Automated Data Lakehouse Pipeline on Databricks (Delta Lake + Spark SQL + Airflow + AWS S3)

Automated Dependency Setup for Data Science & ML Pipelines

```
get_ipython().system('apt-get update -qq && apt-get install -y -qq python3-venv libgl1-mesa-glx')
import os
os.environ["DJANGO_HOME"] = "/usr/lib/python3.11/site-packages"

get_ipython().system('pip install qasync pyqt6 pyqt6-tools delta-spark pandas matplotlib plotly seaborn scikit-learn xgboost opencv-python-headless')
```

Show hidden output

SparkSession Initialization with Delta Lake Extensions for Lakehouse Architecture

```
!apt-get install openjdk-11-jdk -y
!pip install pyspark==3.4.1 delta-spark==2.4.0
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"

from pyspark.sql import SparkSession
from delta import configure_spark_with_delta_pip
builder = SparkSession.builder \
    .appName("DeltaLakeSession") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog") \
    .config("spark.databricks.delta.catalog.enabled", "true") \
    .config("spark.sql.shuffle.partitions", "4") \
    .config("spark.driver.memory", "4g")

spark = configure_spark_with_delta_pip(builder).getOrCreate()
print(" Spark session created:", spark.version)
```

Show hidden output

Multi-Source Data Preparation for Scalable Analytics Workflows

```
!wget -q -O /content/OnlineRetail.xlsx "https://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online%20Retail.xlsx"
import os
RAW_PATH = "/content/datalake/raw"
os.makedirs(RAW_PATH, exist_ok=True)

import pandas as pd
or_xl = pd.read_excel('/content/OnlineRetail.xlsx')
```

```
or_xl.to_csv('/content/online_retail.csv', index=False)
!mv /content/online_retail.csv {RAW_PATH}/online_retail.csv
import numpy as np
import datetime
np.random.seed(42)

n = 50000
start = datetime.datetime(2020, 1, 1)
times = [start + datetime.timedelta(seconds=int(x)) for x in np.cumsum(np.random.exponential(scale=30, size=n))]
user_ids = np.random.randint(1, 5000, size=n)
pages = np.random.choice(['home', 'product', 'search', 'cart', 'checkout'], size=n, p=[0.4, 0.3, 0.15, 0.1, 0.05])

clicks = pd.DataFrame({'event_time': times, 'user_id': user_ids, 'page': pages})
clicks.to_csv(f'{RAW_PATH}/clickstream.csv', index=False)

print('✅ Datasets ready in', RAW_PATH)
```

```
✅ Datasets ready in /content/datalake/raw
```

Delta Lake Table Creation from Raw Retail and Clickstream Sources

```
online_raw = spark.read.option('header',True).option('inferSchema',True).csv(f'{RAW_PATH}/online_retail.csv')
clicks_raw = spark.read.option('header',True).option('inferSchema',True).csv(f'{RAW_PATH}/clickstream.csv')


online_raw.write.format('delta').mode('overwrite').save(f'{RAW_PATH}/online_retail.delta')
clicks_raw.write.format('delta').mode('overwrite').save(f'{RAW_PATH}/clickstream.delta')


print('Raw Delta tables saved')
```

```
Raw Delta tables saved
```

Source Data Validation: Timestamped Retail Transactions with Field-Level Diagnostics

```
online_raw.printSchema()
online_raw.show(5)
```

```
root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: timestamp (nullable = true)
 |-- UnitPrice: double (nullable = true)
 |-- CustomerID: double (nullable = true)
 |-- Country: string (nullable = true)

+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+
|InvoiceNo|StockCode|         Description|Quantity|        InvoiceDate|UnitPrice|CustomerID|       Country|
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+
|   536365|   85123A|WHITE HANGING HEA...|       6|2010-12-01 08:26:00|     2.55|   17850.0|United Kingdom|
|   536365|    71053| WHITE METAL LANTERN|       6|2010-12-01 08:26:00|     3.39|   17850.0|United Kingdom|
|   536365|   84406B|CREAM CUPID HEART...|       8|2010-12-01 08:26:00|     2.75|   17850.0|United Kingdom|
|   536365|   84029G|KNITTED UNION FLA...|       6|2010-12-01 08:26:00|     3.39|   17850.0|United Kingdom|
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|2010-12-01 08:26:00|     3.39|   17850.0|United Kingdom|
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+
only showing top 5 rows
```

Transactional Data Curation Workflow using Delta Format

```
from pyspark.sql.functions import to_timestamp, col, when, concat_ws, lit, year, month, dayofmonth, hour
import os

RAW_PATH = "/content/datalake/raw"
CLEAN_PATH = "/content/datalake/clean"
CURATED_PATH = "/content/datalake/curated"

for path in [RAW_PATH, CLEAN_PATH, CURATED_PATH]:
    os.makedirs(path, exist_ok=True)
online = spark.read.format('delta').load(f'{RAW_PATH}/online_retail.delta')
online_clean = online.filter(col('InvoiceNo').isNotNull()) \
    .withColumn('InvoiceDateTS', to_timestamp(col('InvoiceDate'))) \
    .withColumn('Quantity', col('Quantity').cast('int')) \
    .withColumn('UnitPrice', col('UnitPrice').cast('double')) \
    .filter(col('Quantity') > 0)

online_curated = online_clean.select(
    'InvoiceNo', 'StockCode', 'Description', 'Quantity',
    'UnitPrice', 'CustomerID', 'Country', 'InvoiceDateTS'
)
online_clean.write.format('delta').mode('overwrite').save(f'{CLEAN_PATH}/online_retail_clean.delta')
online_curated.write.format('delta').mode('overwrite').save(f'{CURATED_PATH}/online_retail_curated.delta')
clicks = spark.read.format('delta').load(f'{RAW_PATH}/clickstream.delta')
clicks = clicks.withColumn('event_time_ts', to_timestamp(col('event_time')))
clicks.write.format('delta').mode('overwrite').save(f'{CURATED_PATH}/clickstream_curated.delta')
print('✅ Cleaned and curated datasets saved.')
```

```
✅ Cleaned and curated datasets saved.
```

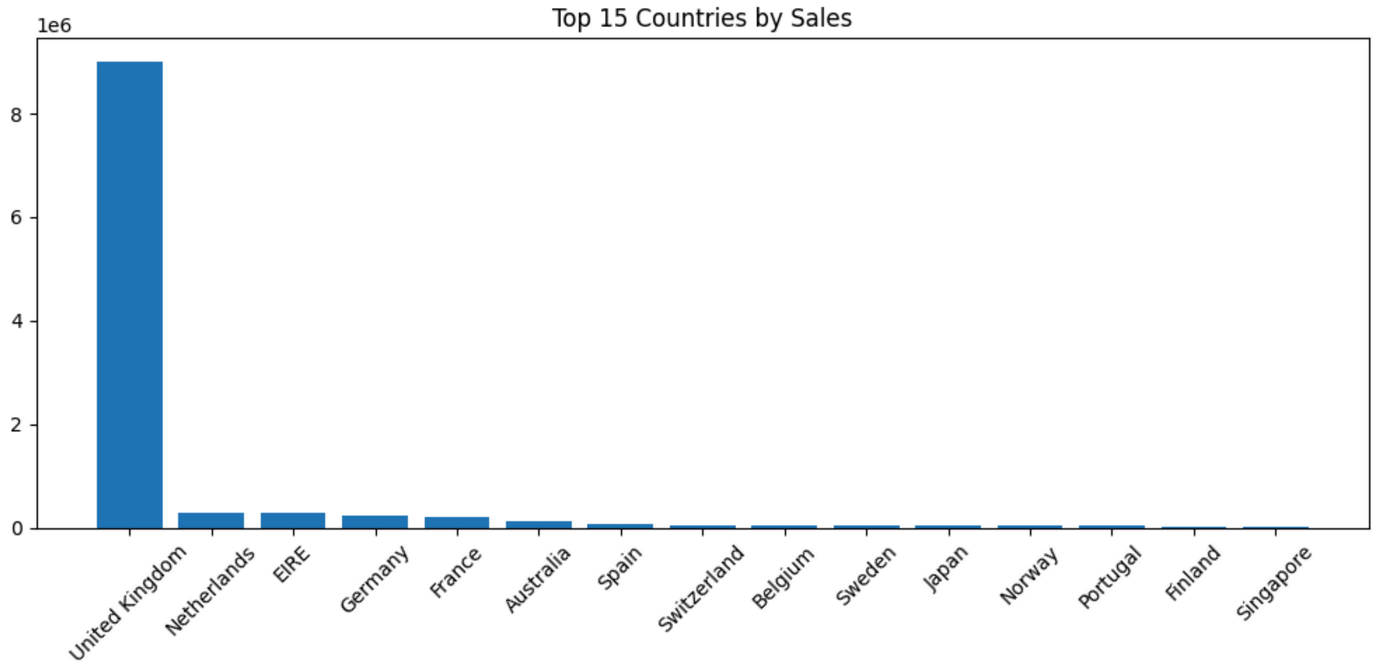
Transactional Data Curation Workflow using Delta Format

```
online_df = spark.read.format('delta').load(f'{CURATED_PATH}/online_retail_curated.delta')
clicks_df = spark.read.format('delta').load(f'{CURATED_PATH}/clickstream_curated.delta')
online_df.createOrReplaceTempView('online')
clicks_df.createOrReplaceTempView('clicks')
```

Sales Performance by Geography: Top 15 Countries

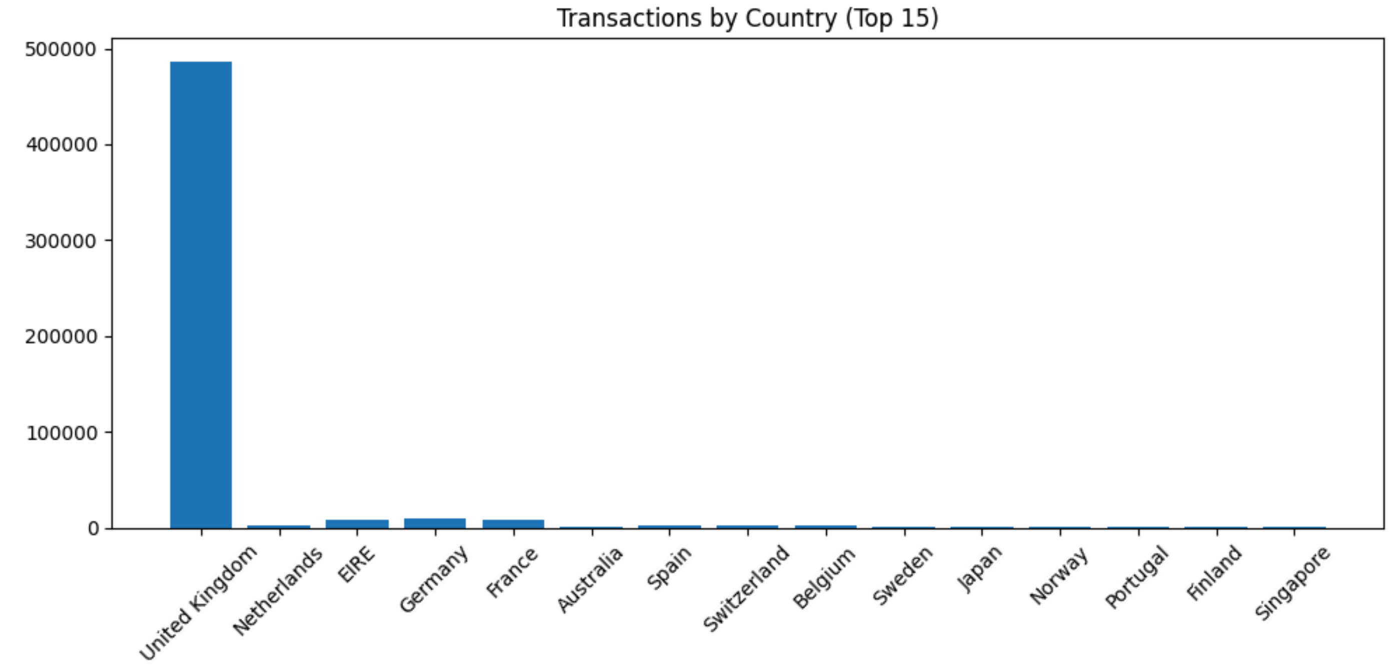
```
q = """
SELECT Country, sum(Quantity*UnitPrice) as total_sales, count(*) as txns
FROM online
GROUP BY Country
ORDER BY total_sales DESC
LIMIT 15
```

```
"""
pdf1 = spark.sql(q).toPandas()


import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
plt.bar(pdf1['Country'], pdf1['total_sales'])
plt.xticks(rotation=45)
plt.title('Top 15 Countries by Sales')
plt.tight_layout()
plt.show()
```



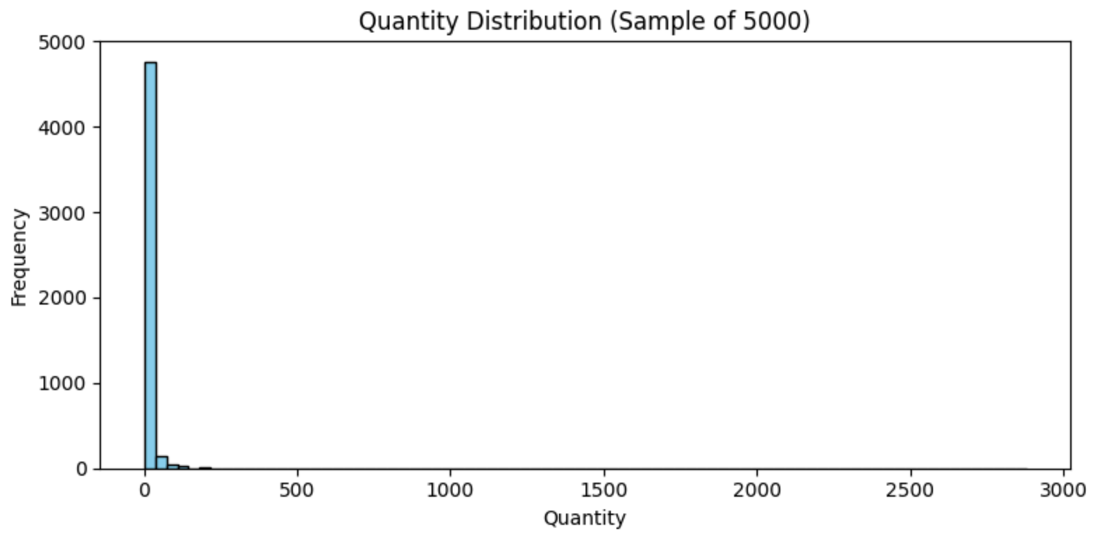
Top 15 Countries by Retail Transaction Count

```
plt.figure(figsize=(10,5))
plt.bar(pdf1['Country'], pdf1['txns'])
plt.xticks(rotation=45)
plt.title('Transactions by Country (Top 15)')
plt.tight_layout()
plt.show()
```



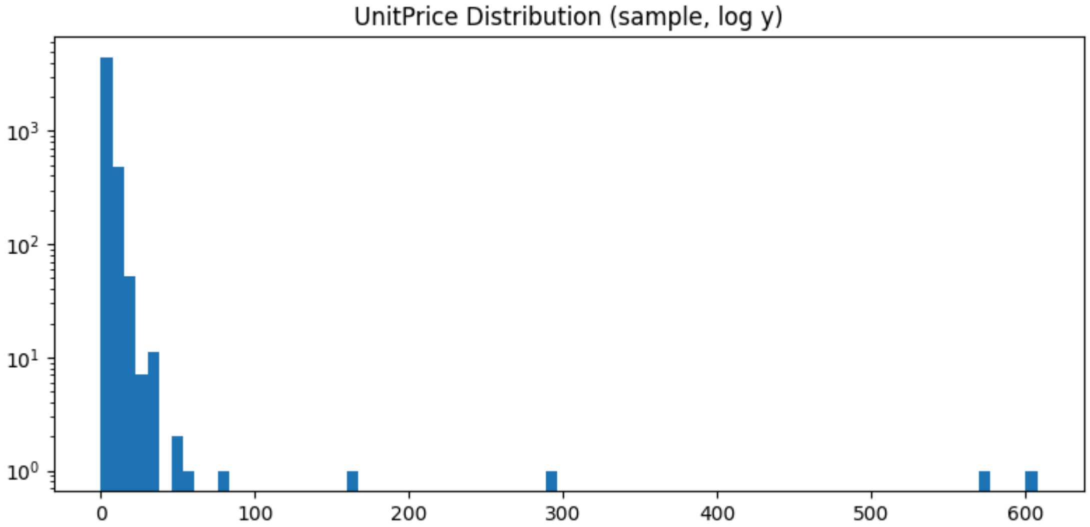
Quantity distribution (hist)

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

numeric_cols = [c for c, dtype in online_df.dtypes if dtype in ('int', 'bigint', 'double')]
online_df_numeric = online_df.select(*numeric_cols)
sample = online_df_numeric.limit(5000).toPandas()

plt.figure(figsize=(8, 4))
plt.hist(sample["Quantity"].dropna(), bins=80, color="skyblue", edgecolor="black")
plt.title("Quantity Distribution (Sample of 5000)")
plt.xlabel("Quantity")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



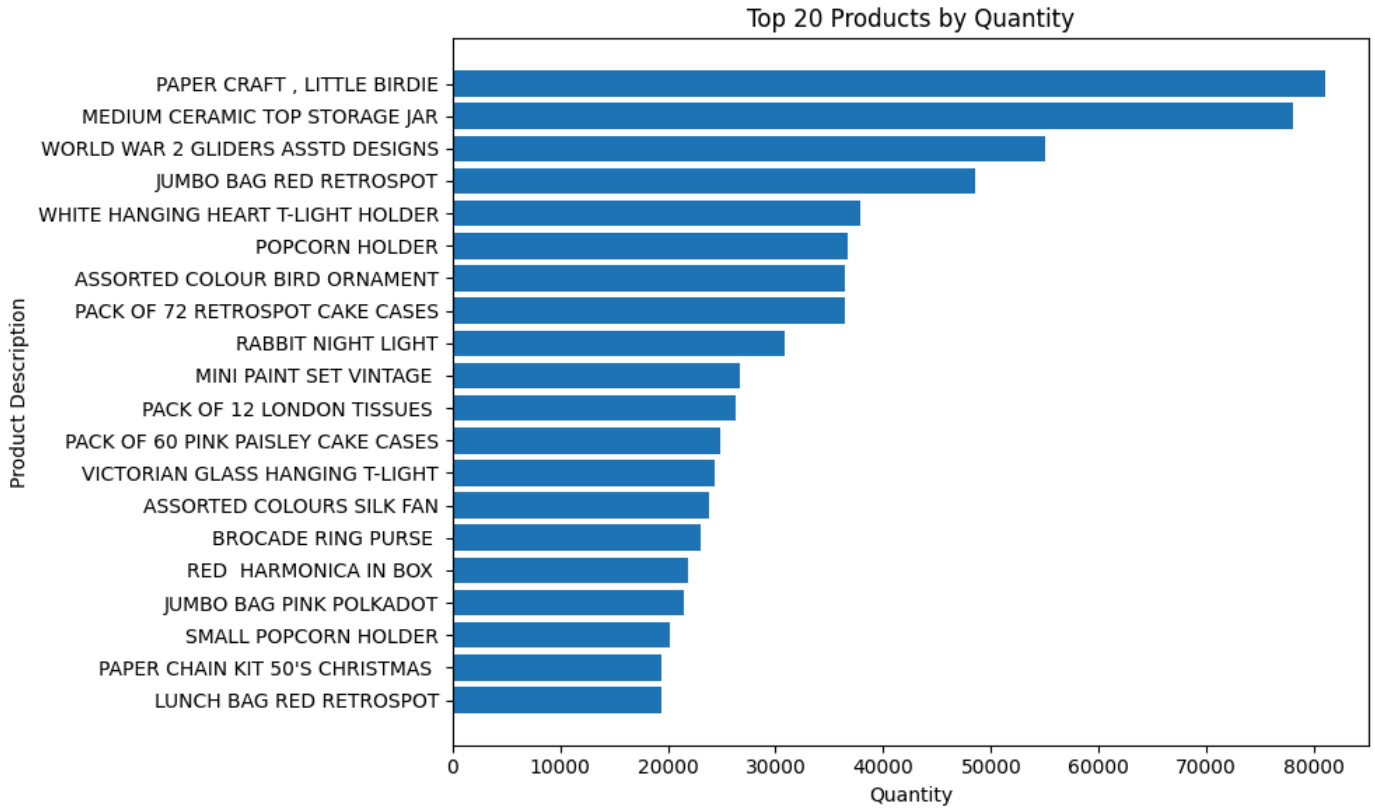
UnitPrice distribution (log scale)

```
plt.figure(figsize=(8,4))
plt.hist(sample['UnitPrice'].replace([float('inf'),-float('inf')],float('nan')).dropna(), bins=80)
plt.yscale('log')
plt.title('UnitPrice Distribution (sample, log y)')
plt.tight_layout()
plt.show()
```



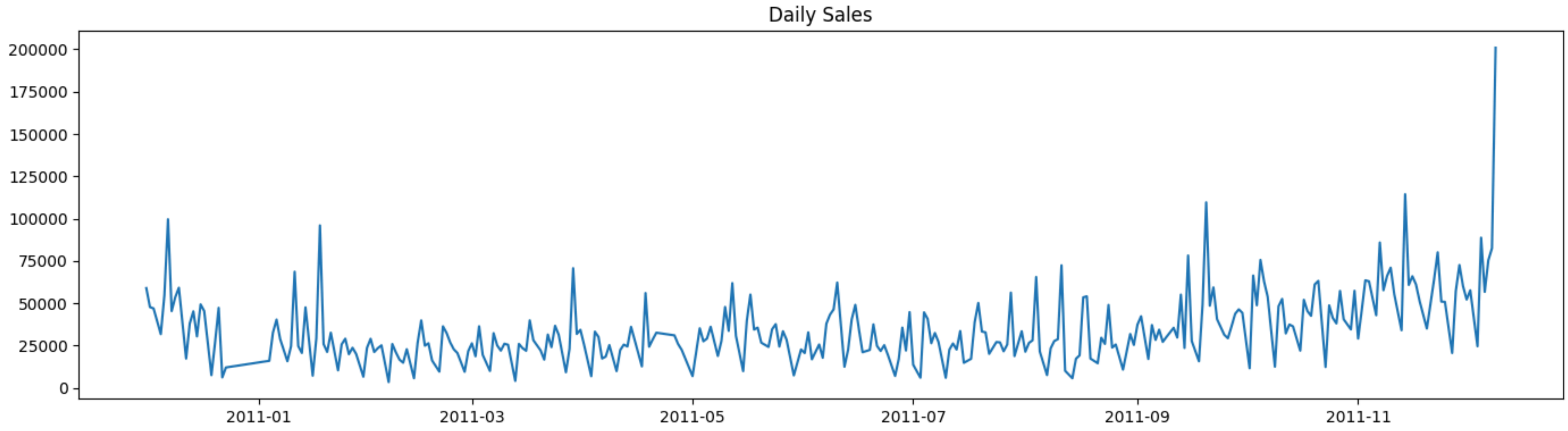
Top 20 products by quantity (horizontal bar)

```
q2 = """
SELECT Description, SUM(Quantity) AS qty
FROM online
WHERE Description IS NOT NULL
GROUP BY Description
ORDER BY qty DESC
LIMIT 20
"""
pdf2 = spark.sql(q2).toPandas()
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.barh(pdf2['Description'][::-1], pdf2['qty'][::-1])
plt.title('Top 20 Products by Quantity')
plt.xlabel('Quantity')
plt.ylabel('Product Description')
plt.tight_layout()
plt.show()
```



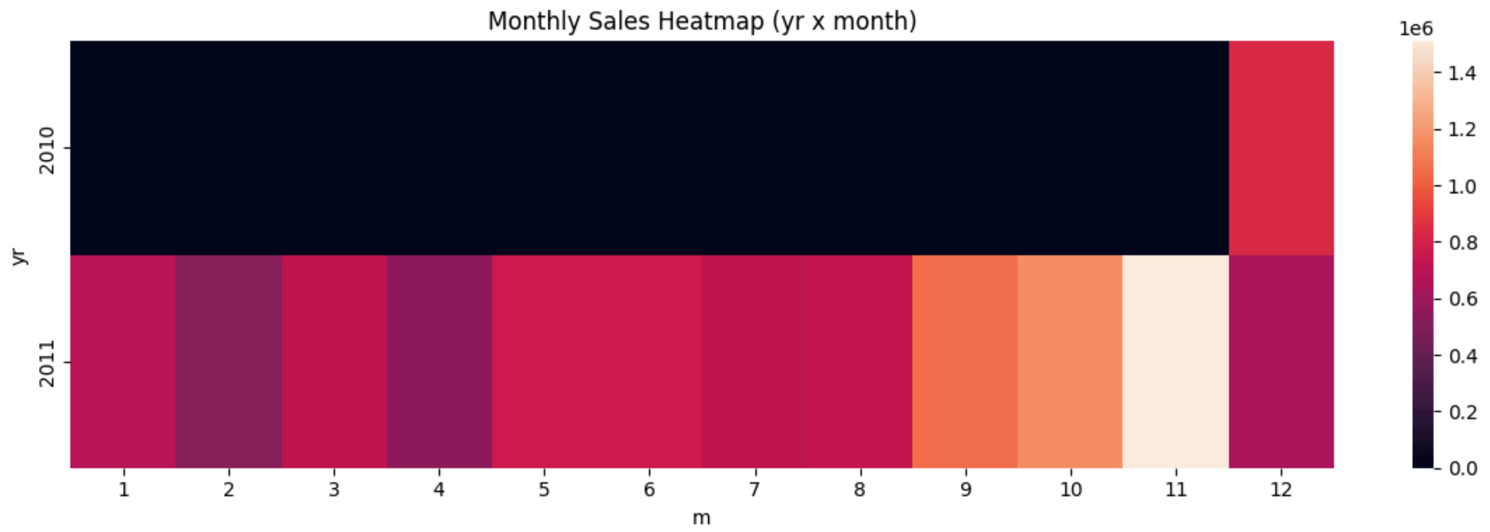
Sales over time (daily)

```
q3 = "SELECT to_date(InvoiceDateTS) as day, sum(Quantity*UnitPrice) as sales FROM online GROUP BY day ORDER BY day"
daily = spark.sql(q3).toPandas()
plt.figure(figsize=(14,4))
plt.plot(daily['day'], daily['sales'])
plt.title('Daily Sales')
plt.tight_layout()
plt.show()
```



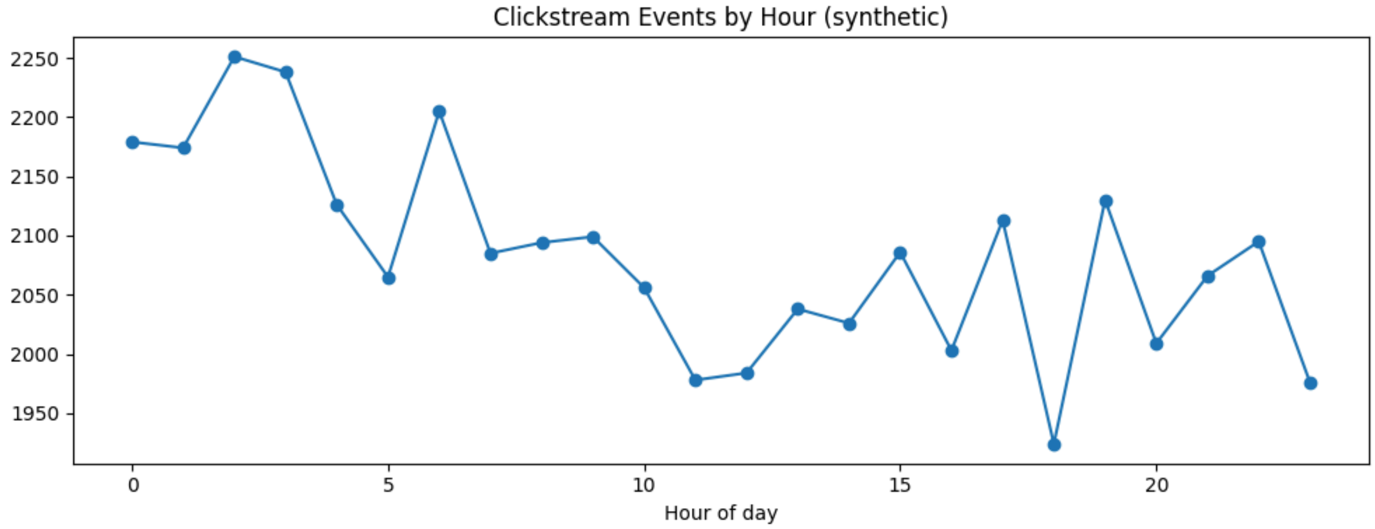
Monthly sales heatmap (year-month)

```
q4 = "SELECT year(InvoiceDateTS) as yr, month(InvoiceDateTS) as m, sum(Quantity*UnitPrice) as sales FROM online GROUP BY yr,m ORDER BY yr,m"
monthly = spark.sql(q4).toPandas()
monthly_pivot = monthly.pivot(index='yr', columns='m', values='sales').fillna(0)
plt.figure(figsize=(12,4))
import seaborn as sns
sns.heatmap(monthly_pivot, annot=False)
plt.title('Monthly Sales Heatmap (yr x month)')
plt.tight_layout()
plt.show()
```



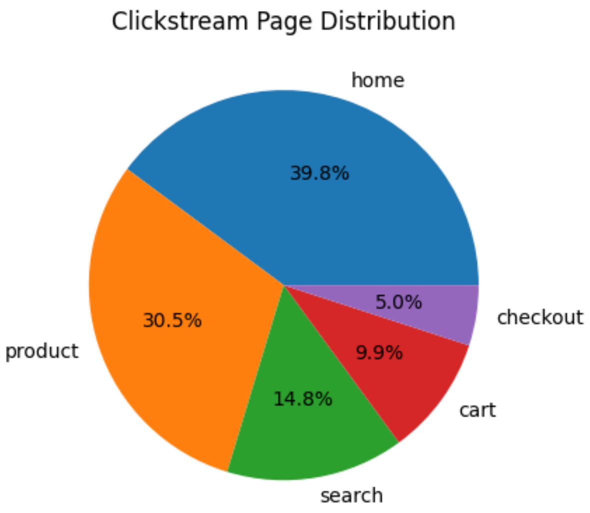
Hourly clickstream volume (synthetic clickstream)

```
q5 = "SELECT hour(event_time_ts) as hr, count(*) as cnt FROM clicks GROUP BY hr ORDER BY hr"
hours = spark.sql(q5).toPandas()
plt.figure(figsize=(10,4))
plt.plot(hours['hr'], hours['cnt'], marker='o')
plt.title('Clickstream Events by Hour (synthetic)')
plt.xlabel('Hour of day')
plt.tight_layout()
plt.show()
```



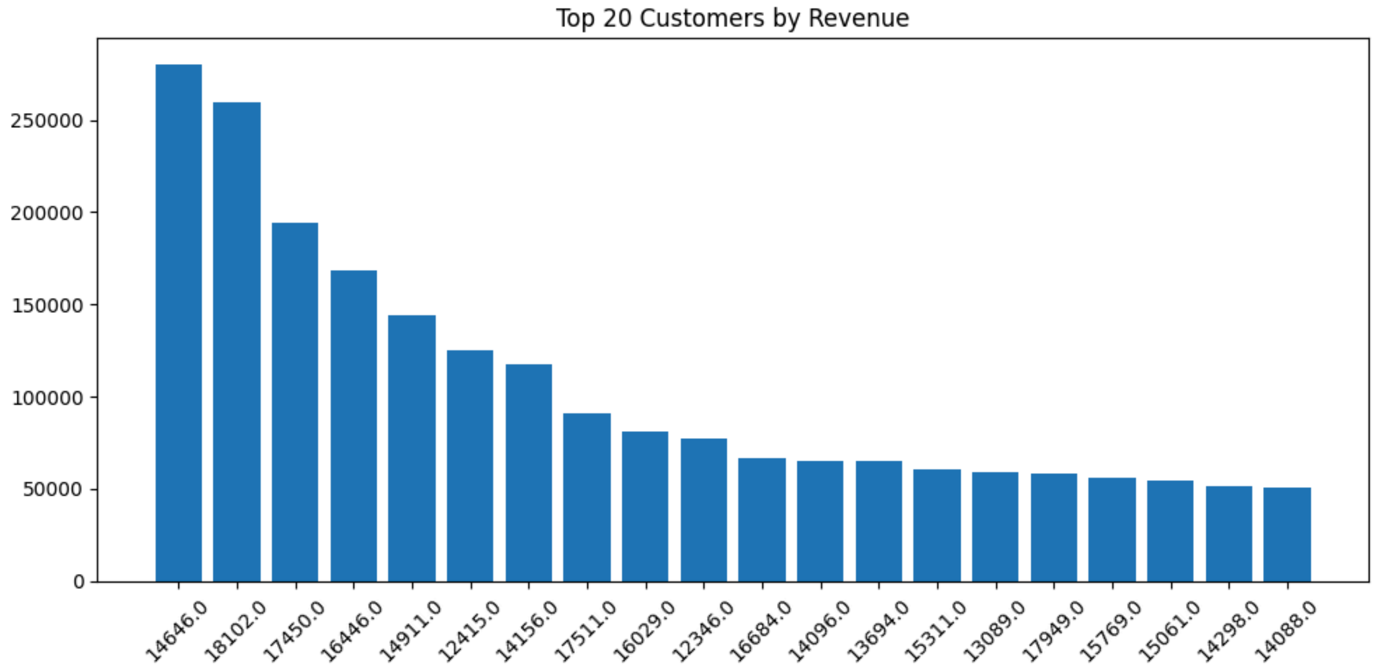
Page distribution in clickstream

```
q6 = "SELECT page, count(*) as cnt FROM clicks GROUP BY page ORDER BY cnt DESC"
pages_pdf = spark.sql(q6).toPandas()
plt.figure(figsize=(6,4))
plt.pie(pages_pdf['cnt'], labels=pages_pdf['page'], autopct='%1.1f%%')
plt.title('Clickstream Page Distribution')
plt.tight_layout()
plt.show()
```



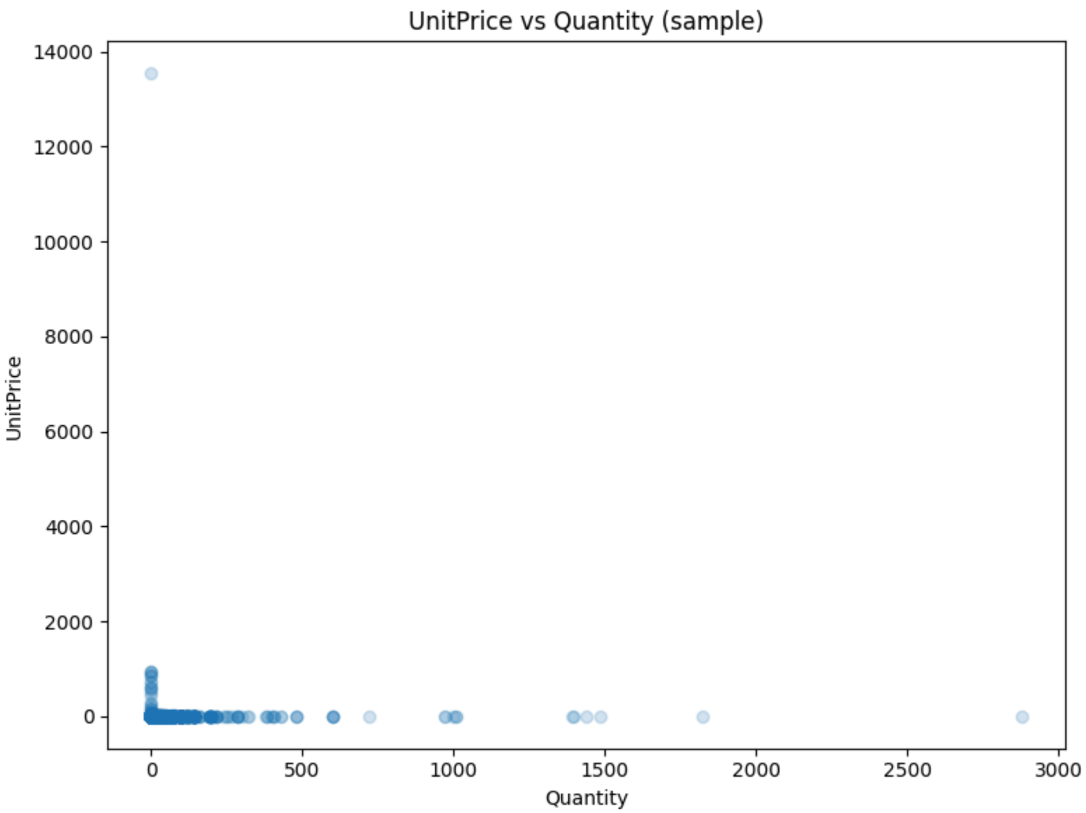
Top customers by revenue

```
q7 = "SELECT CustomerID, sum(Quantity*UnitPrice) as revenue FROM online WHERE CustomerID IS NOT NULL GROUP BY CustomerID ORDER BY revenue DESC LIMIT 20"
cust = spark.sql(q7).toPandas()
plt.figure(figsize=(10,5))
plt.bar(cust['CustomerID'].astype(str), cust['revenue'])
plt.xticks(rotation=45)
plt.title('Top 20 Customers by Revenue')
plt.tight_layout()
plt.show()
```



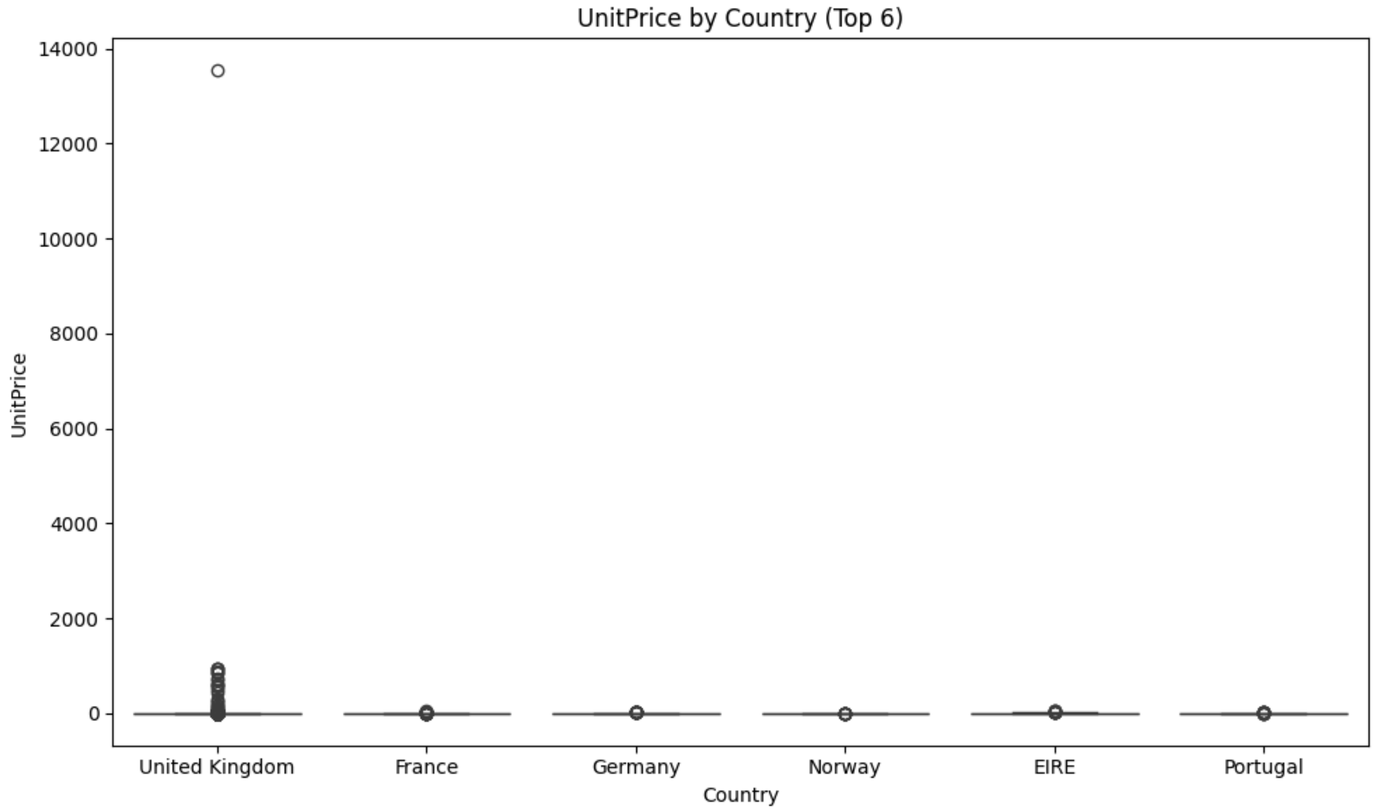
Correlation: UnitPrice vs Quantity (scatter)

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

numeric_cols = [c for c, dtype in online_df.dtypes if dtype in ('int', 'bigint', 'double')]
online_df_numeric = online_df.select(*numeric_cols)
s = online_df_numeric.limit(20000).toPandas()

plt.figure(figsize=(8, 6))
plt.scatter(s['Quantity'], s['UnitPrice'], alpha=0.2)
plt.xlabel('Quantity')
plt.ylabel('UnitPrice')
plt.title('UnitPrice vs Quantity (sample)')
plt.tight_layout()
plt.show()
```



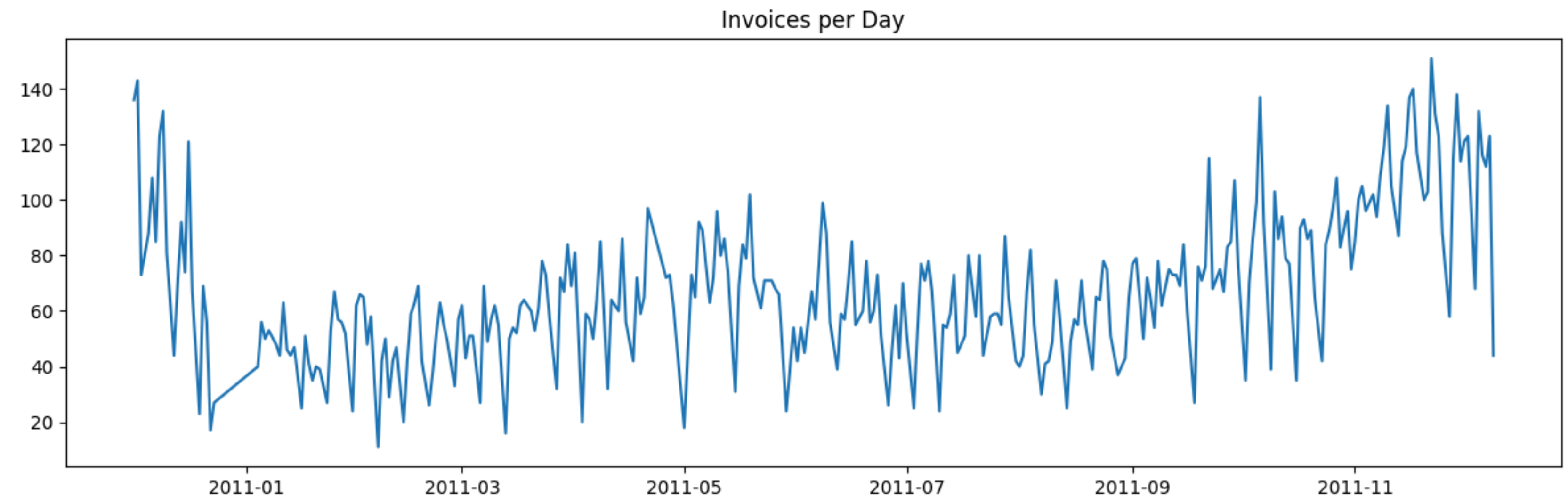
Boxplot of UnitPrice by Country (top 6 countries)

```
cols = ['Country', 'UnitPrice']
online_df_subset = online_df.select(*cols)
s = online_df_subset.limit(20000).toPandas()
s = s.dropna(subset=['Country', 'UnitPrice'])
top_countries = s['Country'].value_counts().index[:6]

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.boxplot(x='Country', y='UnitPrice', data=s[s['Country'].isin(top_countries)])
plt.title('UnitPrice by Country (Top 6)')
plt.tight_layout()
plt.show()
```



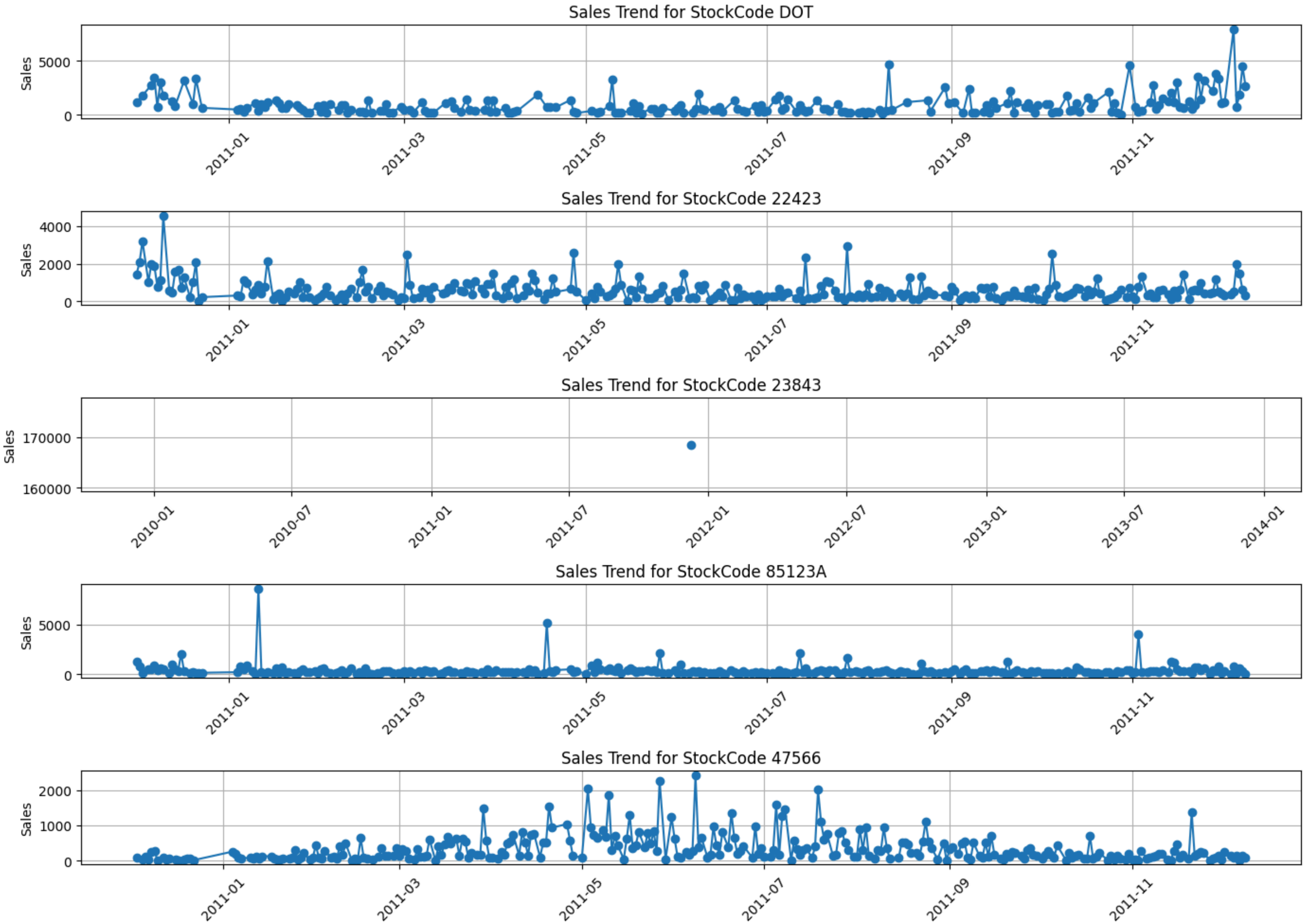
Distribution of invoices per day (hist)

```
q8 = "SELECT to_date(InvoiceDateTS) as day, count(distinct InvoiceNo) as invoices FROM online GROUP BY day ORDER BY day"
daily_invoices = spark.sql(q8).toPandas()
plt.figure(figsize=(12,4))
plt.plot(daily_invoices['day'], daily_invoices['invoices'])
plt.title('Invoices per Day')
plt.tight_layout()
plt.show()
```



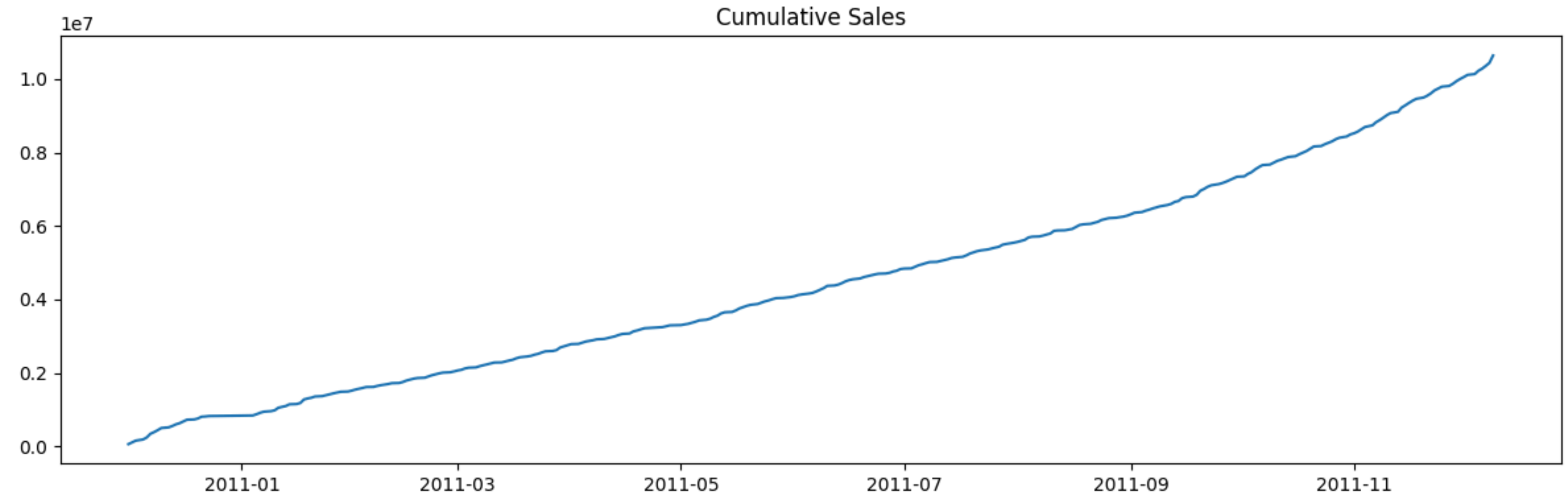
Top 10 StockCodes by sales trend (small multiples)

```
top_codes = [row['StockCode'] for row in spark.sql("""
    SELECT StockCode, SUM(Quantity * UnitPrice) AS sales
    FROM online
    GROUP BY StockCode
    ORDER BY sales DESC
""").collect()]

import matplotlib.pyplot as plt
plt.figure(figsize=(14, 10))
for i, code in enumerate(top_codes[:5], start=1):
    q = f"""
        SELECT DATE(InvoiceDateTS) AS day, SUM(Quantity * UnitPrice) AS sales
        FROM online
        WHERE StockCode = '{code}'
        GROUP BY day
        ORDER BY day
    """
    dfp = spark.sql(q).toPandas()

    plt.subplot(5, 1, i)
    plt.plot(dfp['day'], dfp['sales'], marker='o', linestyle='-')
    plt.title(f'Sales Trend for StockCode {code}')
    plt.xticks(rotation=45)
    plt.ylabel('Sales')
    plt.grid(True)

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



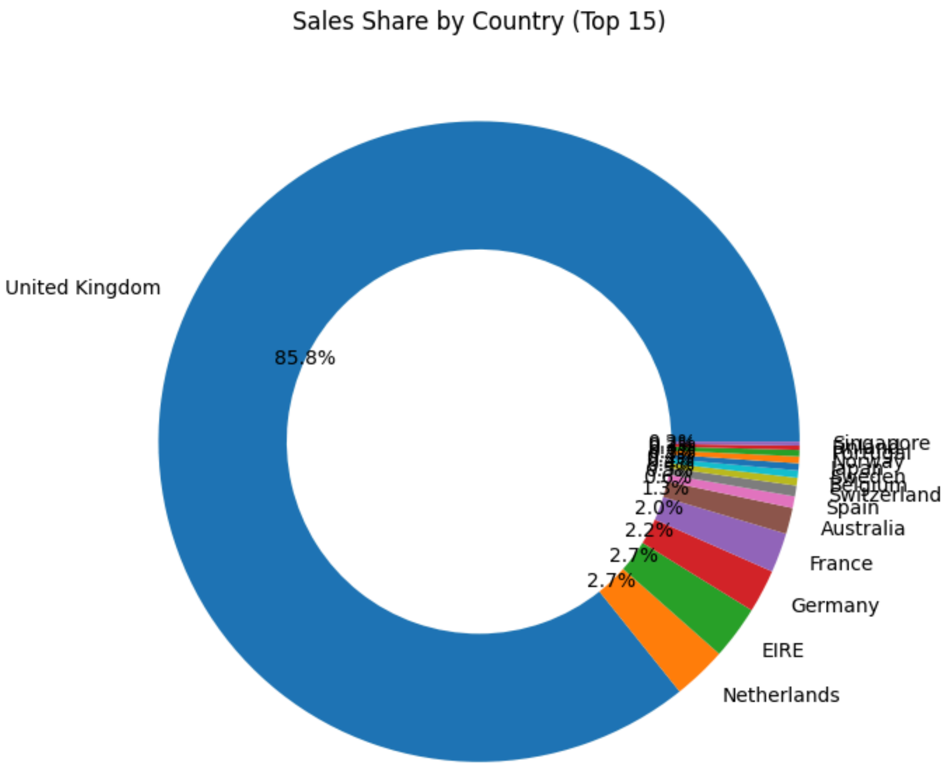
Cumulative sales curve

```
cumsum = daily.copy()
cumsum['cum_sales'] = cumsum['sales'].cumsum()
plt.figure(figsize=(12,4))
plt.plot(cumsum['day'], cumsum['cum_sales'])
plt.title('Cumulative Sales')
plt.tight_layout()
plt.show()
```



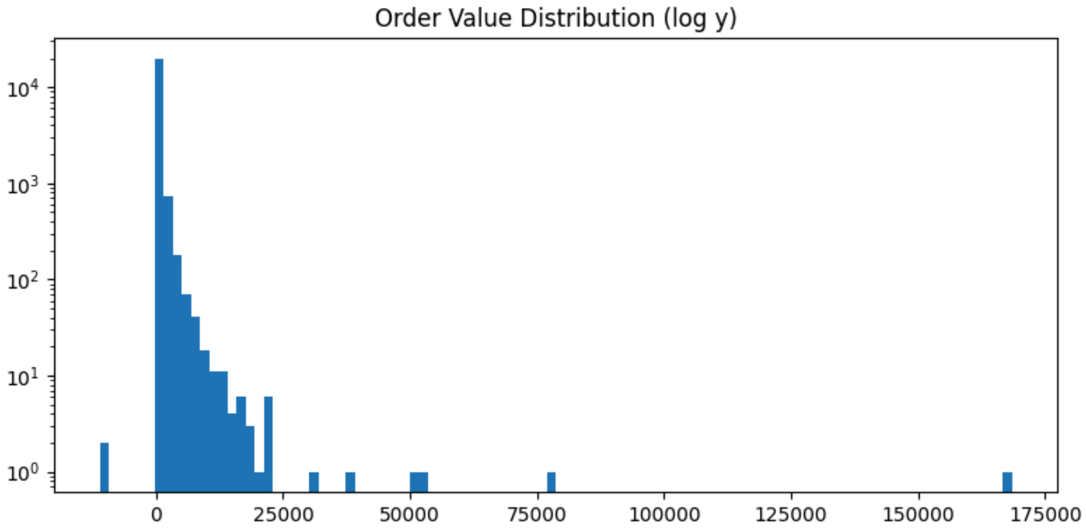
Sales share by country (donut)

```
plt.figure(figsize=(7,7))
plt.pie(pdf1['total_sales'], labels=pdf1['Country'], wedgeprops=dict(width=0.4), autopct='%1.1f%%')
plt.title('Sales Share by Country (Top 15)')
plt.tight_layout()
plt.show()
```



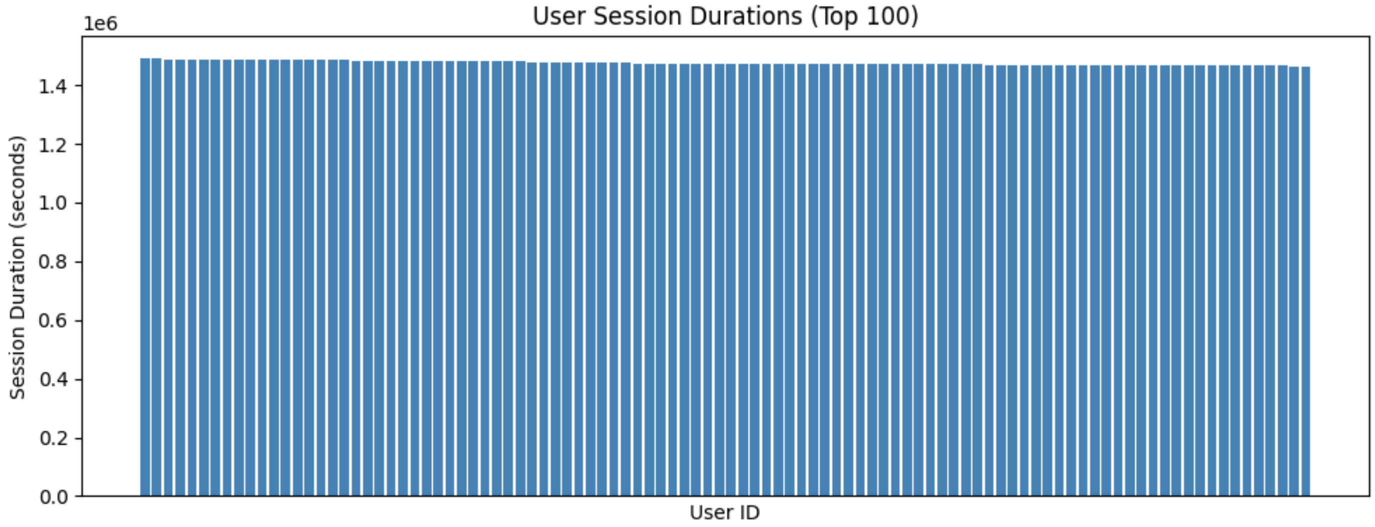
Average order value distribution

```
q9 = "SELECT InvoiceNo, sum(Quantity*UnitPrice) as order_value FROM online GROUP BY InvoiceNo"
ord_val = spark.sql(q9).toPandas()
plt.figure(figsize=(8,4))
plt.hist(ord_val['order_value'], bins=100)
plt.yscale('log')
plt.title('Order Value Distribution (log y)')
plt.tight_layout()
plt.show()
```



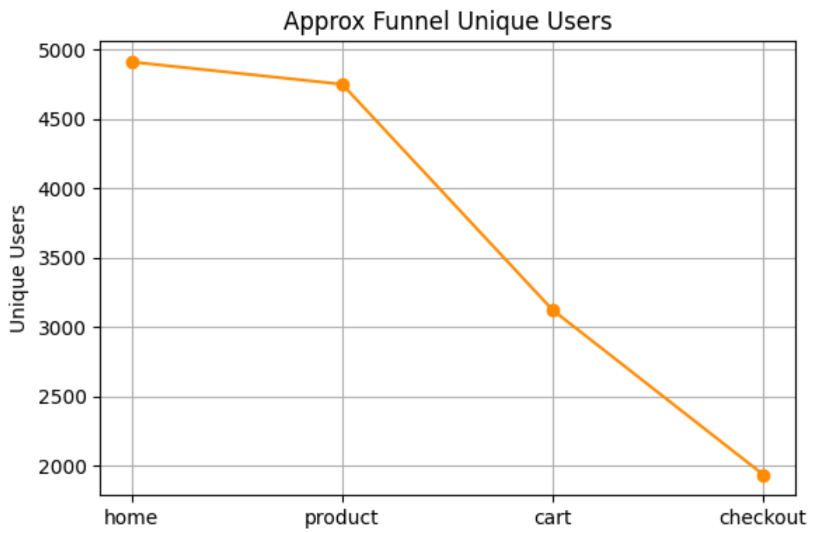
Time between first and last event per user (clickstream)

```
q10 = """
SELECT user_id,
       unix_timestamp(max(event_time_ts)) - unix_timestamp(min(event_time_ts)) AS ttl_sec
FROM clicks
GROUP BY user_id
ORDER BY ttl_sec DESC
LIMIT 100
"""

user_dur = spark.sql(q10).toPandas()
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))
plt.bar(user_dur['user_id'].astype(str), user_dur['ttl_sec'], color='steelblue')
plt.title('User Session Durations (Top 100)')
plt.xlabel('User ID')
plt.ylabel('Session Duration (seconds)')
plt.xticks([], [])
plt.tight_layout()
plt.show()
```



Page funnel conversion (home->product->cart->checkout) approximation

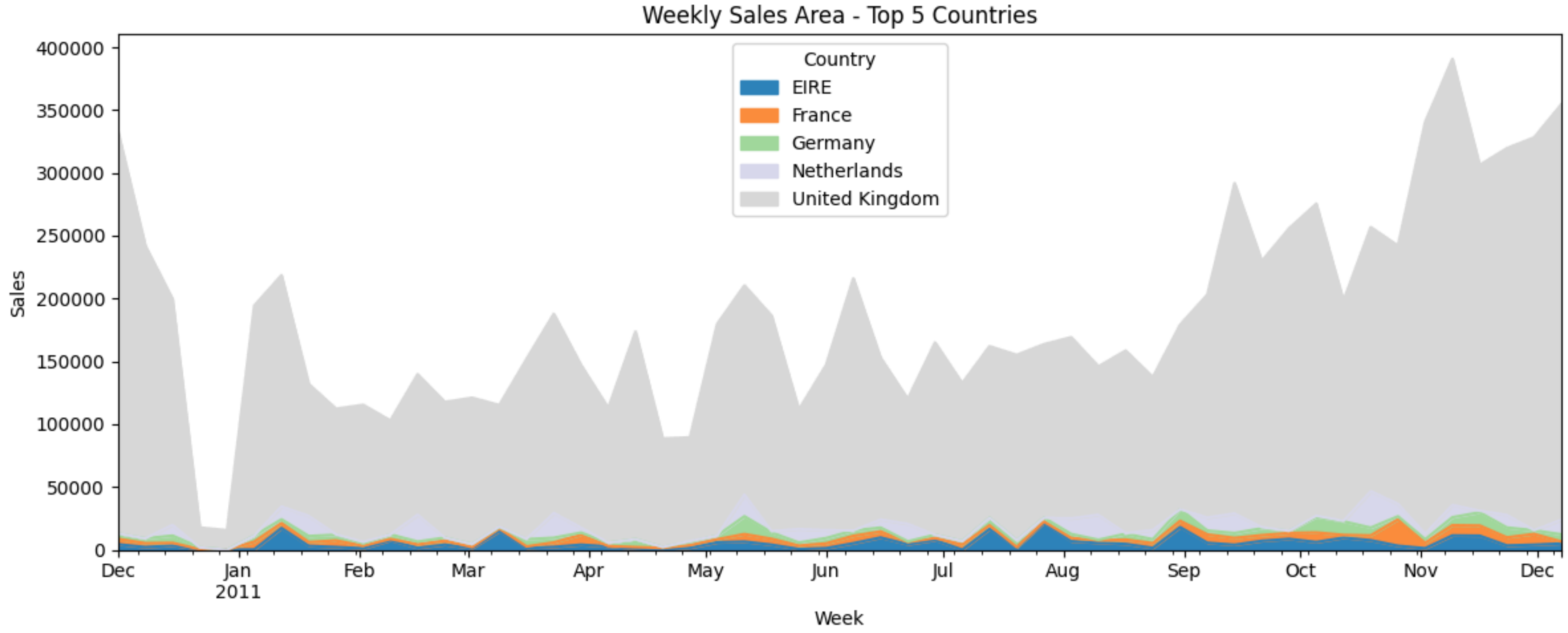
```
clicks_df_clean = clicks_df.select("page", "user_id")
pages_df = clicks_df_clean.limit(50000).toPandas()
home = pages_df[pages_df['page'] == 'home']['user_id'].nunique()
product = pages_df[pages_df['page'] == 'product']['user_id'].nunique()
cart = pages_df[pages_df['page'] == 'cart']['user_id'].nunique()
checkout = pages_df[pages_df['page'] == 'checkout']['user_id'].nunique()

import matplotlib.pyplot as plt
plt.figure(figsize=(6, 4))
plt.plot(['home', 'product', 'cart', 'checkout'], [home, product, cart, checkout], marker='o', color='darkorange')
plt.title('Approx Funnel Unique Users')
plt.ylabel('Unique Users')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Top N countries stacked monthly sales (stacked area)

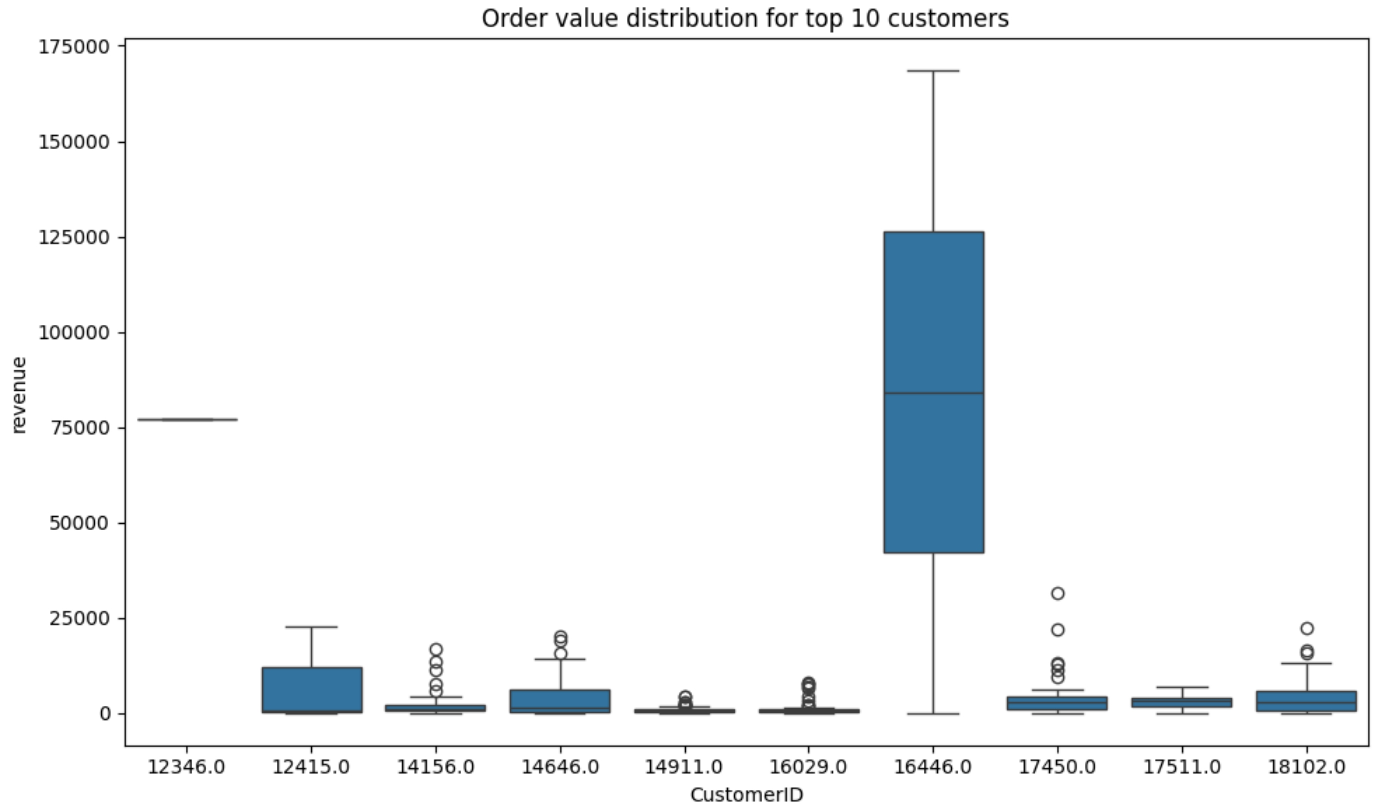
```
top_countries = pdf1['Country'].head(5).tolist()
q11 = f"""
SELECT Country, to_date(InvoiceDateTS) AS day, SUM(Quantity * UnitPrice) AS sales
FROM online
WHERE Country IN ({','.join(["'" + c + "'" for c in top_countries])})
GROUP BY Country, day
ORDER BY day
"""
stacked = spark.sql(q11).toPandas()
stacked['day'] = pd.to_datetime(stacked['day'], errors='coerce')
stacked_pivot = stacked.pivot_table(index='day', columns='Country', values='sales', aggfunc='sum').fillna(0)
stacked_pivot.index = pd.DatetimeIndex(stacked_pivot.index)
weekly_sales = stacked_pivot.resample('7D').sum()

import matplotlib.pyplot as plt
plt.figure(figsize=(12, 5))
weekly_sales.plot.area(figsize=(12, 5), cmap='tab20c')
plt.title('Weekly Sales Area - Top 5 Countries')
plt.ylabel('Sales')
plt.xlabel('Week')
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x500 with 0 Axes>
```



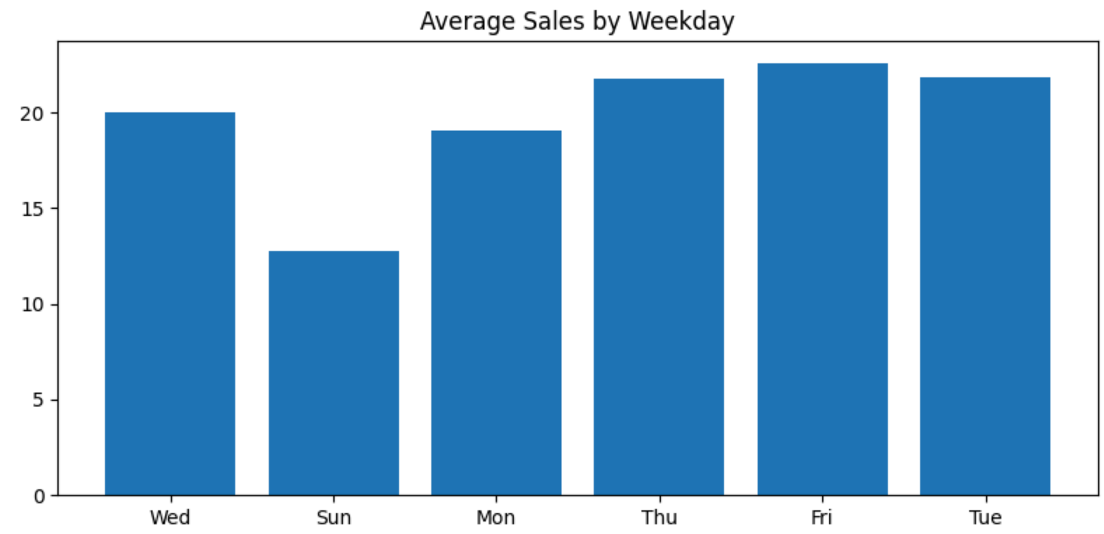
Top 10 customers spend distribution (box)

```
top_custs = cust['CustomerID'].head(10).tolist()
q12 = f"SELECT CustomerID, sum(Quantity*UnitPrice) as revenue, InvoiceNo FROM online WHERE CustomerID IN ({','.join([str(c) for c in top_custs])}) GROUP BY CustomerID, InvoiceNo"
cust_orders = spark.sql(q12).toPandas()
plt.figure(figsize=(10,6))
sns.boxplot(x='CustomerID', y='revenue', data=cust_orders)
plt.title('Order value distribution for top 10 customers')
plt.tight_layout()
plt.show()
```



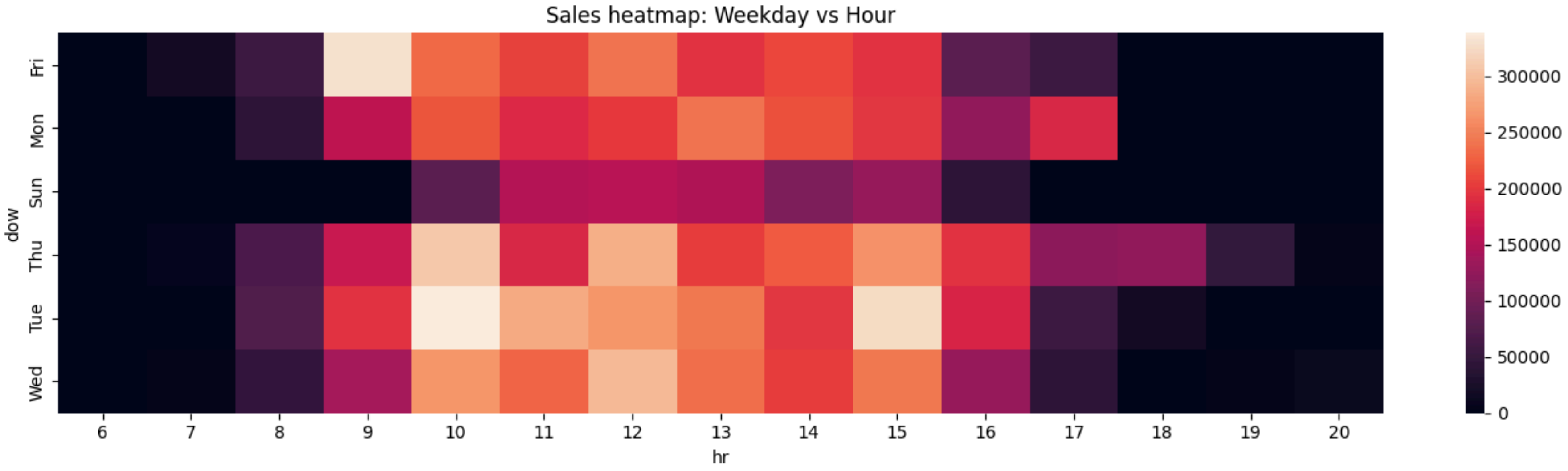
Seasonality analysis - average sales by weekday

```
q13 = "SELECT date_format(InvoiceDateTS,'E') as dow, avg(Quantity*UnitPrice) as avg_sales FROM online GROUP BY dow"
dow = spark.sql(q13).toPandas()
plt.figure(figsize=(8,4))
plt.bar(dow['dow'], dow['avg_sales'])
plt.title('Average Sales by Weekday')
plt.tight_layout()
plt.show()
```



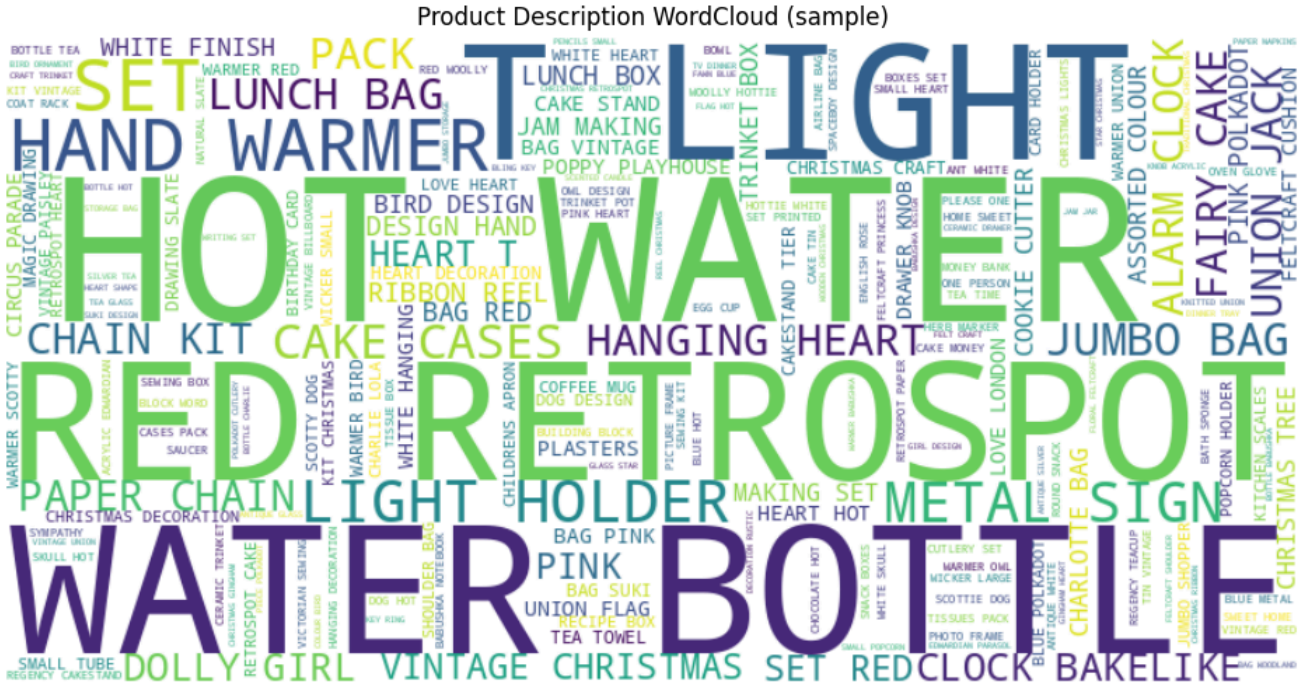
Heatmap of hour vs weekday (sales)

```
q14 = "SELECT date_format(InvoiceDateTS,'E') as dow, hour(InvoiceDateTS) as hr, sum(Quantity*UnitPrice) as sales FROM online GROUP BY dow, hr"
hm = spark.sql(q14).toPandas()
hm_pivot = hm.pivot(index='dow', columns='hr', values='sales').fillna(0)
plt.figure(figsize=(14,4))
sns.heatmap(hm_pivot)
plt.title('Sales heatmap: Weekday vs Hour')
plt.tight_layout()
plt.show()
```



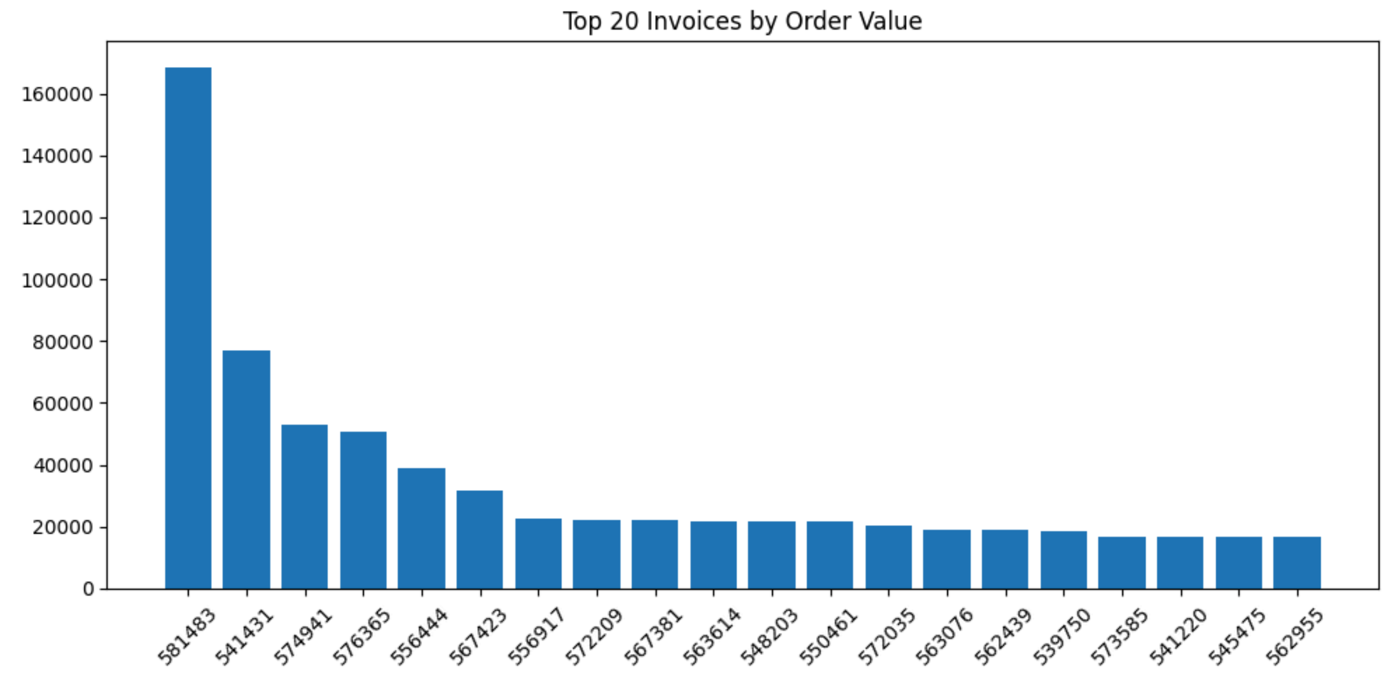
Product description word cloud (requires wordcloud lib)

```
!pip install -q wordcloud
from wordcloud import WordCloud
text = ' '.join(online_df.select('Description').na.drop().limit(20000).toPandas()['Description'].astype(str).tolist())
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)
plt.figure(figsize=(12,6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Product Description WordCloud (sample)')
plt.show()
```



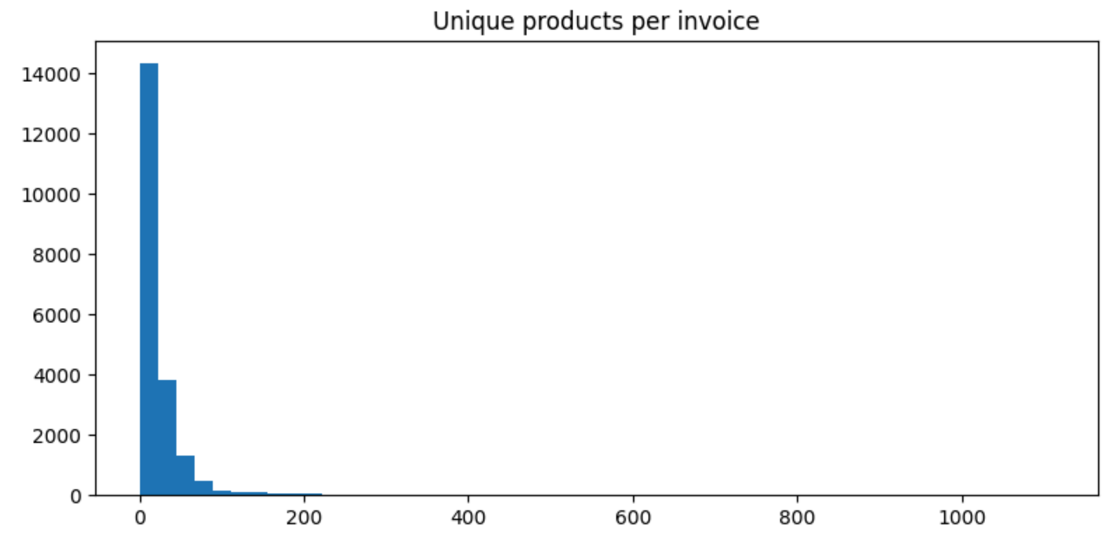
Top 20 invoices by order value (bar)

```
q15 = "SELECT InvoiceNo, sum(Quantity*UnitPrice) as order_value FROM online GROUP BY InvoiceNo ORDER BY order_value DESC LIMIT 20"
top_inv = spark.sql(q15).toPandas()
plt.figure(figsize=(10,5))
plt.bar(top_inv['InvoiceNo'].astype(str), top_inv['order_value'])
plt.xticks(rotation=45)
plt.title('Top 20 Invoices by Order Value')
plt.tight_layout()
plt.show()
```



Distribution of unique products purchased per invoice

```
q16 = "SELECT InvoiceNo, count(distinct StockCode) as unique_products FROM online GROUP BY InvoiceNo"
uniq_prod = spark.sql(q16).toPandas()
plt.figure(figsize=(8,4))
plt.hist(uniq_prod['unique_products'], bins=50)
plt.title('Unique products per invoice')
plt.tight_layout()
plt.show()
```

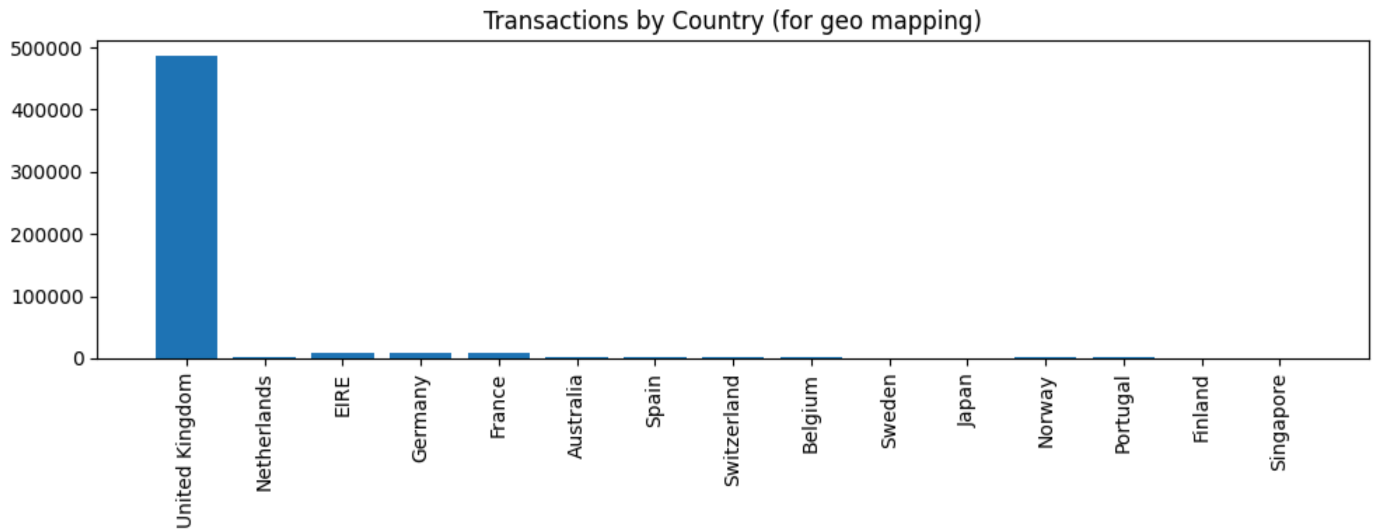


> **Geographical mapping hint - country counts (bar) (for full geo

---

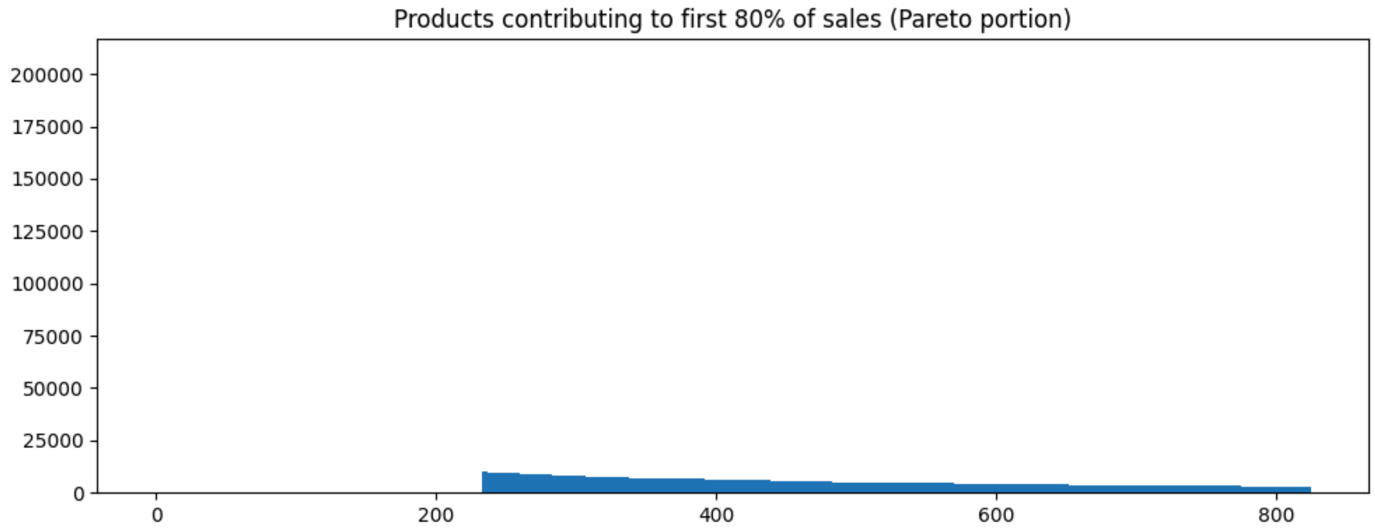
map in production, use plotly/choropleth)**

```
plt.figure(figsize=(10,4))
plt.bar(pdf1['Country'], pdf1['txns'])
plt.xticks(rotation=90)
plt.title('Transactions by Country (for geo mapping)')
plt.tight_layout()
plt.show()
```



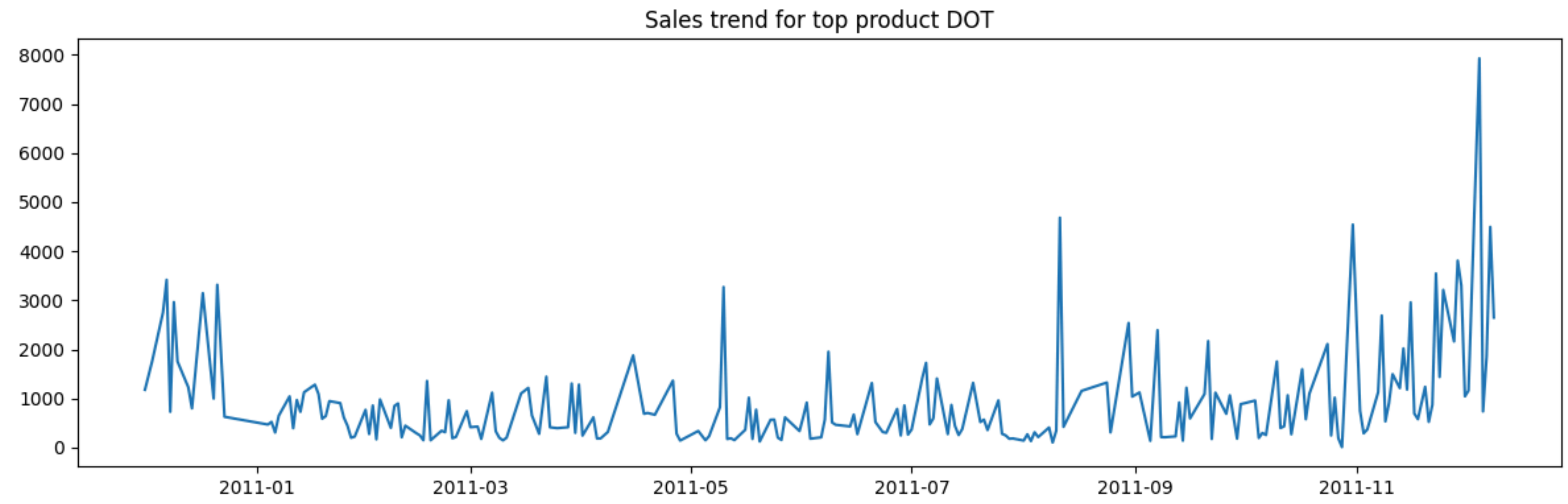
Pareto chart of products contributing to 80% sales

```
prod_sales = spark.sql("SELECT Description, sum(Quantity*UnitPrice) as sales FROM online GROUP BY Description ORDER BY sales DESC").toPandas()
prod_sales['cum_pct'] = prod_sales['sales'].cumsum() / prod_sales['sales'].sum()
cutoff = prod_sales[prod_sales['cum_pct']<=0.8]
plt.figure(figsize=(10,4))
plt.bar(range(len(cutoff)), cutoff['sales'])
plt.title('Products contributing to first 80% of sales (Pareto portion)')
plt.tight_layout()
plt.show()
```



Animated time series (static approach - multiple frames saved)

```
code = top_codes[0]
q_start = f"SELECT to_date(InvoiceDateTS) as day, sum(Quantity*UnitPrice) as sales FROM online WHERE StockCode='{code}' GROUP BY day ORDER BY day"
prod_trend = spark.sql(q_start).toPandas()
plt.figure(figsize=(12,4))
plt.plot(prod_trend['day'], prod_trend['sales'])
plt.title(f'Sales trend for top product {code}')
plt.tight_layout()
plt.show()
```



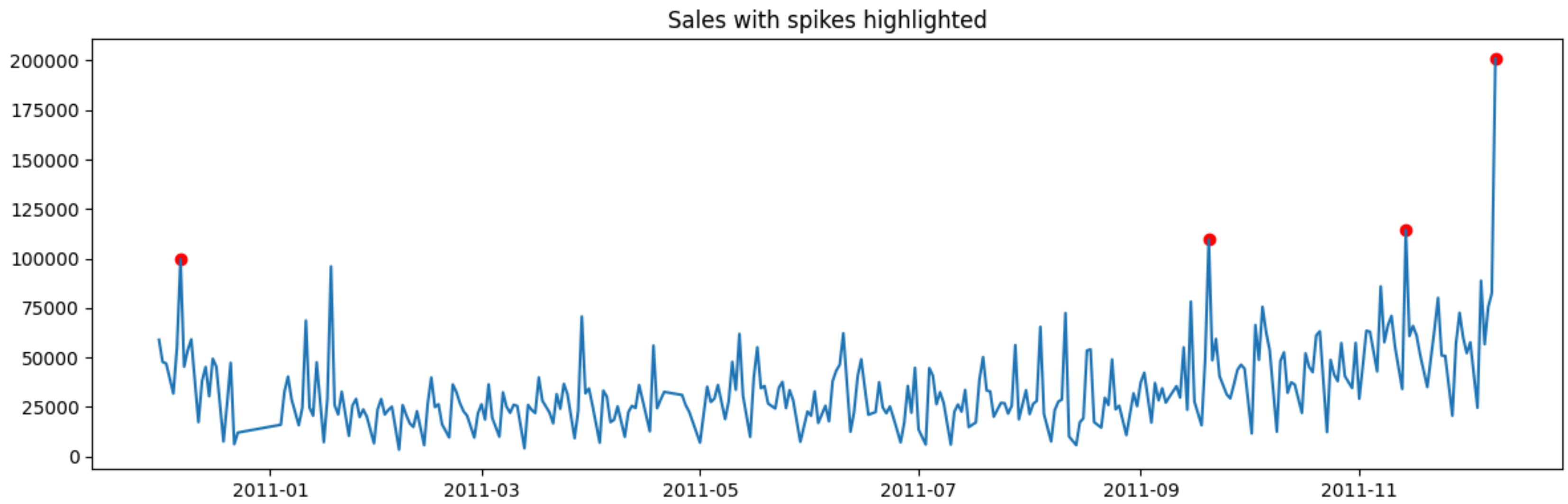
Multi-metric dashboard snapshot (3 metrics)

```
total_sales = daily['sales'].sum()
avg_order = ord_val['order_value'].mean()
num_customers = online_df.select('CustomerID').na.drop().distinct().count()
print('Total sales (sum):', total_sales)
print('Avg order value:', avg_order)
print('Unique customers:', num_customers)
```

```
Total sales (sum): 10644560.42400001
Avg order value: 513.5353350057893
Unique customers: 4339
```

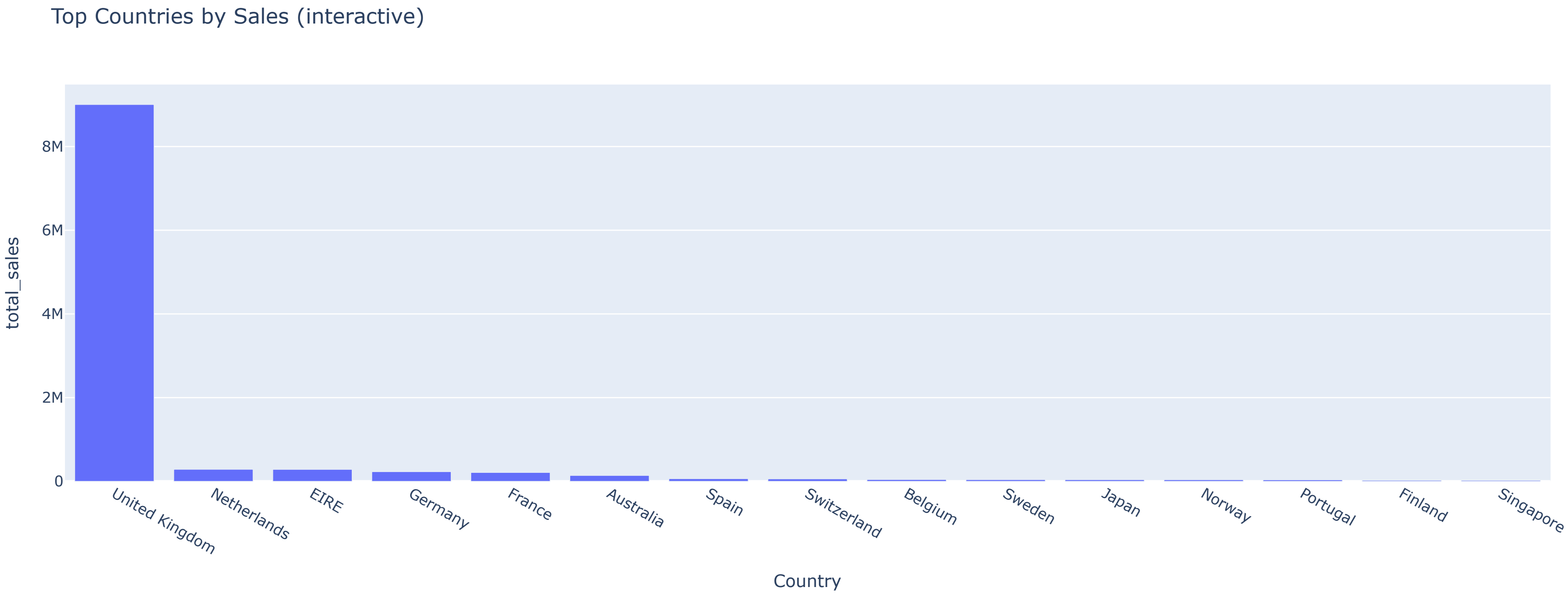
Spike detection - find days with sales > mean + 3*std

```
import numpy as np
mu = daily['sales'].mean()
sigma = daily['sales'].std()
spikes = daily[daily['sales'] > mu + 3*sigma]
plt.figure(figsize=(12,4))
plt.plot(daily['day'], daily['sales'])
plt.scatter(spikes['day'], spikes['sales'], color='red')
plt.title('Sales with spikes highlighted')
plt.tight_layout()
plt.show()
```



Export a Plotly interactive figure (Top countries by sales)

```
import plotly.express as px
fig = px.bar(pdf1, x='Country', y='total_sales', title='Top Countries by Sales (interactive)')
fig.show()
```



ML example (Spark MLlib)

```
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.regression import LinearRegression

df_ml = online_df.select('Quantity','UnitPrice','Country').na.drop()
indexer = StringIndexer(inputCol='Country', outputCol='CountryIdx')
df_ml = indexer.fit(df_ml).transform(df_ml)
assembler = VectorAssembler(inputCols=['Quantity','CountryIdx'], outputCol='features')
df_ml = assembler.transform(df_ml).withColumnRenamed('UnitPrice','label')
train,test = df_ml.randomSplit([0.8,0.2], seed=42)
lr = LinearRegression(featuresCol='features', labelCol='label')
model = lr.fit(train)
print('Trained LinearRegression model coefficients:', model.coefficients)
```

```
Trained LinearRegression model coefficients: [-0.00074418905854326866,0.1508679470622859]
```

Save curated Delta tables and sample Parquet for BI

```
spark.read.format("delta") \
    .load(f"{CURATED_PATH}/online_retail_curated.delta") \
    .write.mode("overwrite") \
    .parquet("/content/online_retail_curated_parquet")
```