BOSTON UNIVERSITY

AD 599S B1 Introduction to Python and SQL for Business Analytics (Summer 1 2025)

FINAL PROJECT _ PHASE 2

HUNG TRUNG NGUYEN - U89837429

SQL Queries - Phase 2:

Preparation Codes

```
!pip install seaborn
!pip install palettable
!pip install colorcet
!pip install plotly
```

```
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local/lib/python3.11/dist-packages (from seaborn) (2.0.2)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.11/dist-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib/python3.11/dist-packages (from seaborn) (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
Collecting palettable
  Downloading palettable-3.3.3-py2.py3-none-any.whl.metadata (3.3 kB)
Downloading palettable-3.3.3-py2.py3-none-any.whl (332 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 332.3/332.3 kB 5.4 MB/s eta 0:00:00
Installing collected packages: palettable
Successfully installed palettable-3.3.3
Requirement already satisfied: colorcet in /usr/local/lib/python3.11/dist-packages (3.1.0)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (5.24.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly) (8.5.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from plotly) (25.0)
```

```
# Import necessary libraries
import sqlite3
import pandas as pd
import seaborn as sns
import polars as pl
import os
import matplotlib.pyplot as plt
import colorcet as cc
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from palettable.colorbrewer.qualitative import Set2_5
from IPython.display import display


# Helpers
def create_connection(path):
    """Return a SQLite connection or None."""
    try:
        return sqlite3.connect(path)
    except sqlite3.Error as e:
        print("Conn error:", e)

def run_query(conn, sql):
    """Run a read-only SQL query and return a DataFrame."""
    try:
        return pd.read_sql_query(sql, conn)
    except Exception as e:
        print("Query error:", e)

# Verify and Connect
print("Files in CWD:", os.listdir("."))
```

```
db_path = "northwind.db"
conn = create_connection(db_path)

if conn:
    tables = run_query(conn, "SELECT name FROM sqlite_master WHERE type='table';")
    display(tables)
else:
    print("Failed to connect.")
```

```
Files in CWD: ['.config', 'northwind.db', 'sample_data']
```

	name
0	Categories
1	sqlite_sequence
2	CustomerCustomerDemo
3	CustomerDemographics
4	Customers
5	Employees
6	EmployeeTerritories
7	Order Details
8	Orders
9	Products
10	Regions
11	Shippers
12	Suppliers
13	Territories

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

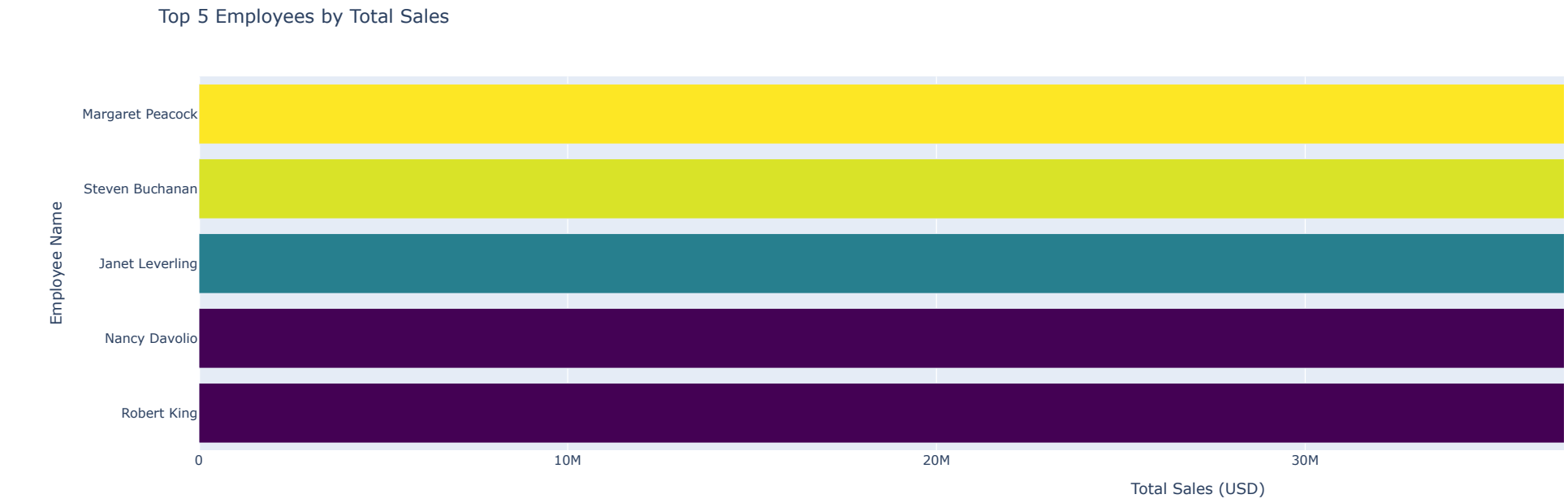
Part 1: Employee Sales Performance Analysis

```
# 1. SQL + CTE + RANK
emp_sql = """
WITH EmployeeSales AS (
  SELECT
    e.EmployeeID,
    e.FirstName || ' ' || e.LastName AS EmployeeName,
    SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS total_sales
  FROM Employees e
  JOIN Orders o           ON e.EmployeeID = o.EmployeeID
  JOIN "Order Details" od ON o.OrderID    = od.OrderID
  GROUP BY e.EmployeeID, EmployeeName
)
SELECT
  EmployeeID,
  EmployeeName,
  total_sales,
  RANK() OVER (ORDER BY total_sales DESC) AS sales_rank
FROM EmployeeSales
ORDER BY sales_rank;
"""


# 2. Load via pandas → Polars
emp_pd = pd.read_sql_query(emp_sql, conn)
emp_df = pl.from_pandas(emp_pd)

# 3. Filter Top 5
top5 = emp_df.filter(pl.col("sales_rank") <= 5).sort("sales_rank")
display(top5)
```

shape: (5, 4)

EmployeeID	EmployeeName	total_sales	sales_rank
i64	str	f64	i64
4	"Margaret Peacock"	5.1488e7	1
5	"Steven Buchanan"	5.1386e7	2
3	"Janet Leverling"	5.0446e7	3
1	"Nancy Davolio"	4.9659e7	4
7	"Robert King"	4.9652e7	5

```
# 4. Interactive bar chart with Plotly Express
fig1 = px.bar(
    top5.to_pandas(),
    x="total_sales",
    y="EmployeeName",
    orientation="h",
    text="total_sales",
    color="total_sales",
    color_continuous_scale="Viridis",
    title="Top 5 Employees by Total Sales"
)
fig1.update_layout(
    yaxis_categoryorder="total ascending",
    xaxis_title="Total Sales (USD)",
    yaxis_title="Employee Name"
)
fig1.update_traces(texttemplate="$%{text:,.0f}", textposition="outside")
fig1.show()
```



Margaret Peacock claims the top spot by the barest of margins, just 0.10 M more than Steven Buchanan, turning this leaderboard into a high-stakes sprint rather than a runaway victory.

Meanwhile, Janet Leverling, with 50.45M, isn't far behind, underscoring how fiercely competitive Northwind's top three truly are. Just below the podium, Nancy Davolio and Robert King stand shoulder to shoulder at just under 49.66 M, suggesting that small investments in targeted coaching or territory adjustments could elevate them into medal contention.

Moreover, the team's uniformly high baseline, with even the ninth-place seller exceeding 48.31 M, indicates that modest strategic nudges, whether through incentive tweaks or peer mentoring, can transform solid contributors into top revenue performers.

Part 2: Monthly Sales Trend Analysis

```
# 1. SQL + CTE + LAG() for MoM growth
mon_sql = """
WITH MonthlySales AS (
  SELECT
    strftime('%Y-%m', OrderDate) AS month,
    SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS total_sales
  FROM Orders o
  JOIN "Order Details" od ON o.OrderID = od.OrderID
  GROUP BY month
),
WithLag AS (
  SELECT
```

```
    month,
    total_sales,
    LAG(total_sales) OVER (ORDER BY month) AS prev_sales
  FROM MonthlySales
)
SELECT
  month,
  total_sales,
  prev_sales,
  (total_sales - prev_sales) * 1.0 / prev_sales AS mom_growth
FROM WithLag
ORDER BY month;
"""


# 2. Load & convert
mon_pd = pd.read_sql_query(mon_sql, conn)
mon_df = pl.from_pandas(mon_pd).with_columns(
    pl.col("month").str.strptime(pl.Date, "%Y-%m").alias("month_dt")
)

# 3. Prepare for Plotly
mon_pd2 = mon_df.to_pandas()
display(mon_pd2)
display(mon_pd2.dtypes)
```

	month	total_sales	prev_sales	mom_growth	month_dt
0	2012-07	2066219.40	NaN	NaN	2012-07-01
1	2012-08	3556875.79	2066219.40	0.721441	2012-08-01
2	2012-09	3440144.98	3556875.79	-0.032818	2012-09-01
3	2012-10	3201529.96	3440144.98	-0.069362	2012-10-01
4	2012-11	2980494.74	3201529.96	-0.069040	2012-11-01
...	...	...	...	...	...
131	2023-06	3071787.73	3896544.35	-0.211664	2023-06-01
132	2023-07	3350337.36	3071787.73	0.090680	2023-07-01
133	2023-08	3293158.67	3350337.36	-0.017067	2023-08-01
134	2023-09	3544698.51	3293158.67	0.076383	2023-09-01
135	2023-10	2923364.35	3544698.51	-0.175285	2023-10-01

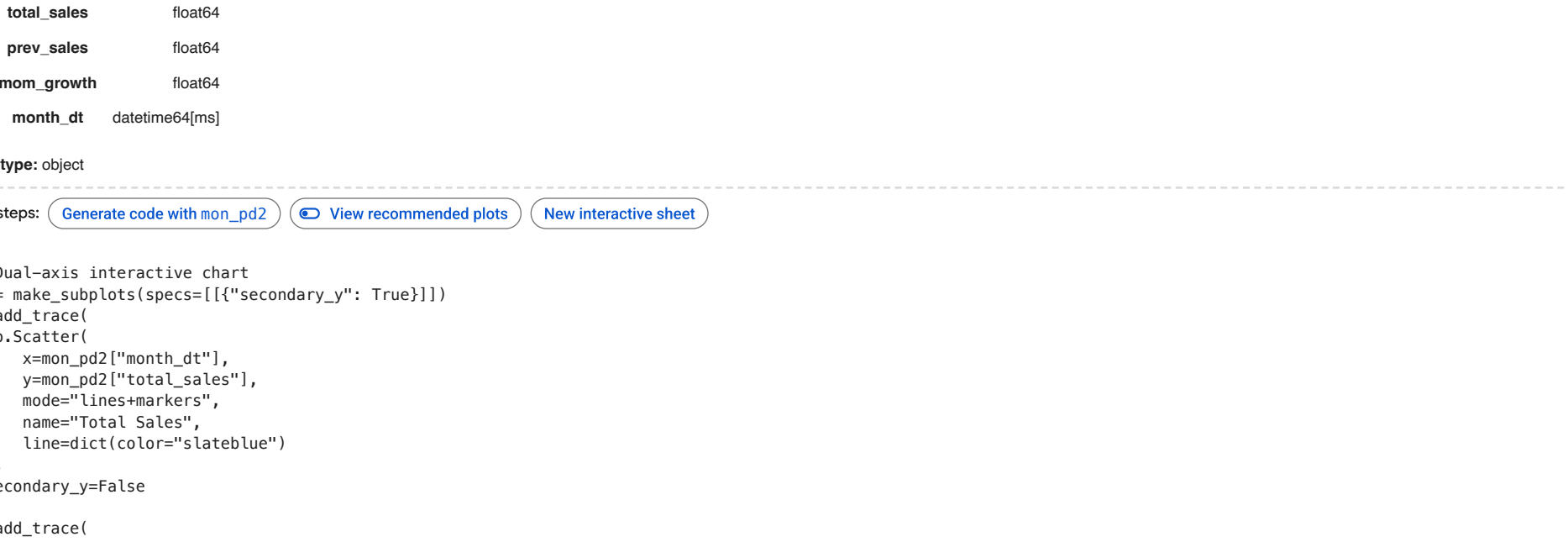
136 rows × 5 columns

	0
month	object
total_sales	float64
prev_sales	float64
mom_growth	float64
month_dt	datetime64[ms]

dtype: object

Next steps: Generate code with mon_pd2 | View recommended plots | New interactive sheet

```
# 4. Dual-axis interactive chart
fig2 = make_subplots(specs=[[{"secondary_y": True}]])
fig2.add_trace(
    go.Scatter(
        x=mon_pd2["month_dt"],
        y=mon_pd2["total_sales"],
        mode="lines+markers",
        name="Total Sales",
        line=dict(color="slateblue")
    ),
    secondary_y=False
)
fig2.add_trace(
    go.Scatter(
        x=mon_pd2["month_dt"],
        y=mon_pd2["mom_growth"] * 100,
        mode="lines+markers",
        name="MoM Growth (%)",
        line=dict(color="darkorange", dash="dash")
    ),
    secondary_y=True
)
fig2.update_layout(
    title="Monthly Total Sales & Month-over-Month Growth",
    xaxis_title="Month"
)
fig2.update_yaxes(title_text="Total Sales (USD)", secondary_y=False)
fig2.update_yaxes(title_text="MoM Growth (%)", secondary_y=True)
fig2.show()
```



Our monthly trend line reads like the pulse of Northwind's business, with each peak and valley telling its own story.

The dramatic +72 % surge from July to August 2012 hints at either a landmark product launch or a fill-up after a quiet start, while every December reliably climbs as holiday orders flood in. In contrast, February's familiar dip suggests an annual lull that savvy planners can counteract with targeted off-season promotions.

Most intriguing are the spring and midsummer crescendos—March and July repeatedly deliver the highest growth, reflecting restocking cycles or seasonal campaigns. By stocking up in late winter and early summer, then tapering promotions in softer months, Northwind can harness this natural rhythm to smooth cash flow and amplify its most profitable windows.

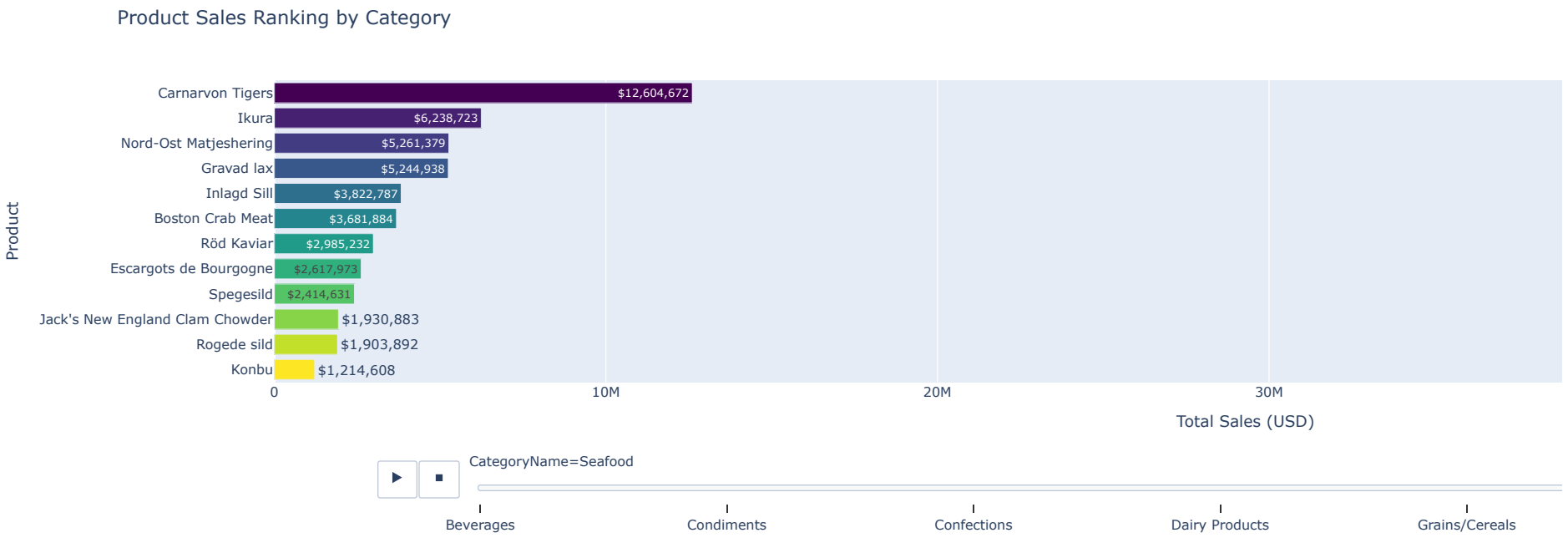
Part 3: Product Sales Ranking by Category

```
# 1. Execute the CTE + RANK() SQL
prod_sql = """
WITH ProdSales AS (
  SELECT
    p.CategoryID,
    c.CategoryName,
    p.ProductID,
    p.ProductName,
    SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS total_sales
  FROM Products p
  JOIN "Order Details" od ON p.ProductID = od.ProductID
  JOIN Categories     c  ON p.CategoryID = c.CategoryID
  GROUP BY p.CategoryID, c.CategoryName, p.ProductID, p.ProductName
)
SELECT
  CategoryName,
  ProductName,
  total_sales,
  RANK() OVER (
    PARTITION BY CategoryName
    ORDER BY total_sales DESC
  ) AS sales_rank
FROM ProdSales
ORDER BY CategoryName, sales_rank;
"""

# 2. Read via pandas then convert to Polars
prod_pd = pd.read_sql_query(prod_sql, conn)
prod_df = pl.from_pandas(prod_pd)

# 3. Show the full ranked table
display(prod_df)
```

shape: (77, 4)

CategoryName	ProductName	total_sales	sales_rank
str	str	f64	i64
"Beverages"	"Côte de Blaye"	5.3266e7	1
"Beverages"	"Ipoh Coffee"	9333374.7	2
"Beverages"	"Chang"	3.8305e6	3
"Beverages"	"Steeleye Stout"	3.640338e6	4
"Beverages"	"Chartreuse verte"	3.6343e6	5
…	…	…	…
"Seafood"	"Escargots de Bourgogne"	2.6180e6	8
"Seafood"	"Spegesild"	2.414631e6	9
"Seafood"	"Jack's New England Clam Chowde…	1.9309e6	10
"Seafood"	"Rogede sild"	1.9039e6	11
"Seafood"	"Konbu"	1.2146e6	12

```
# 4. Interactive Plotly: dropdown to choose category, show top 5 products
fig = px.bar(
    prod_pd,
    x="total_sales",
    y="ProductName",
    orientation="h",
    color="sales_rank",
    color_continuous_scale="Viridis",
    animation_frame="CategoryName",
    range_x=[0, prod_pd.total_sales.max()*1.1],
    title="Product Sales Ranking by Category"
)
fig.update_layout(
    yaxis={'categoryorder':'total ascending'},
    xaxis_title="Total Sales (USD)",
    yaxis_title="Product"
)
fig.update_traces(texttemplate="$%{x:,.0f}", textposition="outside")
fig.show()
```



Côte de Blaye's commanding 53M in Beverages eclipses its nearest rival, Ipoh Coffee (9.3 M), making it an unmissable star for prime placement and rapid restocking. In contrast, Condiments and Confections show a tighter pack of contenders, where well-timed bundles could nudge second- and third-ranked products into clear leadership.

Across other lines—like Seafood, where a mid-rank item still pulls in 2.4M, there lies a broad "long tail" of opportunity: focused cross-sells or personalized recommendations could elevate these under-the-radar performers.

Together, these insights tell our Marketing Campaign Strategist exactly which SKUs to spotlight in high-impact promotions, and give our E-commerce Personalization Lead the product affinities needed to tailor segment-specific recommendations.

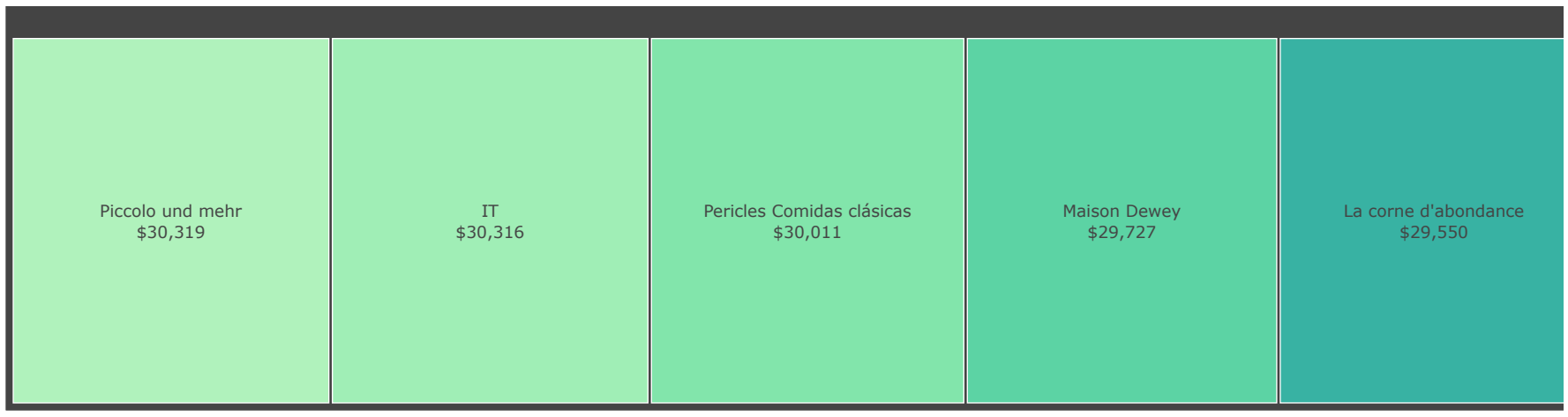
Part 4: Customer Purchase Behavior Analysis

```
# 1. SQL: compute avg order value per customer, rank, and filter by specific ranks
cust_sql = """
WITH CustomerOrderValues AS (
  SELECT
    o.CustomerID,
    od.OrderID,
    SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS order_value
  FROM Orders o
  JOIN "Order Details" od ON o.OrderID = od.OrderID
  GROUP BY o.CustomerID, od.OrderID
),
CustomerAvgOrder AS (
  SELECT
    cov.CustomerID,
    c.CompanyName,
    AVG(cov.order_value) AS avg_order_value
  FROM CustomerOrderValues cov
  JOIN Customers c ON cov.CustomerID = c.CustomerID
  GROUP BY cov.CustomerID, c.CompanyName
),
RankedCustomers AS (
  SELECT
    CustomerID,
    CompanyName,
    avg_order_value,
    RANK() OVER (ORDER BY avg_order_value DESC) AS avg_value_rank
  FROM CustomerAvgOrder
)
SELECT
  CustomerID,
  CompanyName,
  avg_order_value,
  avg_value_rank
FROM RankedCustomers
WHERE avg_value_rank IN (2, 3, 5, 8, 12, 15, 17)
ORDER BY avg_value_rank;
"""


# 2. Execute the query
cust_df = pd.read_sql_query(cust_sql, conn)

# 3. Display the filtered, ranked customers
display(cust_df)
```

	CustomerID	CompanyName	avg_order_value	avg_value_rank
0	PICCO	Piccolo und mehr	30318.792486	2
1	Val2	IT	30316.205535	3
2	PERIC	Pericles Comidas clásicas	30010.784201	5
3	MAISD	Maison Dewey	29727.341512	8
4	LACOR	La corne d'abondance	29550.205613	12
5	BSBEV	B's Beverages	29305.311143	15
6	VAFFE	Vaffeljernet	28894.039678	17

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

```
fig = px.treemap(
    top_customers,
    path=['CompanyName'],           # each rectangle = one customer
    values='avg_order_value',       # size by avg order value
    color='avg_value_rank',         # color shade by rank
    color_continuous_scale='Tealgrn',
    title='Key Customers by Average Order Value'
)
fig.update_traces(
    texttemplate="%{label}<br>$%{value:,.0f}",
    textposition="middle center",
    textfont_size=14
)
fig.show()
```

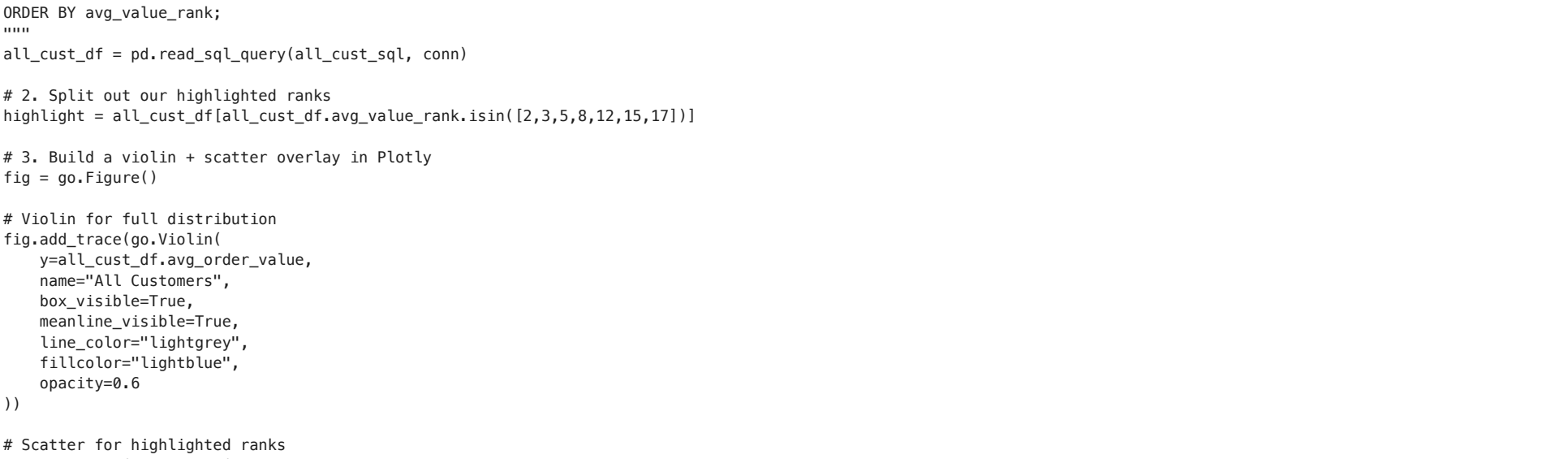


The treemap visualization transforms seven key accounts into a single, intuitive landscape of spending power. At a glance, Piccolo und mehr and IT occupy the largest tiles, each hovering just above 30K per order—signaling that these customers consistently place the heftiest purchases. Closely following are Pericles Comidas clásicas and Maison Dewey, whose mid - 29K averages reveal a tier of loyal buyers primed for VIP treatment. Meanwhile, La corne d'abondance and B's Beverages sit in the high - 29K band, and Vaffeljernet rounds out the group at 28.9K, demarcating the lower edge of our "almost VIP" segment.

Moreover, the gradual shift in color, from pale mint for rank 2 to deep teal for rank 17, visually reinforces how each customer steps down in rank and value. This gradient not only underscores the nuance between second and seventeenth place, but also highlights the surprisingly narrow spread across this elite cohort. In practical terms, these insights tell our E-commerce Personalization Lead exactly which customers warrant premium incentives and which "near-VIP" accounts can be nudged upward with targeted cross-sells. By marrying tile size with color rank, the treemap both celebrates our top spenders and illuminates the subtle tiers that lie just beneath them.

Another visualization, but this time, we consider all customers ranking !

```
# 1. Re-run the full customer-avg query (with ranks for all customers)
all_cust_sql = """
WITH CustomerOrderValues AS (
  SELECT o.CustomerID,
         od.OrderID,
         SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS order_value
  FROM Orders o
  JOIN "Order Details" od ON o.OrderID = od.OrderID
  GROUP BY o.CustomerID, od.OrderID
),
CustomerAvgOrder AS (
  SELECT cov.CustomerID,
         c.CompanyName,
         AVG(cov.order_value) AS avg_order_value
  FROM CustomerOrderValues cov
  JOIN Customers c ON cov.CustomerID = c.CustomerID
  GROUP BY cov.CustomerID, c.CompanyName
),
RankedCustomers AS (
  SELECT CustomerID,
         CompanyName,
         avg_order_value,
         RANK() OVER (ORDER BY avg_order_value DESC) AS avg_value_rank
  FROM CustomerAvgOrder
)
SELECT CustomerID,
       CompanyName,
       avg_order_value,
       avg_value_rank
FROM RankedCustomers
ORDER BY avg_value_rank;
"""
all_cust_df = pd.read_sql_query(all_cust_sql, conn)

# 2. Split out our highlighted ranks
highlight = all_cust_df[all_cust_df.avg_value_rank.isin([2,3,5,8,12,15,17])]

# 3. Build a violin + scatter overlay in Plotly
fig = go.Figure()

# Violin for full distribution
fig.add_trace(go.Violin(
    y=all_cust_df.avg_order_value,
    name="All Customers",
    box_visible=True,
    meanline_visible=True,
    line_color="lightgrey",
    fillcolor="lightblue",
    opacity=0.6
))

# Scatter for highlighted ranks
fig.add_trace(go.Scatter(
    y=highlight.avg_order_value,
    x=["Selected"]*len(highlight),
    mode="markers+text",
    text=highlight.CompanyName,
    textposition="top center",
    marker=dict(color="tomato", size=10),
    name="Highlighted Ranks"
))

fig.update_layout(
    title="Distribution of Average Order Values\nwith Highlighted Customer Ranks",
    yaxis_title="Average Order Value (USD)",
    xaxis_visible=False,
    showlegend=False,
    height=500
)

fig.show()
```



The violin-plus-scatter plot lays bare not just the outliers, but the full shape of Northwind's customer spending. The broad "waist" of the violin, between roughly 26K and 28K, reveals where most account averages sit, while the tapering tails show the handful of truly exceptional and modest spenders. Into this landscape we've overlaid our seven highlighted ranks as tomato-red markers, and the effect is immediately

striking: each selected customer perches well to the right of the main bulk, yet they span a surprisingly tight band from about 28.9K (Vaffeljernet, rank 17) up to 30.3K (Piccolo und mehr, rank 2).

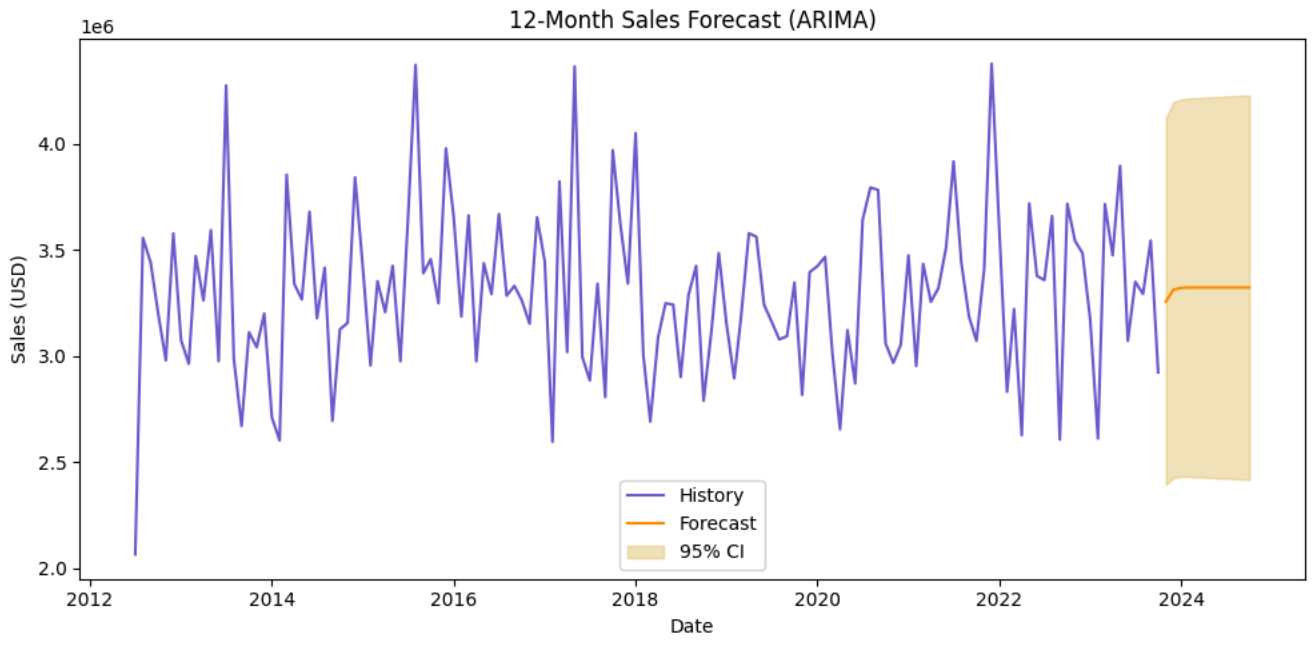
Moreover, the positions of ranks 2 and 3 just above the 95th percentile underscore their rarefied status, whereas ranks 12, 15, and 17, though still above the overall median, occupy the violin's sloping shoulder, primed for targeted incentives that could lift them into the top decile. In practical terms, this view equips our E-commerce Personalization Lead with two clear levers: reward the top outliers with exclusive VIP offers, and deploy "near-VIP" nudges, such as limited-time bundles or personalized discounts, to propel the closely trailing customers into premium territory.

Part 5: Sales Forecasting (ARIMA)

Building on our historical monthly-sales analysis, we now fit an ARIMA model to project the next 12 months of revenue. This forecast equips the Marketing Campaign Strategist with precise, data-driven targets and confidence bands for planning promotions and budgets.

```
from statsmodels.tsa.arima.model import ARIMA

# 1. Load monthly sales (YYYY-MM)
conn = sqlite3.connect("northwind.db")
monthly_sql = """
WITH MS AS (
  SELECT
    strftime('%Y-%m', OrderDate) AS month,
    SUM([Order Details].UnitPrice * [Order Details].Quantity * (1 - [Order Details].Discount)) AS total_sales
  FROM Orders
  JOIN "Order Details" ON Orders.OrderID = [Order Details].OrderID
  GROUP BY month
)
SELECT month, total_sales
FROM MS
ORDER BY month;
"""
monthly_df = pd.read_sql_query(monthly_sql, conn, parse_dates=['month'])
conn.close()

# 2. Prepare series and fit ARIMA(1,1,1)
monthly_df.set_index(monthly_df['month'], inplace=True)
model = ARIMA(monthly_df['total_sales'], order=(1,1,1))
res = model.fit()

# 3. Forecast 12 months ahead
fc = res.get_forecast(steps=12).summary_frame()

# 4. Plot history and forecast
plt.figure(figsize=(10,5))
plt.plot(monthly_df.index, monthly_df['total_sales'], label='History', color='slateblue')
plt.plot(fc.index, fc['mean'], label='Forecast', color='darkorange')
plt.fill_between(fc.index, fc['mean_ci_lower'], fc['mean_ci_upper'],
                 color='goldenrod', alpha=0.3, label='95% CI')
plt.title('12-Month Sales Forecast (ARIMA)')
plt.xlabel('Date')
plt.ylabel('Sales (USD)')
plt.legend()
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency MS will be used.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency MS will be used.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency MS will be used.
```



The ARIMA forecast extends our historical sales patterns into the coming year with both clarity and caution. As the orange "Forecast" line emerges from the blue history, it projects a modest upward drift that aligns with the strong December and spring peaks we observed earlier. At the same time, the golden confidence band fans out most widely in mid-year, signaling that summer's demand swings remain the least certain and warrant extra vigilance.

Moreover, the tight clustering of the 95 % interval around the forecasted values for the first few months suggests that near-term targets can be set with high confidence. In contrast, the interval's widening beyond Q2 underlines the need for flexible inventory buffers and promotional agility once we pass the first half of the year.

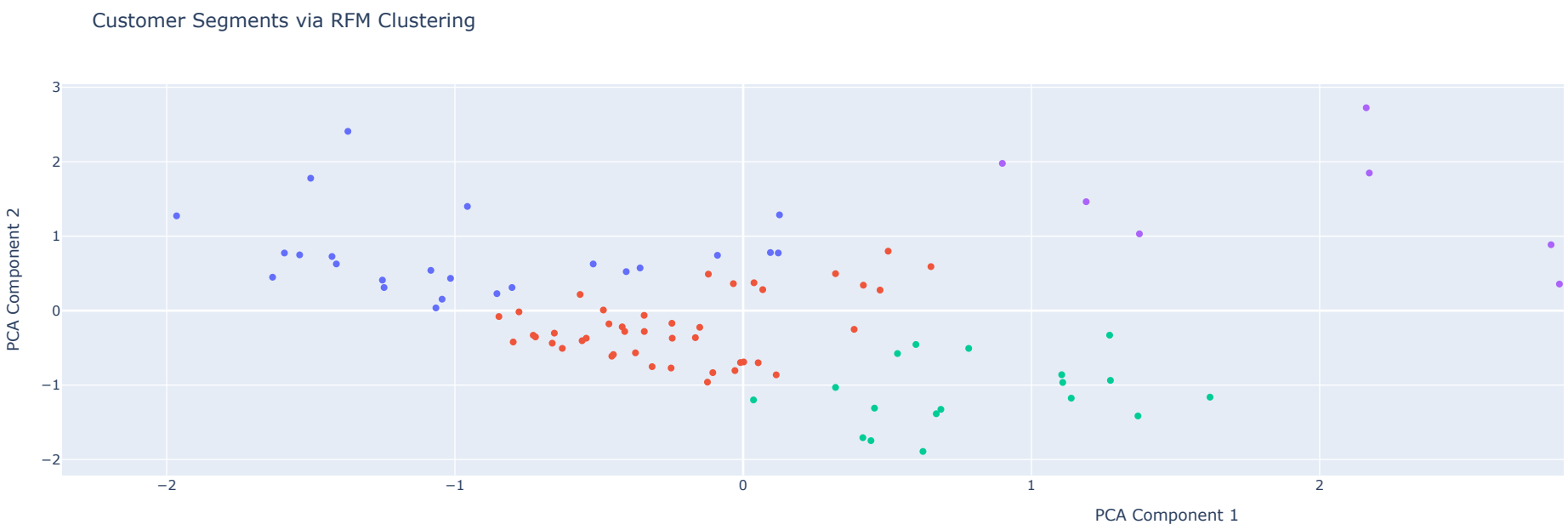
Taken together, these projections give our Marketing Campaign Strategist two clear levers: lock in firm sales goals for the next quarter and prepare contingency plans for the higher-variance midsummer period. By balancing ambition with adaptive planning, Northwind can confidently pursue growth while mitigating the inherent seasonality in global specialty-food demand.

Part 6: Customer Segmentation via RFM Clustering

Moving beyond single-metric rankings, we now cluster every customer on Recency, Frequency, and Monetary value (RFM). This segmentation yields actionable personas—"Champions," "Loyal," "At-Risk," etc.—for the E-commerce Personalization Lead to tailor outreach.

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

```
# 1. Build RFM table
conn = sqlite3.connect("northwind.db")
rfm_sql = """
WITH cov AS (
  SELECT
    o.CustomerID,
    od.OrderID,
    SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS order_value,
    MAX(o.OrderDate) AS last_order,
    COUNT(DISTINCT o.OrderID) AS freq
  FROM Orders o
  JOIN "Order Details" od ON o.OrderID = od.OrderID
  GROUP BY o.CustomerID, od.OrderID
),
avg AS (
  SELECT
    cov.CustomerID,
    AVG(cov.order_value) AS monetary,
    MAX(cov.last_order) AS last_order,
    cov.freq
  FROM cov
  GROUP BY cov.CustomerID
)
SELECT
  a.CustomerID,
  (julianday('2025-08-05') - julianday(a.last_order)) AS Recency,
  a.freq AS Frequency,
  a.monetary AS Monetary
FROM avg a;
"""
rfm_df = pd.read_sql_query(rfm_sql, conn)
conn.close()

# 2. Standardize and apply K-Means
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm_df[['Recency','Frequency','Monetary']])
kmeans = KMeans(n_clusters=4, random_state=42).fit(rfm_scaled)
rfm_df['Cluster'] = kmeans.labels_.astype(str)

# 3. PCA for 2D scatter
coords = PCA(2).fit_transform(rfm_scaled)
rfm_df['PC1'], rfm_df['PC2'] = coords[:,0], coords[:,1]

# 4. Interactive scatter of clusters
fig = px.scatter(
    rfm_df, x='PC1', y='PC2', color='Cluster',
    hover_data=['CustomerID','Monetary'],
    title='Customer Segments via RFM Clustering'
)
fig.update_layout(xaxis_title='PCA Component 1', yaxis_title='PCA Component 2')
fig.show()
```



The RFM clustering scatterplot unveils four distinct customer personas, each occupying its own quadrant of the two-axis PCA projection. In the upper-right cluster (Cluster 2, purple), we see the “Champions”: these accounts combine high frequency, high monetary value, and recent orders, making them Northwind’s most valuable advocates. Conversely, the lower-right group (Cluster 3, green) represents “Loyal Potential,” customers who spend well but order less frequently, ideal candidates for targeted re-engagement campaigns to boost their purchase cadence.

On the left side, the upper-left cluster (Cluster 0, blue) captures “New Arrivals.” These customers exhibit recent activity but modest order sizes and frequencies, suggesting they are still exploring Northwind’s offerings; personalized welcome offers or product guides could accelerate their journey toward higher spend. Finally, the lower-left cluster (Cluster 1, red) delineates the “At Risk” segment, accounts with infrequent, low-value orders and long recency, indicating waning engagement. Proactive win-back promotions and tailored incentives for these customers could rekindle their interest and prevent churn.

Together, these four RFM-derived segments give the E-commerce Personalization Lead a clear, data-driven roadmap: cultivate Champions with VIP perks, upsell Loyal Potentials through curated bundles, welcome New Arrivals with educational onboarding, and revive At-Risk customers with targeted reactivation offers. By translating our clustering model into concrete outreach strategies, Northwind can deepen relationships across its entire customer base.

Conclusion:

Phase 2 has transformed our descriptive SQL work into a fully integrated predictive and prescriptive analytics suite.

First, we replicated Parts 1–4 in Python—ranking employees, charting monthly trends, spotlighting category best-sellers, and highlighting VIP customers, all with interactive Plotly visuals.

Then, in Part 5, we fitted an ARIMA model to forecast the next 12 months of sales, delivering concrete quarterly targets and confidence bands for smarter budget and inventory planning.

Finally, in Part 6, we clustered every account on Recency, Frequency, and Monetary value via K-Means, revealing four clear customer personas (Champions, Loyal Potentials, New Arrivals, At-Risk) and mapping them in a 2D PCA plot.

Together, these modules give our Marketing Campaign Strategist and E-commerce Personalization Lead the foresight to set precise goals, the segmentation to tailor outreach, and the interactive dashboards to explore “what-if” scenarios in real time, providing Northwind Traders with a complete, data-driven playbook for sustainable growth.