenue
str	str	date	str	str	f64	f64
574941	23332	2011-07-11	United Kingdom	IVORY WICKER HEART LARGE	36.0	142.2
531460	84789	2010-08-11	United Kingdom	ENCHANTED BIRD PLANT CAGE	16.0	47.2
496651	20681	2010-03-02	United Kingdom	PINK SPOTTY CHILDS UMBRELLA	2.0	6.5
525047	84581	2010-03-10	United Kingdom	DOG TOY WITH PINK CROCHET SKIR…	1.0	3.75
553035	22564	2011-12-05	United Kingdom	ALPHABET STENCIL CRAFT	2.0	2.5
552712	48188	2011-11-05	United Kingdom	DOORMAT WELCOME PUPPIES	2.0	15.9
497124	21899	2010-05-02	United Kingdom	KEY FOB , GARAGE DESIGN	4.0	2.6
511033	20725	2010-06-06	United Kingdom	LUNCH BAG RED SPOTTY	10.0	16.5
545722	79321	2011-07-03	United Kingdom	CHILLI LIGHTS	96.0	475.2
514972	21868	2010-07-07	United Kingdom	POTTING SHED TEA MUG	12.0	15.0

```
# For each StockCode compute distribution stats (Q1, Q3, IQR) to detect IQR outliers

source_df = df  # replace with the real variable holding StockCode data

iqr_multiplier = 1.5

# Compute per-product statistics
stock_stats = (
    source_df.group_by("StockCode")
        .agg([
            pl.col("Quantity").quantile(0.25).alias("Q1"),
            pl.col("Quantity").quantile(0.75).alias("Q3"),
            pl.col("Quantity").mean().alias("Mean_Qty"),
            pl.col("Revenue").sum().alias("Total_Revenue"),
            pl.len().alias("Num_Orders"),
        ])
        .with_columns([
            (pl.col("Q3") - pl.col("Q1")).alias("IQR"),
            (pl.col("Q3") + (pl.col("Q3") - pl.col("Q1")) * iqr_multiplier).alias("IQR_Upper"),
        ])
        .drop_nulls()
)
```

print(stock_stats.head(10))



shape: (10, 8)

StockCode	Q1	Q3	Mean_Qty	Total_Revenue	Num_Orders	IQR	IQR_Upper
str	f64	f64	f64	f64	u32	f64	f64
84653	1.0	1.0	-0.8	218.76	5	0.0	1.0
22639	2.0	6.0	5.408333	1592.34	120	4.0	12.0
85113	1.0	6.0	1.380952	193.32	21	5.0	13.5
16049	36.0	144.0	137.333333	347.19	15	108.0	306.0
23089	1.0	12.0	6.84058	883.62	69	11.0	28.5
90147	1.0	1.0	1.214286	345.46	28	0.0	1.0
47343A	2.0	12.0	8.647059	166.31	17	10.0	27.0
22938	3.0	12.0	10.653226	2550.51	124	9.0	25.5
21494	1.0	9.0	7.367647	2554.82	204	8.0	21.0
21135	8.0	16.0	13.22561	3350.4	164	8.0	28.0

Join orders with stock_stats to flag IQR outliers

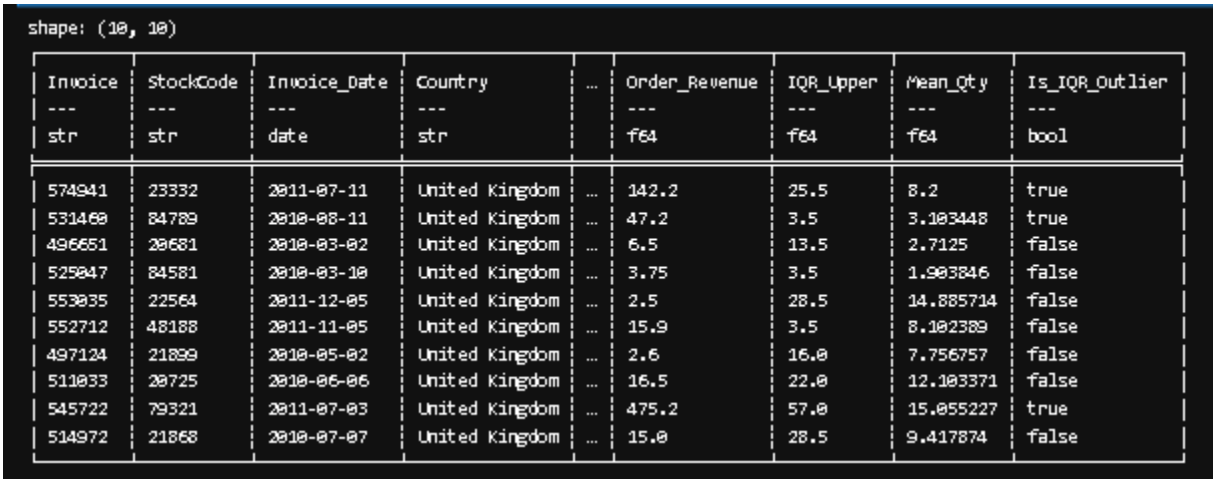
orders_with_stats = orders.join(stock_stats.select(["StockCode", "IQR_Upper", "Mean_Qty"]), on="StockCode", how="left")

orders_with_stats = orders_with_stats.with_columns(

(pl.col("Order_Quantity") > pl.col("IQR_Upper")).alias("Is_IQR_Outlier")

)

print(orders_with_stats.head(10))



shape: (10, 10)

Invoice	StockCode	Invoice_Date	Country	…	Order_Revenue	IQR_Upper	Mean_Qty	Is_IQR_Outlier
str	str	date	str		f64	f64	f64	bool
574941	23332	2011-07-11	United Kingdom	…	142.2	25.5	8.2	true
531460	84789	2010-08-11	United Kingdom	…	47.2	3.5	3.103448	true
496651	20681	2010-03-02	United Kingdom	…	6.5	13.5	2.7125	false
525847	84581	2010-03-10	United Kingdom	…	3.75	3.5	1.903846	false
553035	22564	2011-12-05	United Kingdom	…	2.5	28.5	14.885714	false
552712	48188	2011-11-05	United Kingdom	…	15.9	3.5	8.102389	false
497124	21899	2010-05-02	United Kingdom	…	2.6	16.0	7.756757	false
511033	20725	2010-06-06	United Kingdom	…	16.5	22.0	12.103371	false
545722	79321	2011-07-03	United Kingdom	…	475.2	57.0	15.055227	true
514972	21868	2010-07-07	United Kingdom	…	15.0	28.5	9.417874	false

Flag top percentile outliers (e.g. top 1%)

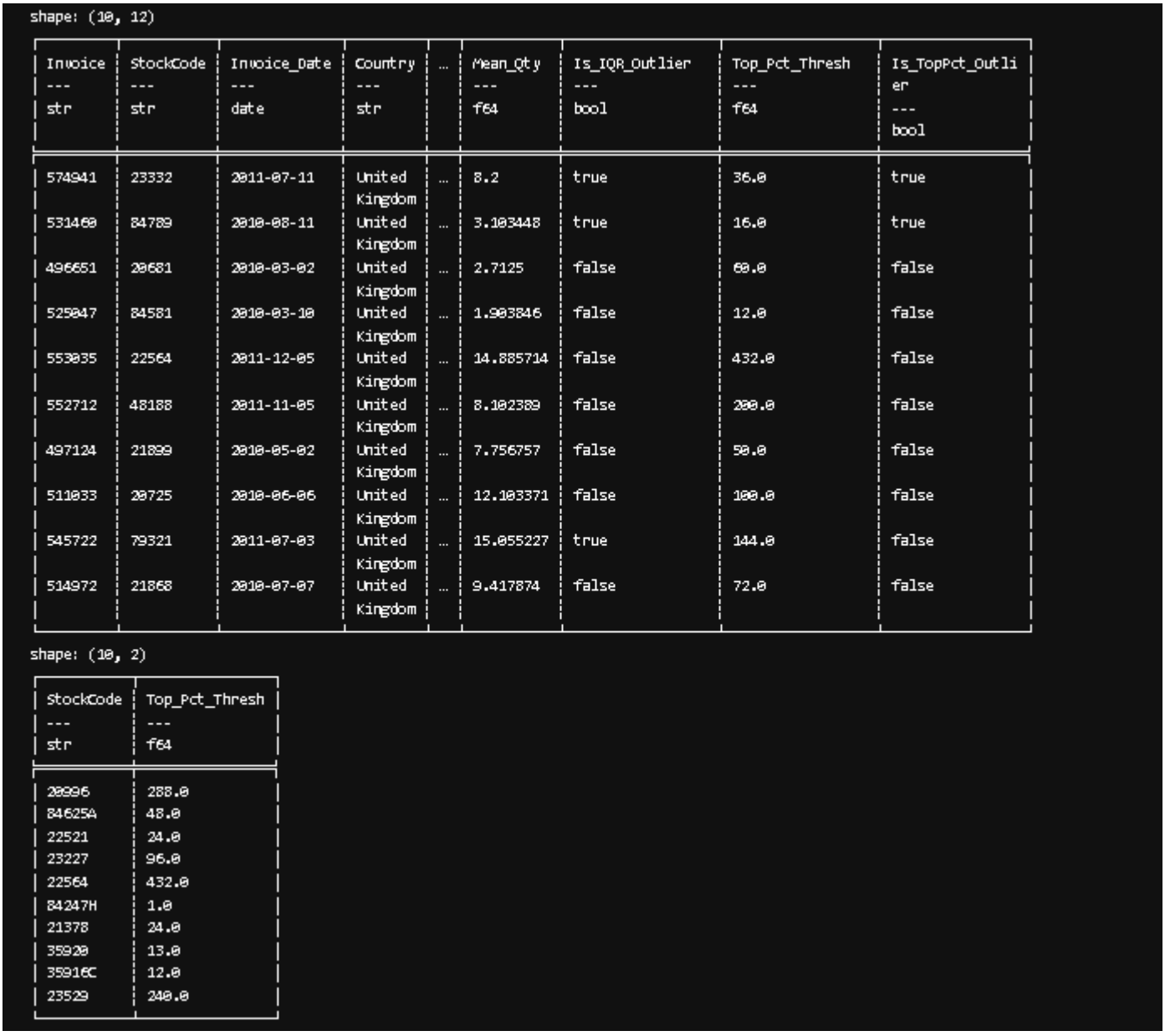
qty_thresholds = (

orders.group_by("StockCode")

.agg(pl.col("Order_Quantity").quantile(outlier_pct).alias("Top_Pct_Thresh"))

)

```
orders_with_stats = orders_with_stats.join(qty_thresholds, on="StockCode", how="left")
orders_with_stats = orders_with_stats.with_columns(
    (pl.col("Order_Quantity") >= pl.col("Top_Pct_Thresh")).alias("Is_TopPct_Outlier")
)


orders_with_stats.sort("Top_Pct_Thresh", descending=False)
qty_thresholds.sort("Top_Pct_Thresh", descending=False)

print(orders_with_stats.head(10))
print(qty_thresholds.head(10))
```



shape: (10, 12)

Invoice	StockCode	Invoice_Date	Country	…	Mean_Qty	Is_IQR_Outlier	Top_Pct_Thresh	Is_TopPct_Outlier
str	str	date	str		f64	bool	f64	bool
574941	23332	2011-07-11	United Kingdom	…	8.2	true	36.0	true
531460	84789	2010-08-11	United Kingdom	…	3.103448	true	16.0	true
496651	20681	2010-03-02	United Kingdom	…	2.7125	false	60.0	false
525847	84581	2010-03-10	United Kingdom	…	1.903846	false	12.0	false
553035	22564	2011-12-05	United Kingdom	…	14.885714	false	432.0	false
552712	48188	2011-11-05	United Kingdom	…	8.102389	false	200.0	false
497124	21899	2010-05-02	United Kingdom	…	7.756757	false	50.0	false
511033	20725	2010-06-06	United Kingdom	…	12.103371	false	100.0	false
545722	79321	2011-07-03	United Kingdom	…	15.055227	true	144.0	false
514972	21868	2010-07-07	United Kingdom	…	9.417874	false	72.0	false

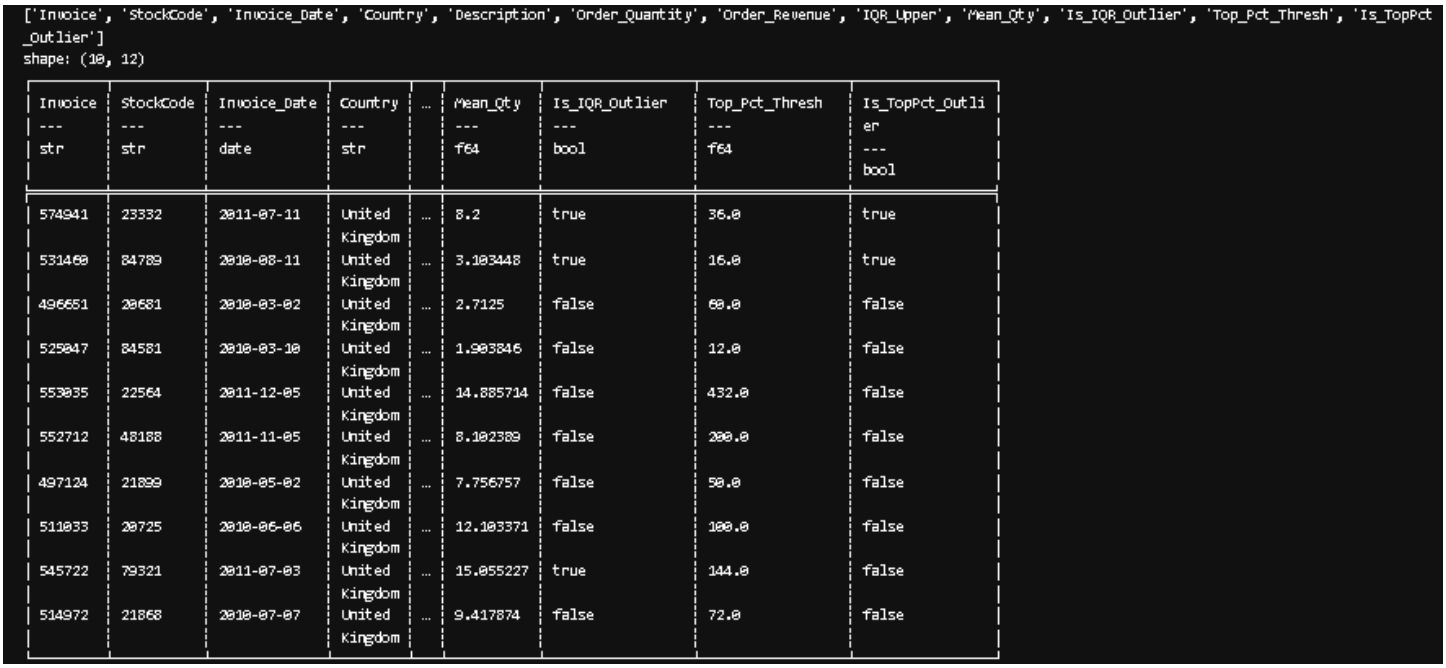
shape: (10, 2)

StockCode	Top_Pct_Thresh
str	f64
20996	288.0
84625A	48.0
22521	24.0
23227	96.0
22564	432.0
84247H	1.0
21378	24.0
35920	13.0
35916C	12.0
23529	240.0

Quantify impact of outliers at product level (share of units and revenue)

```
print(orders_with_stats.columns)

outlier_impact_by_stock = (
    orders_with_stats.group_by("StockCode")
        .agg([
            pl.col("Order_Quantity").sum().alias("Total_Units"),
            pl.col("Order_Revenue").sum().alias("Total_Revenue"),
```

```
        # Compute outlier totals directly via .sum() on the expression
        (
            pl.when(pl.col("Is_IQR_Outlier") | pl.col("Is_TopPct_Outlier"))
              .then(pl.col("Order_Quantity"))
              .otherwise(0)
              .sum()
              .alias("Outlier_Units")
        ),
        (
            pl.when(pl.col("Is_IQR_Outlier") | pl.col("Is_TopPct_Outlier"))
              .then(pl.col("Order_Revenue"))
              .otherwise(0)
              .sum()
              .alias("Outlier_Revenue")
        ),
        pl.col("Description").first().alias("Description"),
    ])
    .with_columns([
        (pl.col("Outlier_Units") / pl.col("Total_Units"))
            .fill_null(0)
            .alias("Outlier_Units_Share"),
        (pl.col("Outlier_Revenue") / pl.col("Total_Revenue"))
            .fill_null(0)
            .alias("Outlier_Revenue_Share"),
    ])
    .sort("Outlier_Revenue", descending=True)
)

print(orders_with_stats.head(10))
```

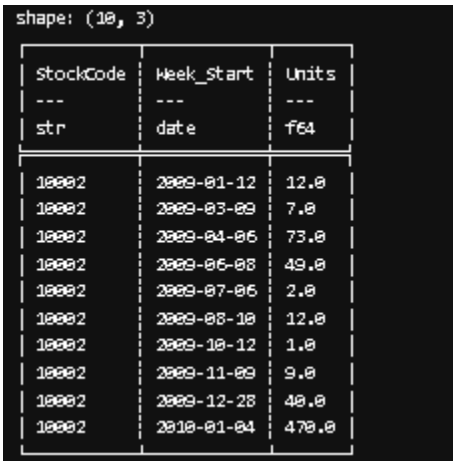


```
['Invoice', 'StockCode', 'Invoice_Date', 'Country', 'Description', 'Order_Quantity', 'Order_Revenue', 'IQR_Upper', 'Mean_Qty', 'Is_IQR_Outlier', 'Top_Pct_Thresh', 'Is_TopPct
_Outlier']
shape: (10, 12)
```

Invoice	StockCode	Invoice_Date	Country	…	Mean_Qty	Is_IQR_Outlier	Top_Pct_Thresh	Is_TopPct_Outli
str	str	date	str		f64	bool	f64	er
								bool
574941	23332	2011-07-11	United Kingdom	…	8.2	true	36.0	true
531460	84789	2010-08-11	United Kingdom	…	3.103448	true	16.0	true
496651	20681	2010-03-02	United Kingdom	…	2.7125	false	60.0	false
525847	84581	2010-03-10	United Kingdom	…	1.903846	false	12.0	false
553035	22564	2011-12-05	United Kingdom	…	14.885714	false	432.0	false
552712	48188	2011-11-05	United Kingdom	…	8.102389	false	200.0	false
497124	21899	2010-05-02	United Kingdom	…	7.756757	false	50.0	false
511033	20725	2010-06-06	United Kingdom	…	12.103371	false	100.0	false
545722	79321	2011-07-03	United Kingdom	…	15.055227	true	144.0	false
514972	21868	2010-07-07	United Kingdom	…	9.417874	false	72.0	false

```
# Simple baseline forecast for inventory planning
# Using average weekly demand per StockCode as baseline forecast.
# Then compute a simple safety stock = z * std_weekly * sqrt(lead_time_weeks)
# Reorder recommendation = forecast_for_one_week * lead_time_weeks + safety_stock

lead_time_weeks = 2.0      # lead time in weeks (tunable)

# Use the Week column (week start Monday)
df_sales = df.filter(~pl.col("IsReturn")).select(["StockCode", "Invoice_Date", "Quantity"])
df_sales = df_sales.with_columns(pl.col("Invoice_Date").alias("Date"))
df_sales = df_sales.with_columns(
   (pl.col("Invoice_Date").cast(pl.Datetime).dt.truncate("1w")).dt.date().alias("Week_Start")
)

weekly = (
   df_sales.group_by(["StockCode", "Week_Start"])
         .agg(pl.sum("Quantity").alias("Units"))
         .sort(["StockCode", "Week_Start"])
)
```

print(weekly.head(10))



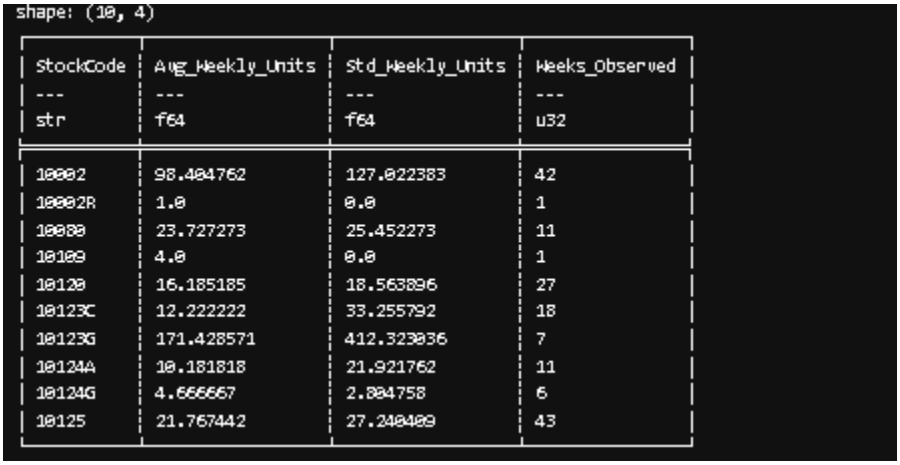
shape: (10, 3)

StockCode	Week_Start	Units
str	date	f64
10002	2009-01-12	12.0
10002	2009-03-09	7.0
10002	2009-04-06	73.0
10002	2009-06-08	49.0
10002	2009-07-06	2.0
10002	2009-08-10	12.0
10002	2009-10-12	1.0
10002	2009-11-09	9.0
10002	2009-12-28	40.0
10002	2010-01-04	470.0

per-stock weekly stats (mean & std)

```
weekly_stats = (
    weekly.group_by("StockCode")
        .agg([
          pl.col("Units").mean().alias("Avg_Weekly_Units"),
          pl.col("Units").std().fill_null(0).alias("Std_Weekly_Units"),
          pl.count().alias("Weeks_Observed")
        ])
)
```

print(weekly_stats.head(10))



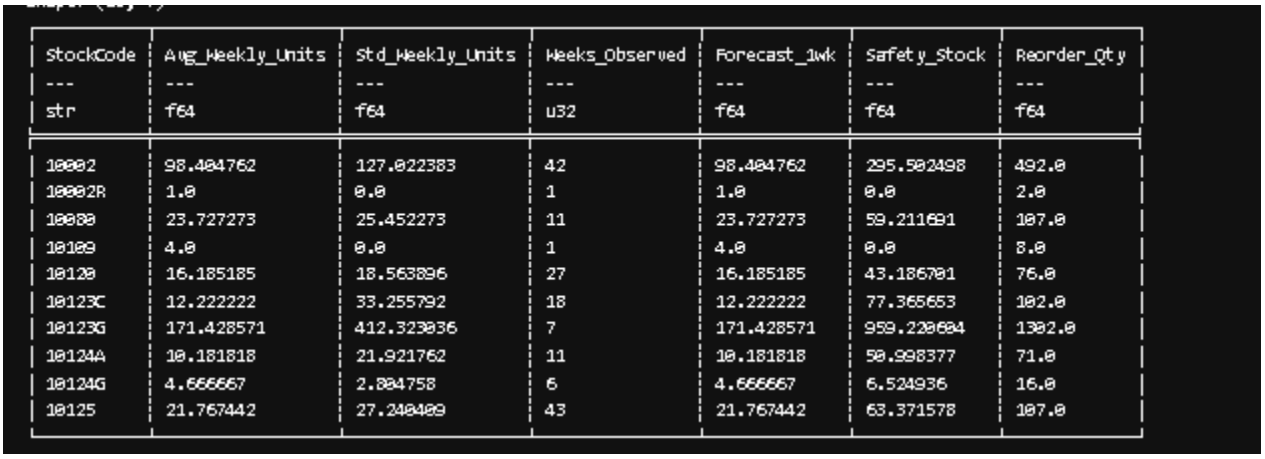
shape: (10, 4)

StockCode	Avg_Weekly_Units	Std_Weekly_Units	Weeks_Observed
str	f64	f64	u32
10002	98.404762	127.022383	42
10002R	1.0	0.0	1
10080	23.727273	25.452273	11
10109	4.0	0.0	1
10120	16.185185	18.563896	27
10123C	12.222222	33.255792	18
10123G	171.428571	412.323036	7
10124A	10.181818	21.921762	11
10124G	4.666667	2.804758	6
10125	21.767442	27.240489	43

Merge stats back to compute safety stock and recommended reorder

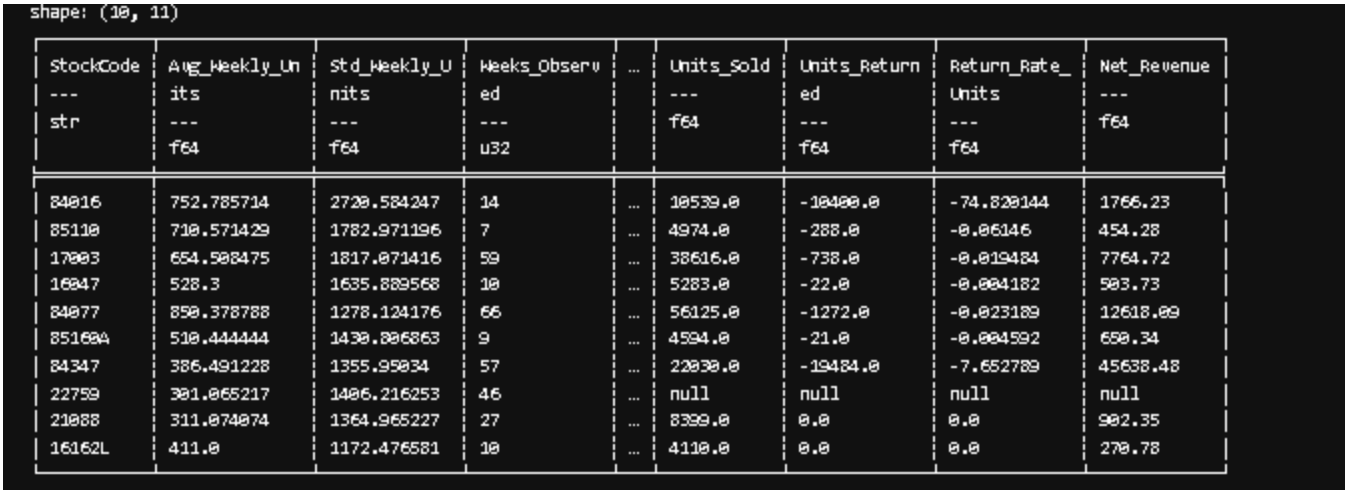
forecast_df = weekly_stats.with_columns([

```
    # forecast for next week (baseline = average weekly)
    pl.col("Avg_Weekly_Units").alias("Forecast_1wk"),
    # safety stock scaled by sqrt(lead time in weeks)
    (pl.col("Std_Weekly_Units") * pl.lit(service_z) * pl.lit(math.sqrt(lead_time_weeks))).alias("Safety_Stock"),
    # reorder = forecast*lead_time + safety stock
    (pl.col("Avg_Weekly_Units") * pl.lit(lead_time_weeks) + (pl.col("Std_Weekly_Units") * pl.lit(service_z) *
pl.lit(math.sqrt(lead_time_weeks)))).round(0).alias("Reorder_Qty")
])

print(forecast_df.head(10))
```



StockCode	Avg_Weekly_Units	Std_Weekly_Units	Weeks_Observed	Forecast_1wk	Safety_Stock	Reorder_Qty
str	f64	f64	u32	f64	f64	f64
10002	98.404762	127.022383	42	98.404762	295.502498	492.0
10002R	1.0	0.0	1	1.0	0.0	2.0
10080	23.727273	25.452273	11	23.727273	59.211691	107.0
10109	4.0	0.0	1	4.0	0.0	8.0
10120	16.185185	18.563896	27	16.185185	43.186701	76.0
10123C	12.222222	33.255792	18	12.222222	77.365653	102.0
10123G	171.428571	412.323036	7	171.428571	959.220604	1302.0
10124A	10.181818	21.921762	11	10.181818	50.998377	71.0
10124G	4.666667	2.804758	6	4.666667	6.524936	16.0
10125	21.767442	27.240409	43	21.767442	63.371578	107.0

```
# Add product description and some current returns stats for context
# Join reorder with product_returns for context
forecast_df = forecast_df.join(
    product_returns.select(["StockCode", "Units_Sold", "Units_Returned", "Return_Rate_Units", "Net_Revenue"]),
    on="StockCode",
    how="left"
).select([
    "StockCode", "Avg_Weekly_Units", "Std_Weekly_Units", "Weeks_Observed",
    "Forecast_1wk", "Safety_Stock", "Reorder_Qty", "Units_Sold", "Units_Returned", "Return_Rate_Units",
"Net_Revenue"
]).sort("Reorder_Qty", descending=True)

print(forecast_df.head(10))
```



shape: (10, 11)

StockCode	Avg_Weekly_Units	Std_Weekly_Units	Weeks_Observed	…	Units_Sold	Units_Returned	Return_Rate_Units	Net_Revenue
str	f64	f64	u32		f64	f64	f64	f64
84016	752.785714	2720.584247	14	…	10539.0	-10400.0	-74.820144	1766.23
85110	710.571429	1782.971196	7	…	4974.0	-288.0	-0.06146	454.28
17003	654.508475	1817.071416	59	…	38616.0	-738.0	-0.019484	7764.72
16047	528.3	1635.889568	10	…	5283.0	-22.0	-0.004182	503.73
84077	850.378788	1278.124176	66	…	56125.0	-1272.0	-0.023189	12618.09
85160A	510.444444	1430.806863	9	…	4594.0	-21.0	-0.004592	650.34
84347	386.491228	1355.95034	57	…	22030.0	-19484.0	-7.652789	45638.48
22759	301.065217	1406.216253	46	…	null	null	null	null
21088	311.074074	1364.965227	27	…	8399.0	0.0	0.0	902.35
16162L	411.0	1172.476581	10	…	4110.0	0.0	0.0	270.78

Export to Excel

print(product_returns)

print(outlier_impact_by_stock)

print(forecast_df)

product_returns.write_csv("product_returns_summary.csv")

outlier_impact_by_stock.write_csv("outlier_impact_by_stock.csv")

forecast_df.write_csv("reorder_recommendations.csv")

StockCode	Description	Units_Sold	Units_Returned	…	Revenue_Returns	Return_Rate_Units	Return_Rate_Value	Net_Revenue
str	str	f64	f64		f64	f64	f64	f64
AMAZONFEE	AMAZON FEE	3.0	-20.0	…	-177020.24	1.176471	1.235908	210809.61
M	Manual	3431.0	-3113.0	…	-137337.58	-9.789308	2.786241	225383.81
POST	POSTAGE	6340.0	-123.0	…	-10565.89	-0.019784	-0.236506	65806.77
D	Discount	0.0	-1230.0	…	-6969.48	1.0	1.0	6969.48
22423	REGENCY CAKESTAND 3 TIER	11297.0	-701.0	…	-5744.85	-0.066157	-0.042392	147007.21
…	…	…	…	…	…	…	…	…
22577	WOODEN HEART CHRISTMAS SCANDIN…	4852.0	-300.0	…	0.0	-0.065905	0.0	2700.5
84997b	RED 3 PIECE MINI DOTS CUTLERY …	206.0	0.0	…	0.0	0.0	0.0	1735.42
84997d	PINK 3 PIECE POLKADOT CUTLERY …	154.0	0.0	…	0.0	0.0	0.0	1290.32
22286	DECORATION , WOBBLY RABBIT , M…	735.0	0.0	…	0.0	0.0	0.0	872.11
84623	PINK GINGHAM ROSE FLOOR CUSHIO…	42.0	-53.0	…	0.0	4.818182	0.0	190.96

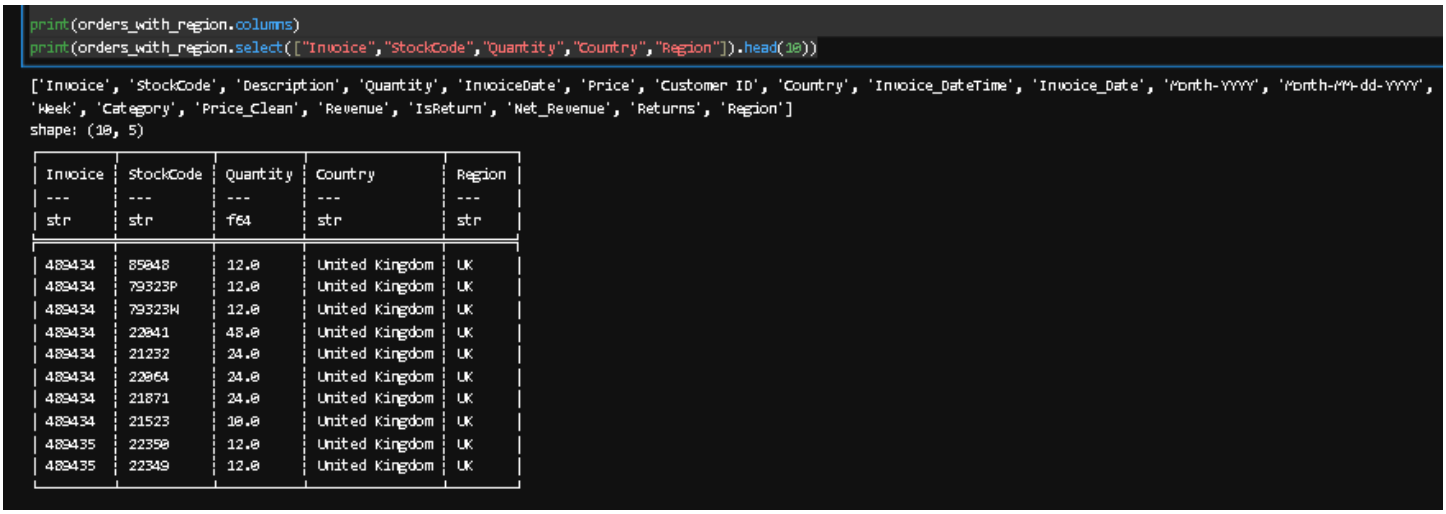
shape: (4_747, 8)

StockCode	Total_Units	Total_Revenue	Outlier_Units	Outlier_Revenue	Description	Outlier_Units_Share	Outlier_Revenue_Share
str	f64	f64	f64	f64	str	f64	f64
22423	11297.0	141262.36	7985.0	93897.83	REGENCY CAKESTAND 3 TIER	0.706825	0.664705
85123A	46934.0	114683.12	33762.0	77542.62	WHITE HANGING HEART T-LIGHT HO…	0.733414	0.676147
85099B	44597.0	81951.12	31772.0	54512.13	JUMBO BAG RED RETROSPOT	0.712425	0.665179
22502	1909.0	49800.23	976.0	43185.44	PICNIC BASKET WICKER SMALL	0.511262	0.867174
22086	17280.0	56150.73	12338.0	39557.64	PAPER CHAIN KIT 50'S CHRISTMAS	0.714005	0.70449
…	…	…	…	…	…	…	…
49031B	400.0	0.0	400.0	0.0	CHROME EURO HOOK 20cm	1.0	NaN
35988C	430.0	98.8	170.0	0.0	REDWHITE STRIPE SCANDINAVIAN H…	0.395349	0.0
23001	200.0	0.0	200.0	0.0	TRAVEL CARD WALLET DOTCOMGIFTS…	1.0	NaN
PADS	4.0	0.0	4.0	0.0	PADS TO	1.0	NaN

Compute Average Order Value (AOV) per region, then run a two-sample t-test (UK vs Non-UK).

Polars handles aggregation; SciPy handles the statistical test.

```
import polars as pl
from scipy.stats import ttest_ind

# Define region classification (U.K vs Non-U.K) = Data Cleaning to add Region
orders_with_region = df.with_columns([
    pl.when(pl.col("Country") == "United Kingdom")
      .then(pl.lit("UK"))        # use pl.lit() for string literals as row value in each column
      .otherwise(pl.lit("Non-UK"))
      .alias("Region")
])

print(orders_with_region.columns)
print(orders_with_region.select(["Invoice","StockCode","Quantity","Country","Region"]).head(10))
```




```
# Compute Revenue per row (if not yet computed)
if "Revenue" not in orders_with_region.columns:
    orders_with_region = orders_with_region.with_columns((pl.col("Quantity") * pl.col("Price")).alias("Revenue"))

print(orders_with_region.select(["Invoice","StockCode","Quantity","Price","Revenue","Country","Region"]).head(10))
```

shape: (10, 7)

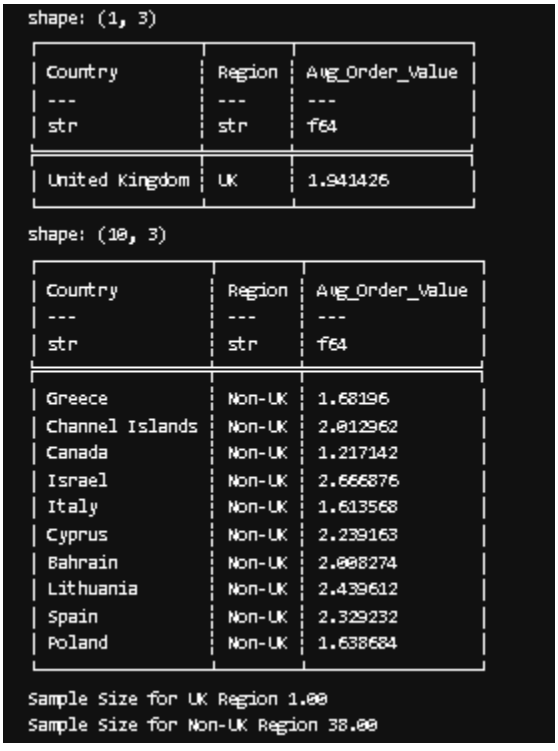
Invoice	StockCode	Quantity	Price	Revenue	Country	Region
str	str	f64	str	f64	str	str
489434	85048	12.0	6.95	83.4	United Kingdom	UK
489434	79323P	12.0	6.75	81.0	United Kingdom	UK
489434	79323W	12.0	6.75	81.0	United Kingdom	UK
489434	22041	48.0	2.1	100.8	United Kingdom	UK
489434	21232	24.0	1.25	30.0	United Kingdom	UK
489434	22064	24.0	1.65	39.6	United Kingdom	UK
489434	21871	24.0	1.25	30.0	United Kingdom	UK
489434	21523	10.0	5.95	59.5	United Kingdom	UK
489435	22350	12.0	2.55	30.6	United Kingdom	UK
489435	22349	12.0	3.75	45.0	United Kingdom	UK

```
# Compute average order value per Country (total revenue / total quantity)
country_aov = (
   orders_with_region.group_by(["Country", "Region"])
     .agg([
        pl.sum("Revenue").alias("Total_Revenue"),
        pl.sum("Quantity").alias("Total_Units")
     ])
     .with_columns([
        (pl.col("Total_Revenue") / pl.col("Total_Units"))
        .alias("Avg_Order_Value")
     ])
     .drop_nulls()
)


print(country_aov.head(10))
```

Country	Region	Total_Revenue	Total_Units	Avg_Order_Value
str	str	f64	f64	f64
Greece	Non-UK	5999.55	3567.0	1.68196
Channel Islands	Non-UK	12279.07	6100.0	2.012962
Canada	Non-UK	2133.65	1753.0	1.217142
Israel	Non-UK	4251.0	1594.0	2.666876
Italy	Non-UK	6778.6	4201.0	1.613568
Cyprus	Non-UK	8157.27	3643.0	2.239163
Bahrain	Non-UK	791.26	394.0	2.008274
Lithuania	Non-UK	4847.51	1987.0	2.439612
Spain	Non-UK	36962.58	15869.0	2.329232
Poland	Non-UK	6089.35	3716.0	1.638684

```
# Extract UK and Non-UK average order value arrays
import polars as pl
import numpy as np
from scipy.stats import ttest_ind

# Extract UK and Non-UK AOV arrays and clean NaN/infinite values
uk_aov = (
    country_aov
    .filter(pl.col("Region") == "UK")
    .select("Avg_Order_Value")
    .drop_nulls()
    .to_series()
    .to_numpy()
)

uk_aov = uk_aov[np.isfinite(uk_aov)]  # remove inf/nan



non_uk_aov = (
    country_aov
    .filter(pl.col("Region") == "Non-UK")
    .select("Avg_Order_Value")
    .drop_nulls()
    .to_series()
    .to_numpy()
)

non_uk_aov = non_uk_aov[np.isfinite(non_uk_aov)]  # remove inf/nan

# Extract top 10 Average Order Values for UK and Non-UK before converting to NumPy
uk_aov_top10 = (
```

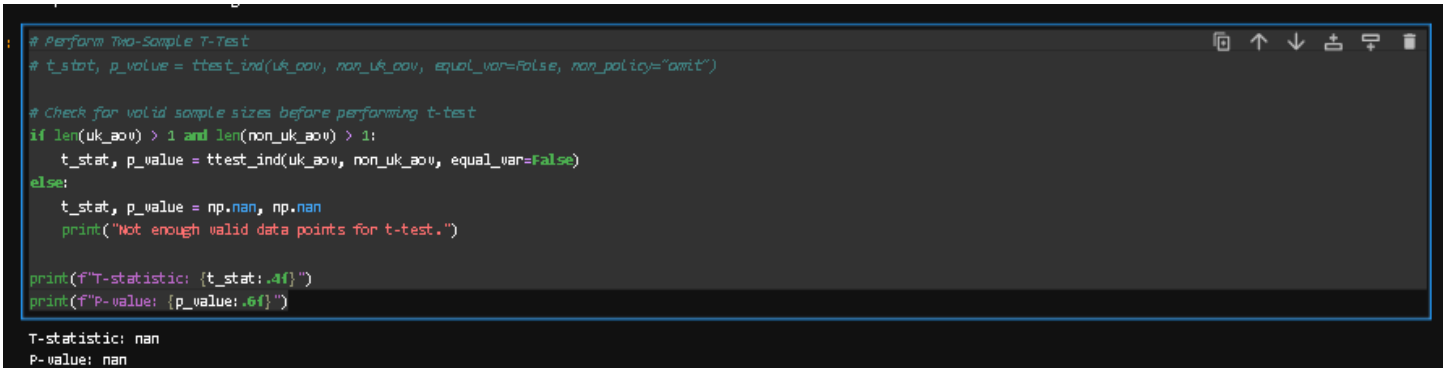
```
    country_aov.filter(pl.col("Region") == "UK")
            .select(["Country", "Region", "Avg_Order_Value"])
            .head(10)
)

non_uk_aov_top10 = (
    country_aov.filter(pl.col("Region") == "Non-UK")
            .select(["Country", "Region", "Avg_Order_Value"])
            .head(10)
)

print(uk_aov_top10)
print(non_uk_aov_top10)

print(f"Sample Size for UK Region {len(uk_aov):.2f}")
print(f"Sample Size for Non-UK Region {len(non_uk_aov):.2f}")
```



```
shape: (1, 3)
┌────────────────┬────────┬─────────────────┐
│ Country        ┆ Region ┆ Avg_Order_Value │
│ ---            ┆ ---    ┆ ---             │
│ str            ┆ str    ┆ f64             │
╞════════════════╪════════╪═════════════════╡
│ United Kingdom ┆ UK     ┆ 1.941426        │
└────────────────┴────────┴─────────────────┘
shape: (10, 3)
┌──────────────────┬────────┬─────────────────┐
│ Country          ┆ Region ┆ Avg_Order_Value │
│ ---              ┆ ---    ┆ ---             │
│ str              ┆ str    ┆ f64             │
╞══════════════════╪════════╪═════════════════╡
│ Greece           ┆ Non-UK ┆ 1.68196         │
│ Channel Islands  ┆ Non-UK ┆ 2.012962        │
│ Canada           ┆ Non-UK ┆ 1.217142        │
│ Israel           ┆ Non-UK ┆ 2.666876        │
│ Italy            ┆ Non-UK ┆ 1.613568        │
│ Cyprus           ┆ Non-UK ┆ 2.239163        │
│ Bahrain          ┆ Non-UK ┆ 2.008274        │
│ Lithuania        ┆ Non-UK ┆ 2.439612        │
│ Spain            ┆ Non-UK ┆ 2.329232        │
│ Poland           ┆ Non-UK ┆ 1.638684        │
└──────────────────┴────────┴─────────────────┘
Sample Size for UK Region 1.00
Sample Size for Non-UK Region 38.00
```

```
# Perform Two-Sample T-Test
# t_stat, p_value = ttest_ind(uk_aov, non_uk_aov, equal_var=False, nan_policy="omit")
```

```
# Check for valid sample sizes before performing t-test
if len(uk_aov) > 1 and len(non_uk_aov) > 1:
    t_stat, p_value = ttest_ind(uk_aov, non_uk_aov, equal_var=False)
else:
    t_stat, p_value = np.nan, np.nan
    print("Not enough valid data points for t-test.")

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.6f}")
```

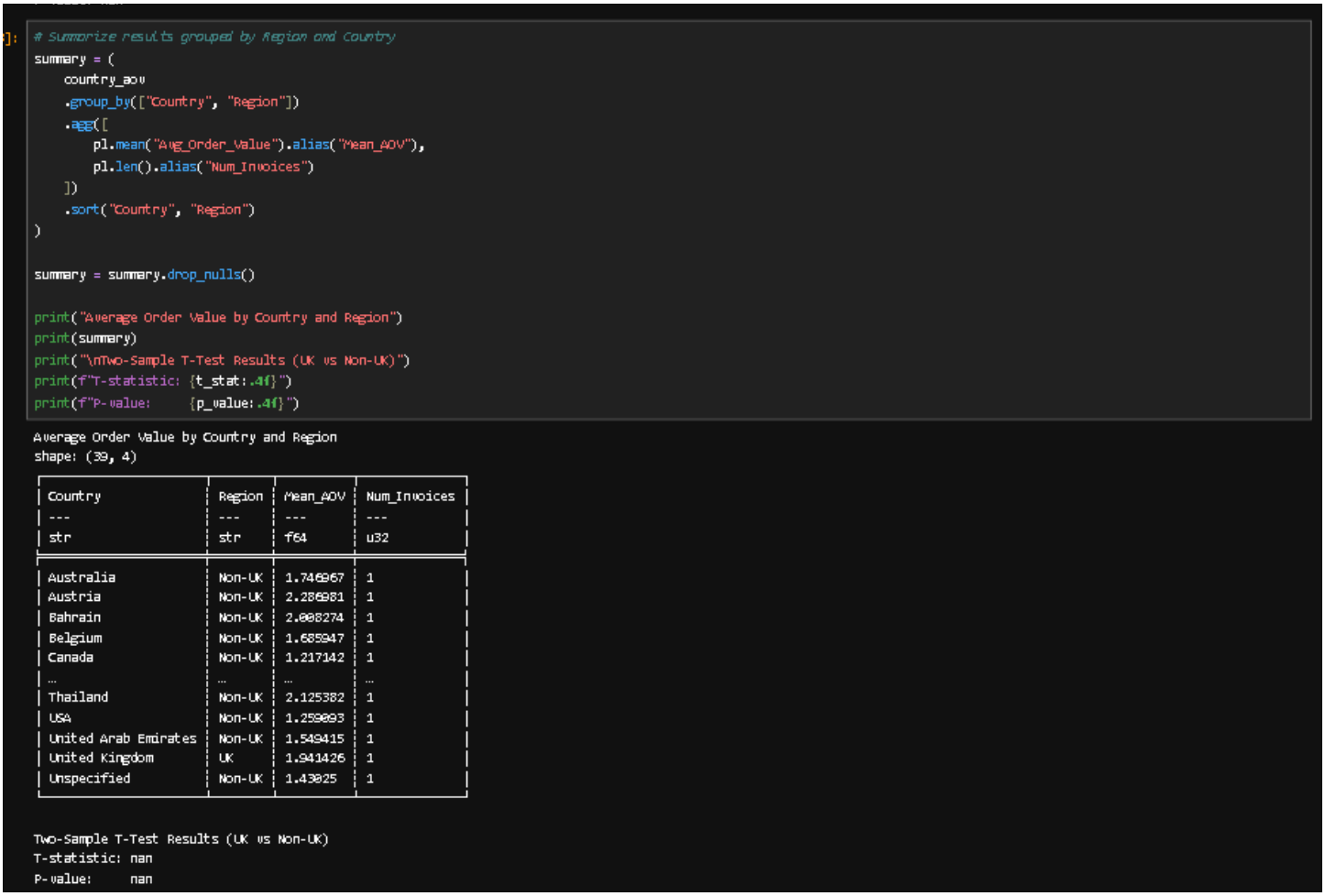



```
# Summarize results grouped by Region and Country
summary = (
    country_aov
    .group_by(["Country", "Region"])
    .agg([
        pl.mean("Avg_Order_Value").alias("Mean_AOV"),
        pl.len().alias("Num_Invoices")
    ])
    .sort("Country", "Region")
)

summary = summary.drop_nulls()

print("Average Order Value by Country and Region")
print(summary)
```

print("\nTwo-Sample T-Test Results (UK vs Non-UK)")

print(f"T-statistic: {t_stat:.4f}")

print(f"P-value: {p_value:.4f}")



```
]: # Summarize results grouped by Region and Country
summary = (
    country_aov
    .group_by(["Country", "Region"])
    .agg([
        pl.mean("Avg_Order_Value").alias("Mean_AOV"),
        pl.len().alias("Num_Invoices")
    ])
    .sort("Country", "Region")
)

summary = summary.drop_nulls()

print("Average Order Value by Country and Region")
print(summary)
print("\nTwo-Sample T-Test Results (UK vs Non-UK)")
print(f"T-statistic: {t_stat:.4f}")
print(f"P-value:     {p_value:.4f}")
```

```
Average Order Value by Country and Region
shape: (39, 4)
```

Country	Region	Mean_AOV	Num_Invoices
str	str	f64	u32
Australia	Non-UK	1.746867	1
Austria	Non-UK	2.286981	1
Bahrain	Non-UK	2.008274	1
Belgium	Non-UK	1.685947	1
Canada	Non-UK	1.217142	1
…	…	…	…
Thailand	Non-UK	2.125382	1
USA	Non-UK	1.259893	1
United Arab Emirates	Non-UK	1.549415	1
United Kingdom	UK	1.941426	1
Unspecified	Non-UK	1.43025	1

```
Two-Sample T-Test Results (UK vs Non-UK)
T-statistic: nan
P-value:     nan
```