Orchestrated Domain-Driven Data Mesh Architecture.

Install system & Python packages

```
!apt-get update -qq
!apt-get install -y openjdk-11-jdk-headless -qq

!pip install -q pyspark==3.3.2 delta-spark==2.2.0 findspark faker kafka-python plotly seaborn matplotlib networkx fastapi uvicorn nest-asyncio pyngrok scikit-learn pandas-profili
```

Show hidden output

```
!pip install faker
```

```
Requirement already satisfied: faker in /usr/local/lib/python3.12/dist-packages (37.8.0)
Requirement already satisfied: tzdata in /usr/local/lib/python3.12/dist-packages (from faker) (2025.2)
```

Python Environment for Analytics & Clustering

```
import os

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["ARROW_PRE_0_15_IPC_FORMAT"] = "1"

import json
import random
import threading
from datetime import datetime

import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.cluster import KMeans
from faker import Faker
faker = Faker()
```

```
!pip install delta-spark
```

Show hidden output

```
!pip install pyspark==3.5.0 delta-spark==3.1.0


```

Show hidden output

```
import os

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["ARROW_PRE_0_15_IPC_FORMAT"] = "1"
```

Unified Data Simulation & Lakehouse Analytics Framework

```
from pyspark.sql import SparkSession
from delta import configure_spark_with_delta_pip

builder = SparkSession.builder \
    .appName("DeltaLake-Colab") \
    .master("local[*]") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")

spark = configure_spark_with_delta_pip(builder).getOrCreate()
```

```
print("Spark version:", spark.version)
```

```
Spark version: 3.5.0
```

Spark + Delta Lake: Reliable Data Storage in Action

```
data = spark.range(0, 5)
data.write.format("delta").mode("overwrite").save("/tmp/delta-table")

df = spark.read.format("delta").load("/tmp/delta-table")
df.show()
```

```
+---+
| id|
+---+
|  2|
|  3|
|  4|
|  0|
|  1|
+---+
```

Delta Lake Integration with PySpark: Init Function Demo

```
def init_spark(app_name="DeltaLake-Colab"):
    from pyspark.sql import SparkSession
    from delta import configure_spark_with_delta_pip
```

```
    builder = SparkSession.builder \
        .appName(app_name) \
        .master("local[*]") \
        .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
        .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")

    return configure_spark_with_delta_pip(builder).getOrCreate()
spark = init_spark()
```

```
spark = init_spark()
```

```
BASE_DELTA = "/tmp/delta_data_mesh"
VIS_DIR = "/tmp/visuals"
os.makedirs(BASE_DELTA, exist_ok=True)
os.makedirs(VIS_DIR, exist_ok=True)

def save_plot(fig, name):
    path = os.path.join(VIS_DIR, name)
    try:
        fig.write_image(path)
    except Exception:
        plt.savefig(path)
    return path

def show_image(path):
    from IPython.display import Image, display
    display(Image(path))
```

Scalable Dataset Creation for Analytics Workflows

```
import numpy as np
import pandas as pd
import random
from datetime import datetime, timedelta
from faker import Faker

fake = Faker()
fake.seed_instance(1234)
np.random.seed(1234)
random.seed(1234)
N_CUSTOMERS = 5000
N_PRODUCTS = 800
N_SALES = 120000
N_FINANCE = 50000
customers = []

for cid in range(1, N_CUSTOMERS + 1):
    signup = fake.date_time_between(start_date='-4y', end_date='now')
    customers.append({
        "customer_id": cid,
```

```
        "name": fake.name(),
        "email": fake.email(),
        "country": fake.country_code(),
        "segment": random.choices(["consumer", "small_business", "enterprise"], [0.7, 0.25, 0.05])[0],
        "signup_date": signup,
        "is_active": random.choices([True, False], weights=[0.92, 0.08])[0]
    })
customers = pd.DataFrame(customers)
categories = ["software", "hardware", "subscription", "service", "accessory"]
products = []
for pid in range(1, N_PRODUCTS + 1):
    launch = fake.date_between(start_date='-6y', end_date='today')
    products.append({
        "product_id": pid,
        "product_name": f"{fake.word().title()} {pid}",
        "category": random.choice(categories),
        "price": round(abs(np.random.normal(120, 60)) + 5, 2),
        "launch_date": launch
    })
products = pd.DataFrame(products)
start_date = datetime.now() - timedelta(days=730)
sales = []
for i in range(1, N_SALES + 1):
    cid = random.randint(1, N_CUSTOMERS)
    pid = random.randint(1, N_PRODUCTS)
    qty = random.choices([1, 2, 3, 4, 5], weights=[0.7, 0.15, 0.08, 0.05, 0.02])[0]
    price = float(products.loc[products.product_id == pid, "price"].values[0])
    ts = fake.date_time_between(start_date=start_date, end_date='now')
    sales.append({
        "transaction_id": i,
        "customer_id": cid,
        "product_id": pid,
        "quantity": qty,
        "unit_price": price,
        "total_price": round(qty * price, 2),
        "ts": ts
    })
sales = pd.DataFrame(sales)
finance = []
for i in range(1, N_FINANCE + 1):
    ts = fake.date_time_between(start_date=start_date, end_date='now')
    finance.append({
        "entry_id": i,
        "date": ts,
        "amount": round(np.random.normal(2000, 800), 2),
        "type": random.choice(["expense", "revenue"]),
        "note": fake.sentence(nb_words=6)
    })
finance = pd.DataFrame(finance)
customers.to_parquet("/tmp/customers.parquet", index=False)
products.to_parquet("/tmp/products.parquet", index=False)
sales.to_parquet("/tmp/sales.parquet", index=False)
finance.to_parquet("/tmp/finance.parquet", index=False)

print("Generated datasets:")
```

```
print("Customers:", customers.shape)
print("Products:", products.shape)
print("Sales:", sales.shape)
print("Finance:", finance.shape)
```

```
Generated datasets:
Customers: (5000, 7)
Products: (800, 5)
Sales: (120000, 7)
Finance: (50000, 5)
```

Analyzing Customer Demographics with Pandas & Plotly

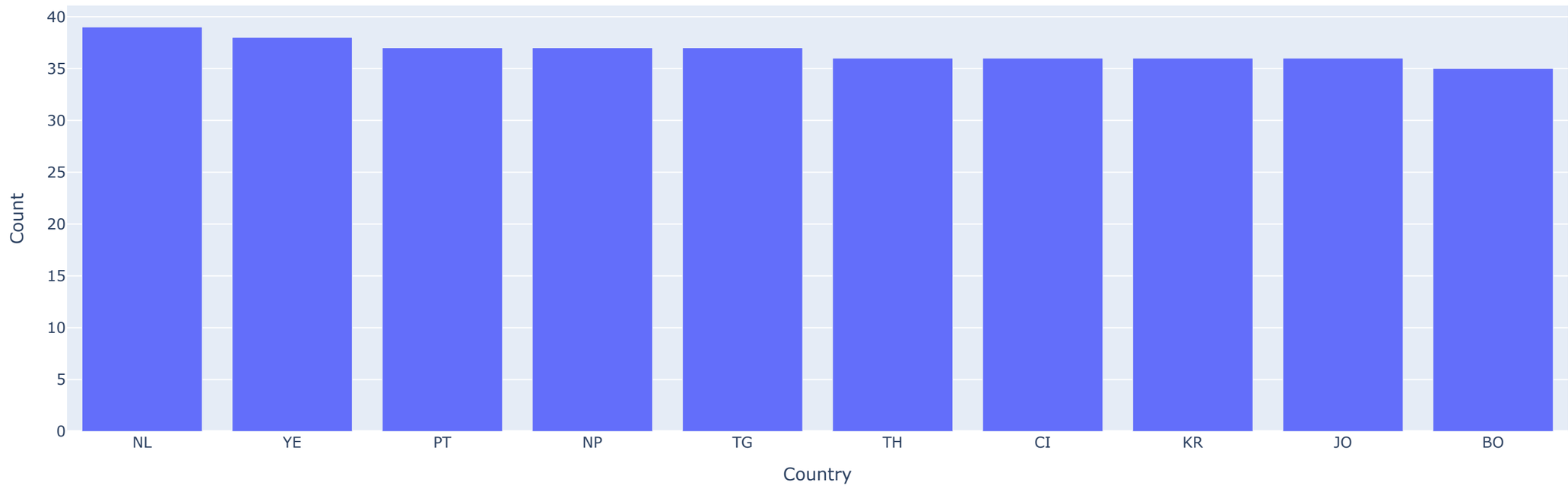
```
display(customers.head(3))
display(products.head(3))
display(sales.head(3))

top_countries = customers['country'].value_counts().head(10).reset_index()
fig = px.bar(top_countries, x='country', y='count', title='Top 10 Customer Countries', labels={'country':'Country','count':'Count'})
fig.show()
```

	customer_id	name	email	country	segment	signup_date	is_active
0	1	Adam Bartlett	powerschristopher@example.com	GW	enterprise	2025-08-04 17:40:31.328413	True
1	2	Mrs. Mariah Washington	lhernandez@example.net	CI	consumer	2021-10-18 05:57:24.262229	True
2	3	Shirley Wright	rjones@example.org	SG	small_business	2025-02-08 19:21:55.569191	True

	product_id	product_name	category	price	launch_date
0	1	Despite 1	accessory	153.29	2021-12-10
1	2	Hand 2	software	53.54	2023-12-27
2	3	Give 3	hardware	210.96	2022-12-31

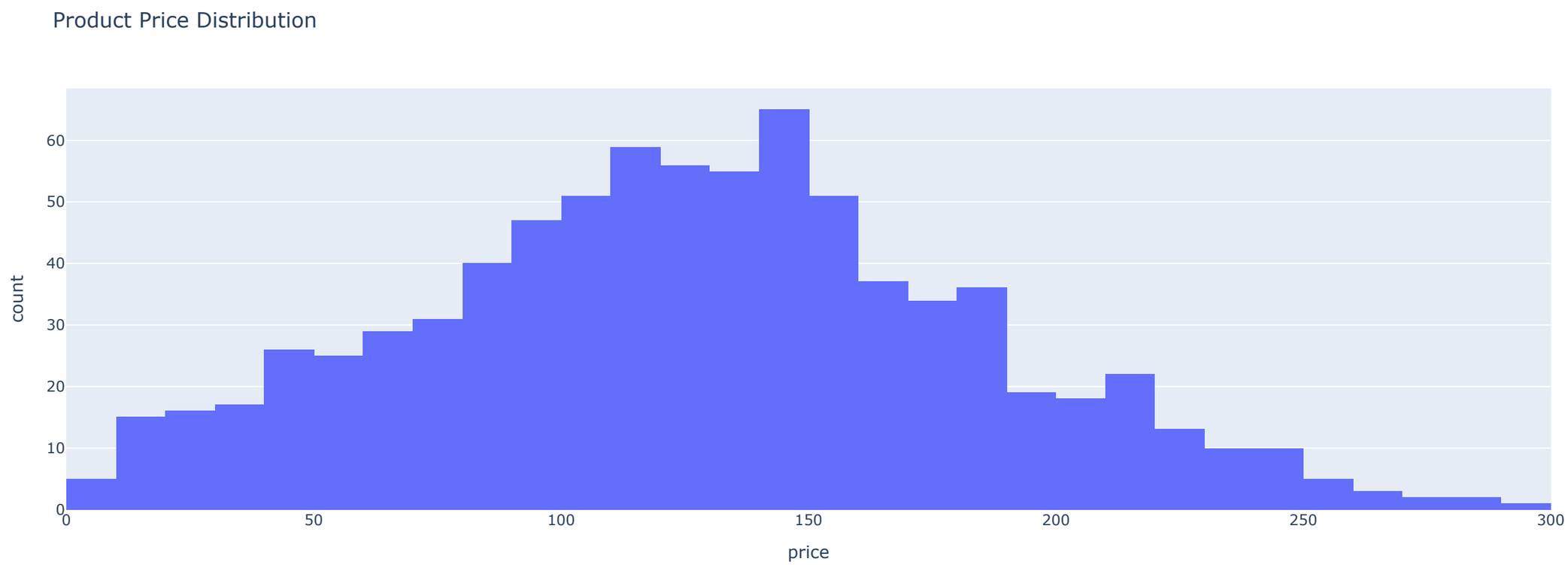
	transaction_id	customer_id	product_id	quantity	unit_price	total_price	ts
0	1	2982	202	1	87.80	87.80	2025-08-23 16:10:34.632931
1	2	3489	562	1	188.01	188.01	2023-10-07 15:37:26.252957
2	3	2659	123	1	84.03	84.03	2024-02-14 05:58:20.986930

Top 10 Customer Countries



product price distribution

```
fig = px.histogram(products, x='price', nbins=50, title='Product Price Distribution')
fig.show()
```



load pandas to spark

```
spark_customers = spark.createDataFrame(customers)
spark_products = spark.createDataFrame(products)
spark_sales = spark.createDataFrame(sales)
spark_finance = spark.createDataFrame(finance)

spark_customers.createOrReplaceTempView("raw_customers")
spark_products.createOrReplaceTempView("raw_products")
spark_sales.createOrReplaceTempView("raw_sales")
spark_finance.createOrReplaceTempView("raw_finance")

print("Registered temp views: raw_customers, raw_products, raw_sales, raw_finance")
```

```
Registered temp views: raw_customers, raw_products, raw_sales, raw_finance
```

Duplicate Rate Analysis Across Domains with PySpark & Plotly

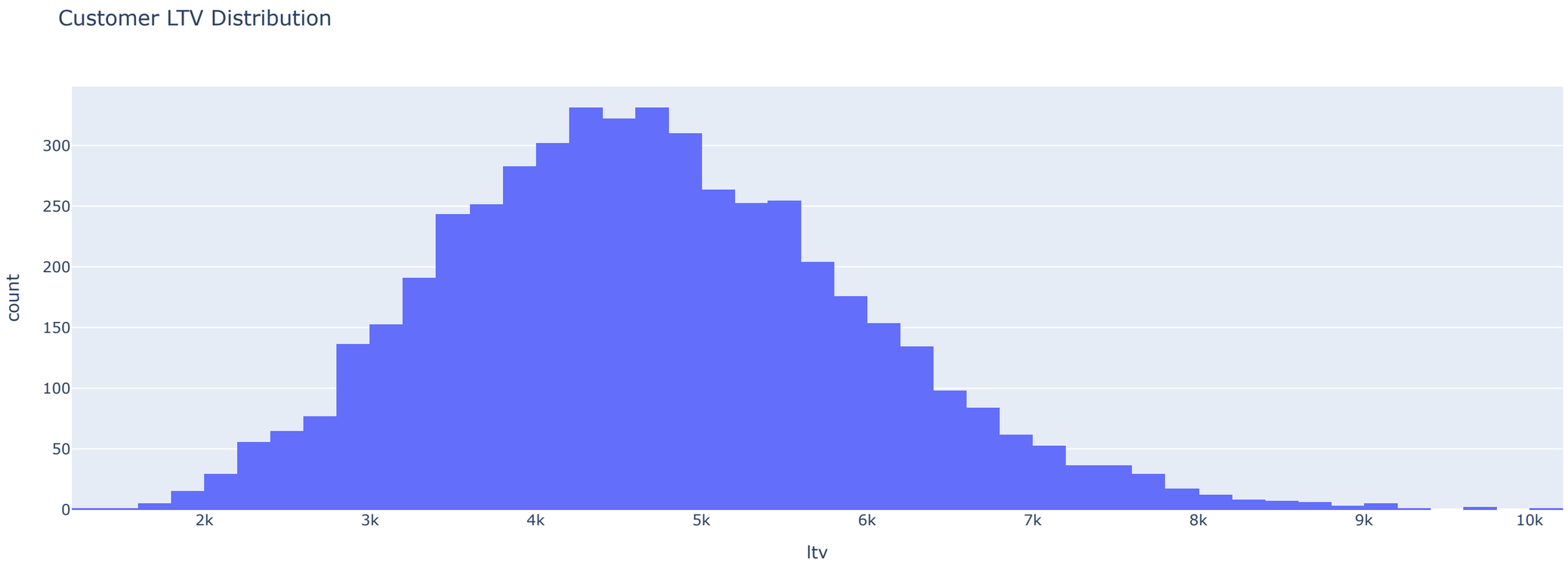
```
from pyspark.sql import functions as F
import plotly.express as px

def dq_report(df, key_cols):
    tot = df.count()
    dedup = df.dropDuplicates(key_cols).count()
    dup_rate = (tot - dedup) / max(1, tot)
    return tot, dedup, dup_rate

dup_rates = {}
for name, df, keys in [
    ("customers", spark_customers, ["customer_id"]),
    ("products", spark_products, ["product_id"]),
    ("sales", spark_sales, ["transaction_id"]),
    ("finance", spark_finance, ["entry_id"])
]:
    tot, dedup, dup_rate = dq_report(df, keys)
    dup_rates[name] = dup_rate
    print(f"{name}: total={tot}, deduped={dedup}, dup_rate={dup_rate:.6f}")

fig = px.bar(
    x=list(dup_rates.keys()),
    y=list(dup_rates.values()),
    title='Duplication Rates by Domain',
    labels={'x': 'domain', 'y': 'dup_rate'},
    text=[f"{v:.6f}" for v in dup_rates.values()]  # Show exact values
)
fig.update_traces(textposition='outside')
fig.update_layout(yaxis=dict(range=[0, 0.05]))  # Force visible range
fig.show()
```

```
customers: total=5000, deduped=5000, dup_rate=0.000000
products: total=800, deduped=800, dup_rate=0.000000
sales: total=120000, deduped=120000, dup_rate=0.000000
finance: total=50000, deduped=50000, dup_rate=0.000000
```

Customer Lifetime Value Distribution with Plotly Histogram

```
spark_sales = spark_sales.withColumn("ts", F.to_timestamp("ts"))
spark_sales = spark_sales.withColumn("total_price", F.round(F.col("total_price").cast("double"),2))

ltv = spark_sales.groupBy("customer_id").agg(F.sum("total_price").alias("ltv"), F.count("*").alias("purchase_count"))
spark_customers = spark_customers.join(ltv, on="customer_id", how="left").fillna({"ltv":0, "purchase_count":0})

from delta.tables import DeltaTable
for name, df in [("customers", spark_customers), ("products", spark_products), ("sales", spark_sales), ("finance", spark_finance)]:
    path = f"{BASE_DELTA}/{name}"
    df.write.format("delta").mode("overwrite").save(path)
    print("Wrote Delta:", path)
pdf_customers = spark_customers.toPandas()
fig = px.histogram(pdf_customers, x='ltv', nbins=80, title='Customer LTV Distribution')
fig.show()
```

```
Wrote Delta: /tmp/delta_data_mesh/customers
Wrote Delta: /tmp/delta_data_mesh/products
Wrote Delta: /tmp/delta_data_mesh/sales
Wrote Delta: /tmp/delta_data_mesh/finance
```




```
!pip install numba==0.56.4
```

```
Collecting numba==0.56.4
  Using cached numba-0.56.4.tar.gz (2.4 MB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

```
!pip install pandas-profiling==3.6.6
```

Show hidden output

```
!pip install pydantic==1.10.13
```

Show hidden output

```
!pip install pandas-profiling==3.6.6
```

Show hidden output

Quick EDA from Delta Tables Using Pandas & HTML Reports

```
!pip install ydata-profiling
from ydata_profiling import ProfileReport

# Assuming spark_customers is already defined and contains your data
# If not, you might need to load it here:
# from pyspark.sql import SparkSession
# spark = SparkSession.builder.appName("profiling").getOrCreate()
# spark_customers = spark.read.format("delta").load("/tmp/delta_data_mesh/customers")

df = spark.read.format("delta").load("/tmp/delta_data_mesh/customers").limit(10000).toPandas()
profile = ProfileReport(df, title="Customers Profile", explorative=True)

profile_path = "/tmp/customers_profile_report.html"
profile.to_file(profile_path)
from IPython.display import display, HTML
display(HTML(f"<b>✅ Saved customers profile:</b> <a href='{profile_path}' target='_blank'>View Report</a>"))
```

```
Requirement already satisfied: ydata-profiling in /usr/local/lib/python3.12/dist-packages (4.16.1)
Requirement already satisfied: scipy<1.16,>=1.4.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (1.15.3)
Requirement already satisfied: pandas!=1.4.0,<3.0,>1.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.2.2)
Requirement already satisfied: matplotlib<=3.10,>=3.5 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.10.0)
Collecting pydantic>=2 (from ydata-profiling)
  Using cached pydantic-2.11.9-py3-none-any.whl.metadata (68 kB)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (6.0.2)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.1.6)
Collecting visions<0.8.2,>=0.7.5 (from visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling)
  Using cached visions-0.8.1-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: numpy<2.2,>=1.16.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.0.2)
Requirement already satisfied: htmlmin==0.1.12 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.1.12)
Requirement already satisfied: phik<0.13,>=0.11.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.12.5)
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.32.4)
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.67.1)
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.13.2)
Requirement already satisfied: multimethod<2,>=1.4 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (1.4)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.14.5)
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.4.4)
Requirement already satisfied: imagehash==4.3.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.3.1)
Requirement already satisfied: wordcloud>=1.9.3 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (1.9.4)
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (1.9.2)
Requirement already satisfied: numba<=0.61,>=0.56.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.60.0)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.12/dist-packages (from imagehash==4.3.1->ydata-profiling) (1.9.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (from imagehash==4.3.1->ydata-profiling) (11.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2<3.2,>=2.11.1->ydata-profiling) (2.1.5)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (4.59.2)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (25.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (2.9.0.post0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages (from numba<=0.61,>=0.56.0->ydata-profiling) (0.43.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas!=1.4.0,<3.0,>1.1->ydata-profiling) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas!=1.4.0,<3.0,>1.1->ydata-profiling) (2025.2)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.12/dist-packages (from phik<0.13,>=0.11.1->ydata-profiling) (1.1.1)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2->ydata-profiling) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2->ydata-profiling) (2.33.2)
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2->ydata-profiling) (4.15.0)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2->ydata-profiling) (0.4.1)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (2025.8.3)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.12/dist-packages (from statsmodels<1,>=0.13.2->ydata-profiling) (1.0.1)
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.12/dist-packages (from visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling) (25.3.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.12/dist-packages (from visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling) (3.5)
Requirement already satisfied: puremagic in /usr/local/lib/python3.12/dist-packages (from visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling) (1.30)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib<=3.10,>=3.5->ydata-profiling) (1.17.0)
Using cached pydantic-2.11.9-py3-none-any.whl (444 kB)
Using cached visions-0.8.1-py3-none-any.whl (105 kB)
Installing collected packages: pydantic, visions
  Attempting uninstall: pydantic
    Found existing installation: pydantic 1.10.13
    Uninstalling pydantic-1.10.13:
      Successfully uninstalled pydantic-1.10.13
```

```
!pip install delta-spark pandas faker

import os
import pandas as pd
import numpy as np
import random
from faker import Faker
from pyspark.sql import SparkSession
from delta import configure_spark_with_delta_pip
from pyspark.sql import functions as F

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["ARROW_PRE_0_15_IPC_FORMAT"] = "1"
builder = SparkSession.builder \
    .appName("Customer360") \
    .master("local[*]") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")

spark = configure_spark_with_delta_pip(builder).getOrCreate()

BASE_DELTA = "/tmp/pp-010"
fake = Faker()
fake.seed_instance(1234)
np.random.seed(1234)
random.seed(1234)
```

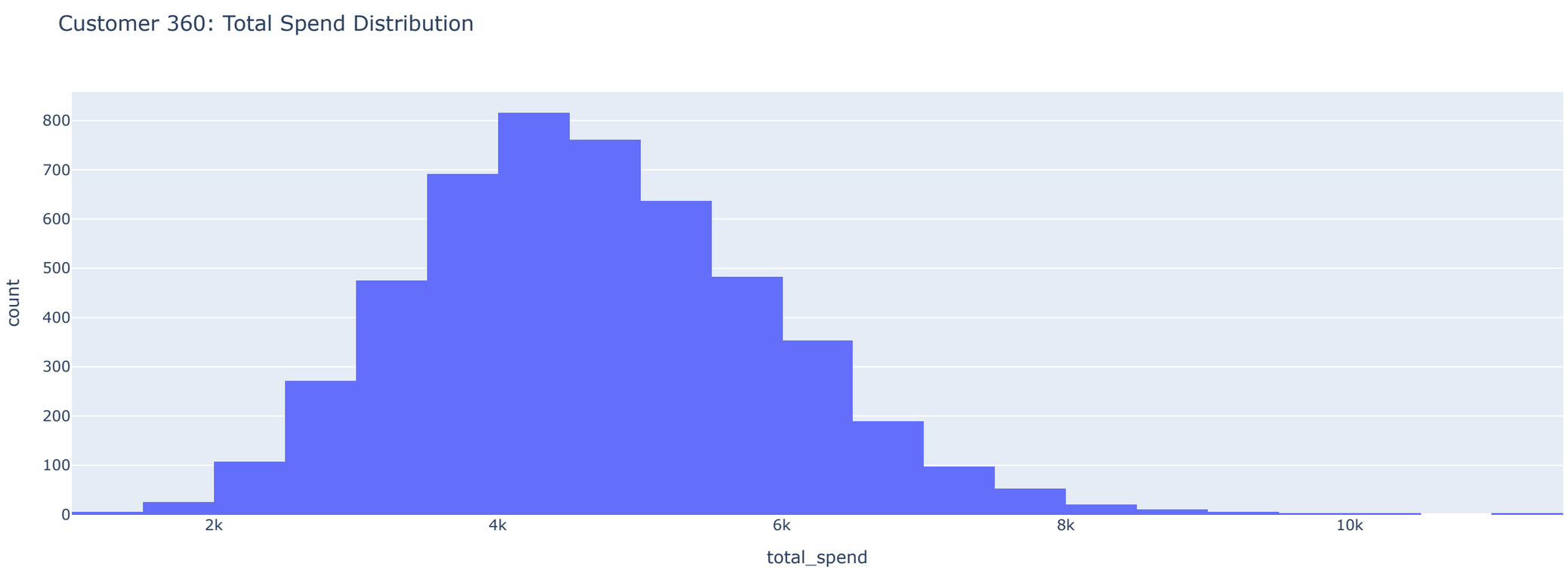
```
customers_pd = pd.DataFrame([
    {
        "customer_id": i,
        "name": fake.name(),
        "email": fake.email(),
        "country": fake.country_code(),
        "signup_date": fake.date_between(start_date='-4y', end_date='today'),
        "segment": random.choice(["consumer", "small_business", "enterprise"]),
        "is_active": random.choices([True, False], weights=[0.92, 0.08])[0]
    }
    for i in range(1, 5001)
])
customers = spark.createDataFrame(customers_pd)
customers.write.format("delta").mode("overwrite").save(f"{BASE_DELTA}/customers")
products_pd = pd.DataFrame([
    {
        "product_id": i,
        "product_name": f"{fake.word().title()} {i}",
        "category": random.choice(["software", "hardware", "subscription", "service", "accessory"]),
        "price": round(abs(np.random.normal(120, 60)) + 5, 2),
        "launch_date": fake.date_between(start_date='-6y', end_date='today')
    }
    for i in range(1, 801)
])
products = spark.createDataFrame(products_pd)
products.write.format("delta").mode("overwrite").save(f"{BASE_DELTA}/products")
sales_pd = pd.DataFrame([
    {
```

```
        "transaction_id": i,
        "customer_id": random.randint(1, 5000),
        "product_id": random.randint(1, 800),
        "quantity": random.choices([1, 2, 3, 4, 5], weights=[0.7, 0.15, 0.08, 0.05, 0.02])[0],
        "unit_price": round(abs(np.random.normal(120, 60)) + 5, 2),
        "ts": fake.date_time_between(start_date='-2y', end_date='now')
    }
    for i in range(1, 120001)
])
sales_pd["total_price"] = sales_pd["quantity"] * sales_pd["unit_price"]
sales = spark.createDataFrame(sales_pd)
sales.write.format("delta").mode("overwrite").save(f"{BASE_DELTA}/sales")

curated_customers = spark.read.format("delta").load(f"{BASE_DELTA}/customers")
curated_sales = spark.read.format("delta").load(f"{BASE_DELTA}/sales")
customer_spend = curated_sales.groupBy("customer_id").agg(F.sum("total_price").alias("total_spend"))
customer_360 = curated_customers.join(customer_spend, on="customer_id", how="left").fillna({"total_spend": 0})
customer_360.write.format("delta").mode("overwrite").save(f"{BASE_DELTA}/customer_360")
print("✅ Created data product: customer_360")
```

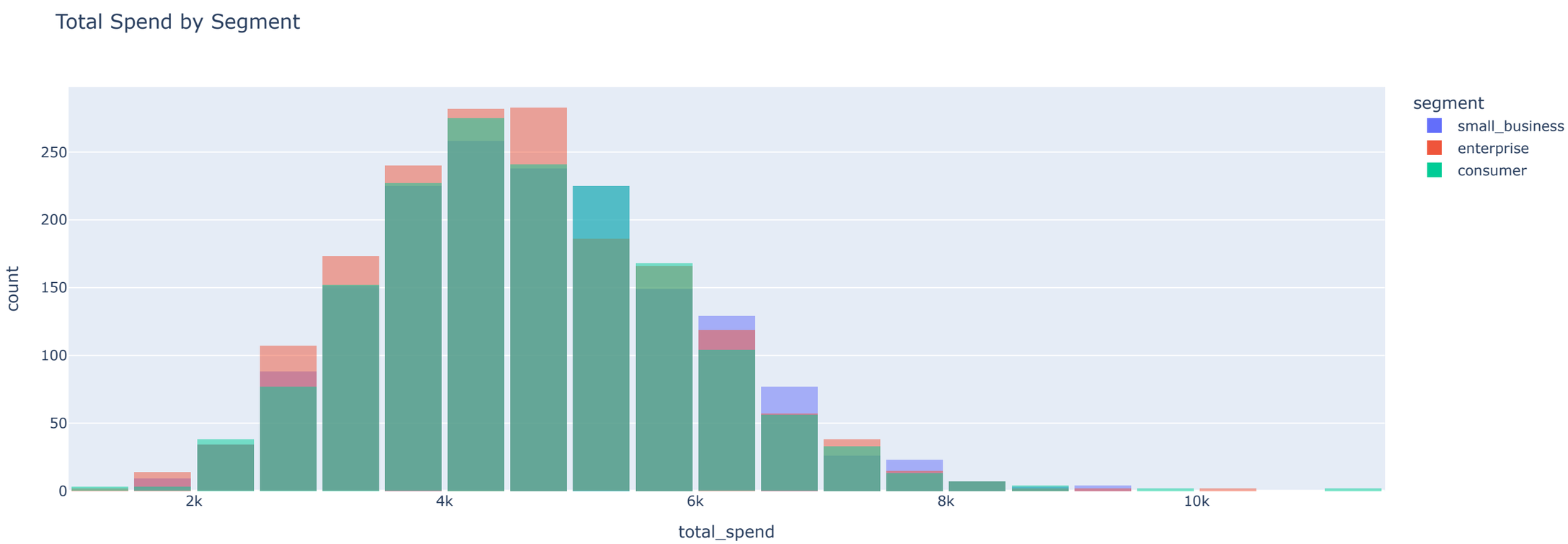
```
Requirement already satisfied: delta-spark in /usr/local/lib/python3.12/dist-packages (3.1.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: faker in /usr/local/lib/python3.12/dist-packages (37.8.0)
Requirement already satisfied: pyspark<3.6.0,>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from delta-spark) (3.5.0)
Requirement already satisfied: importlib-metadata>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from delta-spark) (8.7.0)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.12/dist-packages (from importlib-metadata>=1.0.0->delta-spark) (3.23.0)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.12/dist-packages (from pyspark<3.6.0,>=3.5.0->delta-spark) (0.10.9.7)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
✅ Created data product: customer_360
```

visuals from curated products

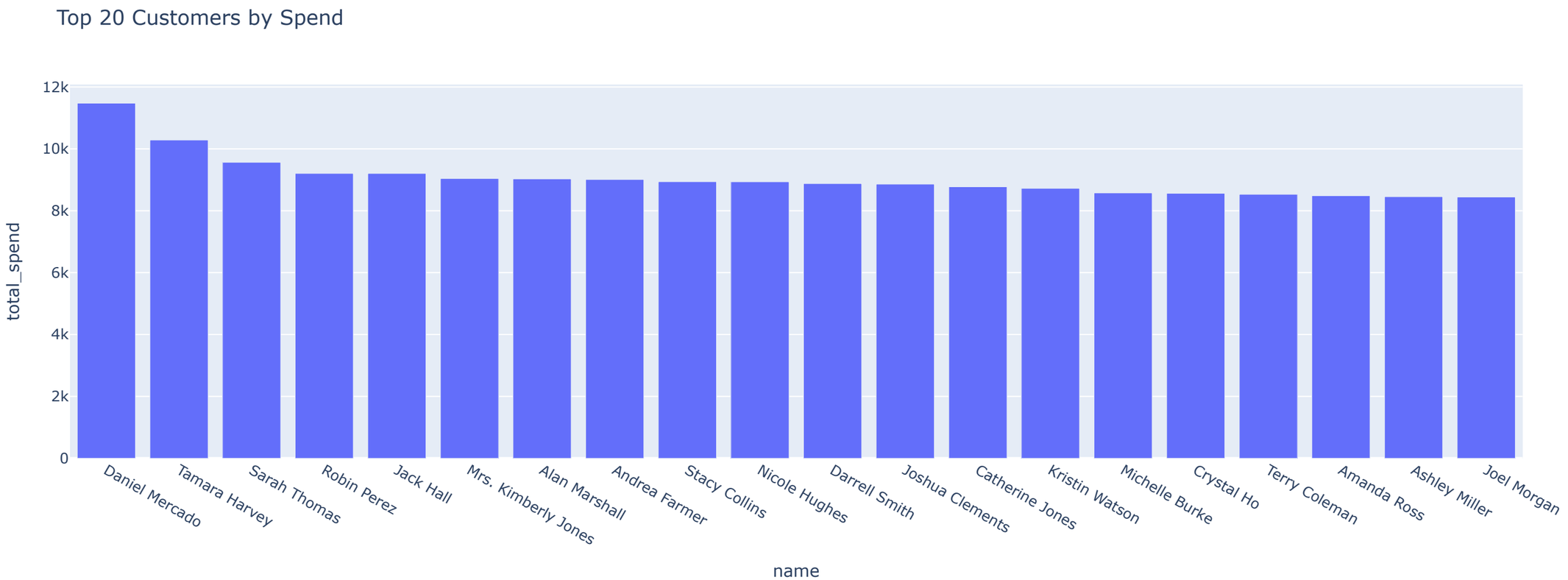
```
import plotly.express as px

pdf_c360 = spark.read.format("delta").load(f"{BASE_DELTA}/customer_360").toPandas()
fig = px.histogram(pdf_c360, x='total_spend', nbins=50, title='Customer 360: Total Spend Distribution')
fig.show()
```




```
fig = px.histogram(
    pdf_c360,
    x='total_spend',
    color='segment',
    nbins=50,
    title='Total Spend by Segment',
    barmode='overlay'
)
fig.update_layout(bargap=0.1)
fig.show()
```




```
top = pdf_c360.sort_values('total_spend', ascending=False).head(20)
fig = px.bar(top, x='name', y='total_spend', title='Top 20 Customers by Spend')
fig.show()
```



Required imports

```
import socket
import threading
import time
import json
import random
from datetime import datetime

customers_pd = customers.toPandas()
products_pd = products.toPandas()
customer_ids = customers_pd['customer_id'].tolist()
product_ids = products_pd['product_id'].tolist()

def start_socket_server(host='127.0.0.1', port=9998, n_events=5000, interval=0.005):
    def run_server():
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.bind((host, port))
        s.listen(1)
        print(f"Socket server listening on {host}:{port}, waiting for consumer...")
        conn, addr = s.accept()
        print("Connected to consumer:", addr)
        for i in range(n_events):
```

```
            ev = {
                "event_id": i,
                "event_type": random.choice(["sale", "signup"]),
                "ts": datetime.utcnow().isoformat(),
                "customer_id": random.choice(customer_ids),
                "product_id": random.choice(product_ids),
                "quantity": random.choices([1, 2, 3], weights=[0.75, 0.2, 0.05])[0],
                "unit_price": float(products_pd.sample(1)['price'].values[0])
            }
            try:
                conn.sendall((json.dumps(ev) + "\n").encode('utf-8'))
            except BrokenPipeError:
                print("Consumer disconnected.")
                break
            time.sleep(interval)

        conn.close()
        s.close()
        print("Socket server finished producing events.")

    t = threading.Thread(target=run_server, daemon=True)
    t.start()
    return t

start_socket_server(n_events=4000, interval=0.003)
```

```
<Thread(Thread-8 (run_server), started daemon 140481728079424)>
Socket server listening on 127.0.0.1:9998, waiting for consumer...
```

spark structured streaming from socket -> delta

```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DoubleType, TimestampType
from pyspark.sql import functions as F

schema = StructType([
    StructField("event_id", IntegerType(), True),
    StructField("event_type", StringType(), True),
    StructField("ts", StringType(), True),
    StructField("customer_id", IntegerType(), True),
    StructField("product_id", IntegerType(), True),
    StructField("quantity", IntegerType(), True),
    StructField("unit_price", DoubleType(), True)
])

socket_df = spark.readStream.format("socket").option("host","127.0.0.1").option("port",9998).load()
json_df = socket_df.select(F.from_json(F.col("value"), schema).alias("data")).select("data.*").withColumn("ts", F.to_timestamp("ts"))
json_df = json_df.withColumn("total_price", F.round(F.col("quantity") * F.col("unit_price"),2))

stream_output = f"{BASE_DELTA}/stream_sales"
checkpoint = f"{BASE_DELTA}/checkpoints/stream_sales"

query = json_df.writeStream.format("delta").outputMode("append").option("checkpointLocation", checkpoint).start(stream_output)
```

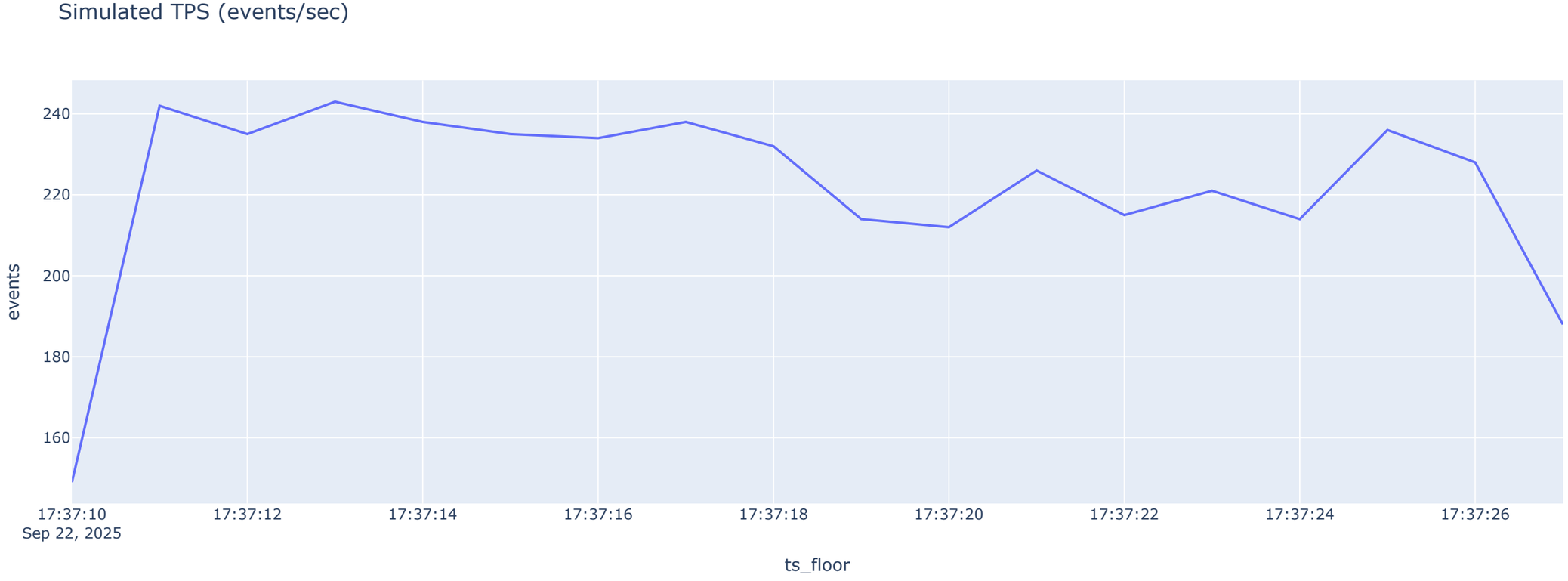
```
print("Started streaming query to", stream_output)
time.sleep(5)
```

```
Started streaming query to /tmp/pp-010/stream_sales
```

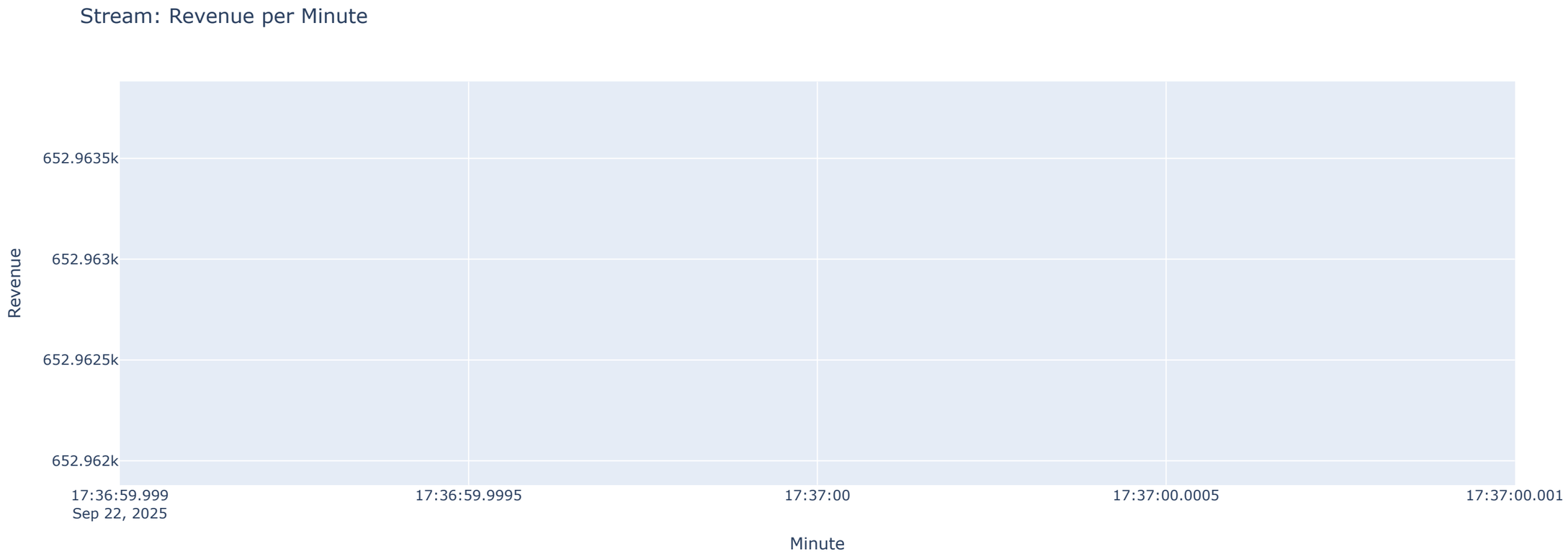
read stream delta into spark/pandas for analytics & TPS plot

```
try:
    stream_df = spark.read.format("delta").load(f"{BASE_DELTA}/stream_sales")
    pdf_stream = stream_df.toPandas()
    print("Stream events read:", len(pdf_stream))
    # Visual 9: TPS events/sec
    pdf_stream['ts_floor'] = pd.to_datetime(pdf_stream['ts']).dt.floor('S')
    tps = pdf_stream.groupby('ts_floor').size().reset_index(name='events')
    fig = px.line(tps, x='ts_floor', y='events', title='Simulated TPS (events/sec)')
    fig.show()
except Exception as e:
    print("Stream not ready yet:", e)
```

```
Stream events read: 4000
/tmp/ipython-input-2380138940.py:7: FutureWarning:

'S' is deprecated and will be removed in a future version, please use 's' instead.
```



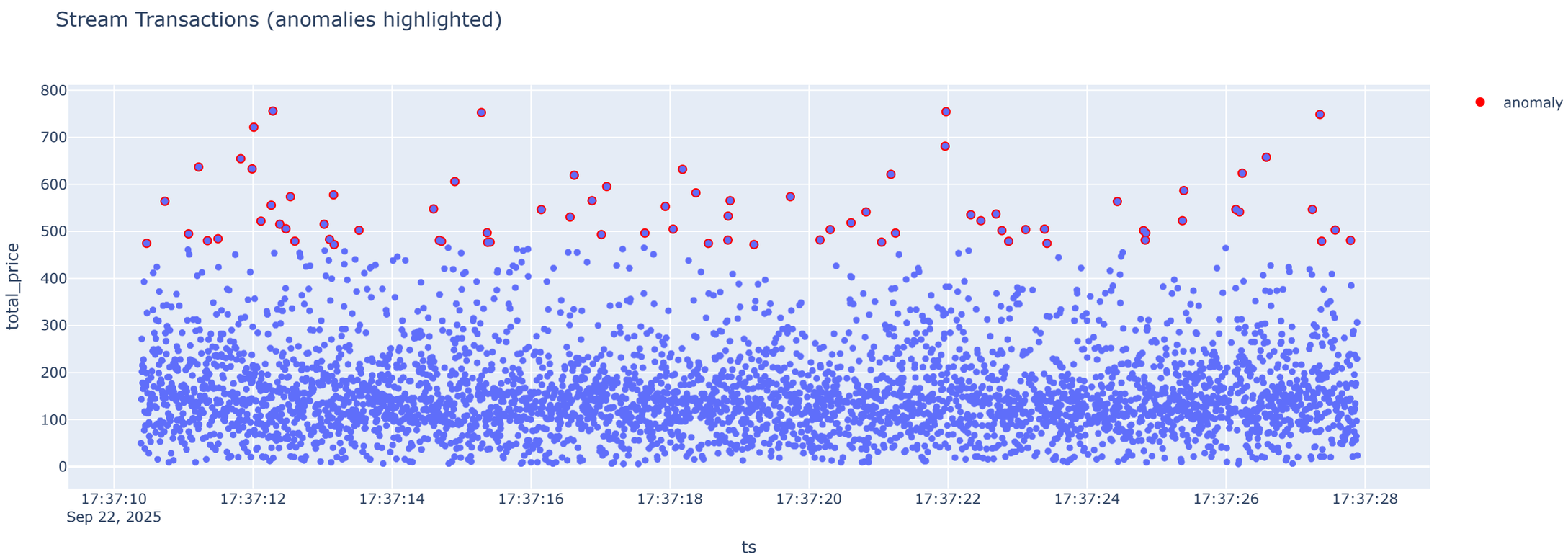
```
import pandas as pd
import plotly.express as px
from pyspark.sql import functions as F

# This aggregation 'agg' is a streaming DataFrame and cannot be saved with .write
# We keep the definition for reference of the streaming logic if needed elsewhere.
agg = json_df \
    .withWatermark("ts", "1 minute") \
    .groupBy(F.window("ts", "1 minute", "30 seconds")) \
    .agg(
        F.sum("total_price").alias("rev"),
        F.count("*").alias("events")
    )
try:

    snapshot = spark.read.format("delta").load(f"{BASE_DELTA}/stream_sales")
    pdf_snap = snapshot.toPandas()
    pdf_snap['ts_minute'] = pd.to_datetime(pdf_snap['ts']).dt.floor('min')
    minute_agg = pdf_snap.groupby('ts_minute')['total_price'].sum().reset_index(name='revenue')

    fig = px.line(minute_agg, x='ts_minute', y='revenue', title='Stream: Revenue per Minute')
    fig.update_layout(xaxis_title='Minute', yaxis_title='Revenue')
    fig.show()

except Exception as e:
    print("Snapshot not available or malformed:", e)
```



crude anomaly detection: transactions with unusually high total_price

```
if 'pdf_stream' in globals() and len(pdf_stream)>0:
    pdf_stream['z'] = (pdf_stream['total_price'] - pdf_stream['total_price'].mean()) / pdf_stream['total_price'].std()
    anomalies = pdf_stream[np.abs(pdf_stream['z'])>3]
    print("Anomalies:", len(anomalies))
    fig = px.scatter(pdf_stream, x='ts', y='total_price', title='Stream Transactions (anomalies highlighted)')
    if len(anomalies)>0:
        fig.add_scatter(x=anomalies['ts'], y=anomalies['total_price'], mode='markers', marker=dict(color='red', size=8), name='anomaly')
    fig.show()
else:
    print("No stream data yet to run anomaly detection")
```

```
Anomalies: 78
```




```
!pip install fastapi==0.103.2 pydantic==1.10.13 uvicorn nest_asyncio
```

Show hidden output

```
!pip install fastapi==0.110.0 pydantic==2.5.2 uvicorn nest_asyncio
```

Show hidden output

FastAPI Interface for Delta Lake Analytics

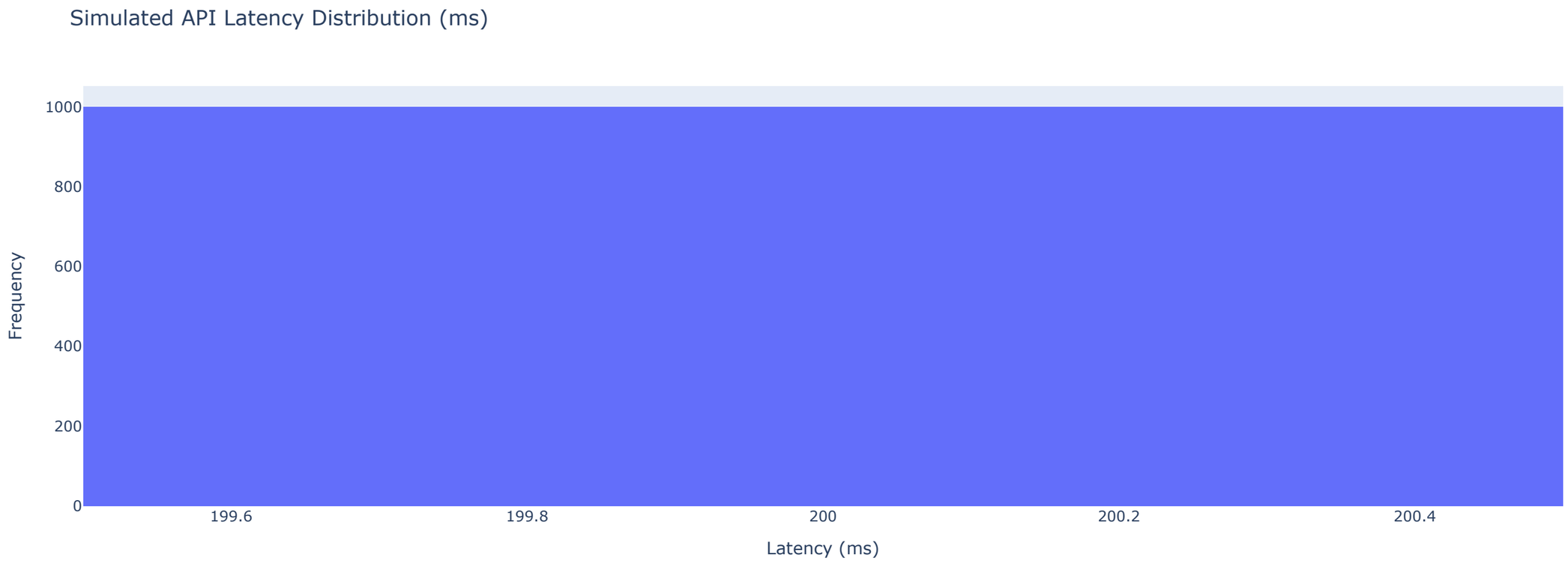
```
import nest_asyncio
import uvicorn
import threading
from fastapi import FastAPI
import pandas as pd

nest_asyncio.apply()

app = FastAPI()

def delta_to_pandas(path, limit=None):
    df = spark.read.format("delta").load(path).toPandas()
    if limit:
        return df.head(limit)
    return df

@app.get("/health")
def health():
    return {"status": "ok"}

@app.get("/api/customer/{cid}")
def get_customer(cid: int):
    df = delta_to_pandas(f"{BASE_DELTA}/customer_360")
    res = df[df['customer_id'] == cid]
    return res.to_dict(orient="records")

@app.get("/api/top_products")
def top_products(limit: int = 10):
    df = delta_to_pandas(f"{BASE_DELTA}/sales")
    agg = df.groupby('product_id')['total_price'].sum().reset_index()
    agg = agg.sort_values('total_price', ascending=False).head(limit)
    return agg.to_dict(orient="records")

def run_api():
    uvicorn.run(app, host="0.0.0.0", port=8000)

threading.Thread(target=run_api, daemon=True).start()
print("✅ FastAPI started on port 8000. Use ngrok if you need a public URL.")
```

```
✅ FastAPI started on port 8000. Use ngrok if you need a public URL.
```

```
import plotly.express as px
```

Simulated API Latency Dashboard with FastAPI & Plotly

```
from fastapi import BackgroundTasks
import plotly.express as px
import pandas as pd
import numpy as np

api_metrics = []

@app.get("/api/sales/summary")
def sales_summary():
    df = delta_to_pandas(f"{BASE_DELTA}/sales")
    tot = df['total_price'].sum()
    cnt = len(df)
    res = {"total_revenue": float(tot), "transactions": int(cnt)}
    api_metrics.append({"endpoint":"/api/sales/summary", "latency_ms": abs(np.random.normal(120, 40))})
    return res

@app.get("/api/product/{pid}/sales")
def product_sales(pid: int):
    df = delta_to_pandas(f"{BASE_DELTA}/sales")
    agg = df[df['product_id'] == pid]['total_price'].sum()
    api_metrics.append({"endpoint":f"/api/product/{pid}/sales", "latency_ms": abs(np.random.normal(90, 30))})
    return {"product_id": pid, "total_revenue": float(agg)}

metrics_df = pd.DataFrame(api_metrics if api_metrics else [{"endpoint": "seed", "latency_ms": 200}] * 1000)
fig = px.histogram(metrics_df, x='latency_ms', nbins=40, title='Simulated API Latency Distribution (ms)')
fig.update_layout(xaxis_title='Latency (ms)', yaxis_title='Frequency')
fig.show()
```

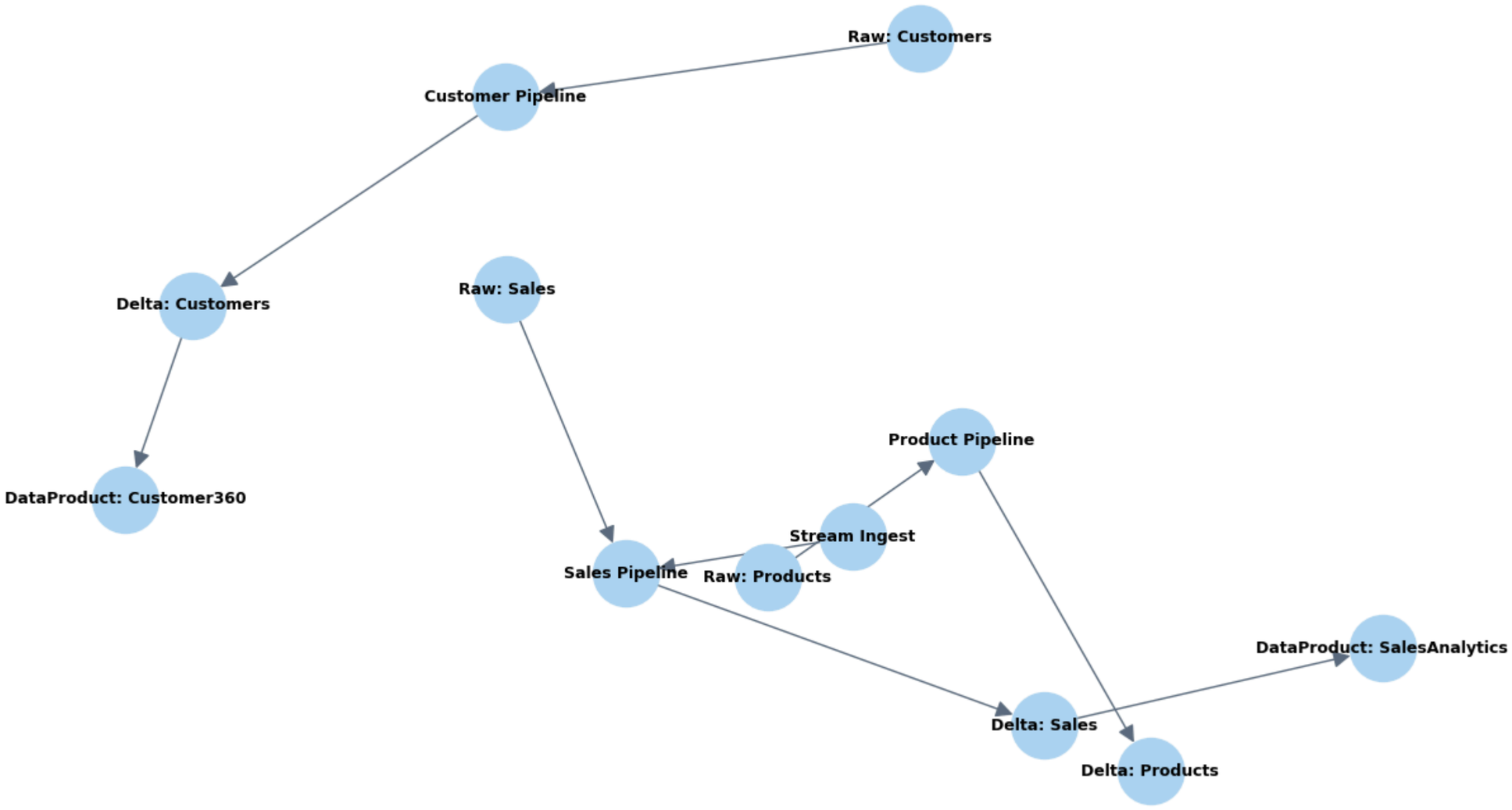


Data Mesh Lineage Visualization with NetworkX

```
import networkx as nx
import matplotlib.pyplot as plt
import os

VIS_DIR = "/tmp/visuals"
os.makedirs(VIS_DIR, exist_ok=True)
from IPython.display import Image, display
def show_image(path):
    display(Image(filename=path))
G = nx.DiGraph()
nodes = [
    "Raw: Sales", "Raw: Customers", "Raw: Products",
    "Sales Pipeline", "Customer Pipeline", "Product Pipeline",
    "Stream Ingest",
    "Delta: Sales", "Delta: Customers", "Delta: Products",
    "DataProduct: SalesAnalytics", "DataProduct: Customer360"
]
G.add_nodes_from(nodes)

edges = [
    ("Raw: Sales", "Sales Pipeline"),
```

```
    ("Raw: Customers", "Customer Pipeline"),
    ("Raw: Products", "Product Pipeline"),
    ("Sales Pipeline", "Delta: Sales"),
    ("Customer Pipeline", "Delta: Customers"),
    ("Product Pipeline", "Delta: Products"),
    ("Stream Ingest", "Sales Pipeline"),
    ("Delta: Sales", "DataProduct: SalesAnalytics"),
    ("Delta: Customers", "DataProduct: Customer360")
]
G.add_edges_from(edges)

plt.figure(figsize=(12, 7))
pos = nx.spring_layout(G, k=0.6, seed=42)
nx.draw(
    G, pos,
    with_labels=True,
    node_size=1400,
    font_size=9,
    arrowsize=20,
    node_color="#AED6F1",
    edge_color="#5D6D7E",
    font_weight="bold"
)
plt.title("📊 Data Mesh Lineage Graph", fontsize=14)
plt.tight_layout()
output_path = f"{VIS_DIR}/14_lineage.png"
plt.savefig(output_path)
show_image(output_path)
```

```
/tmp/ipython-input-3559916282.py:54: UserWarning:

This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.

/tmp/ipython-input-3559916282.py:58: UserWarning:

Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
```



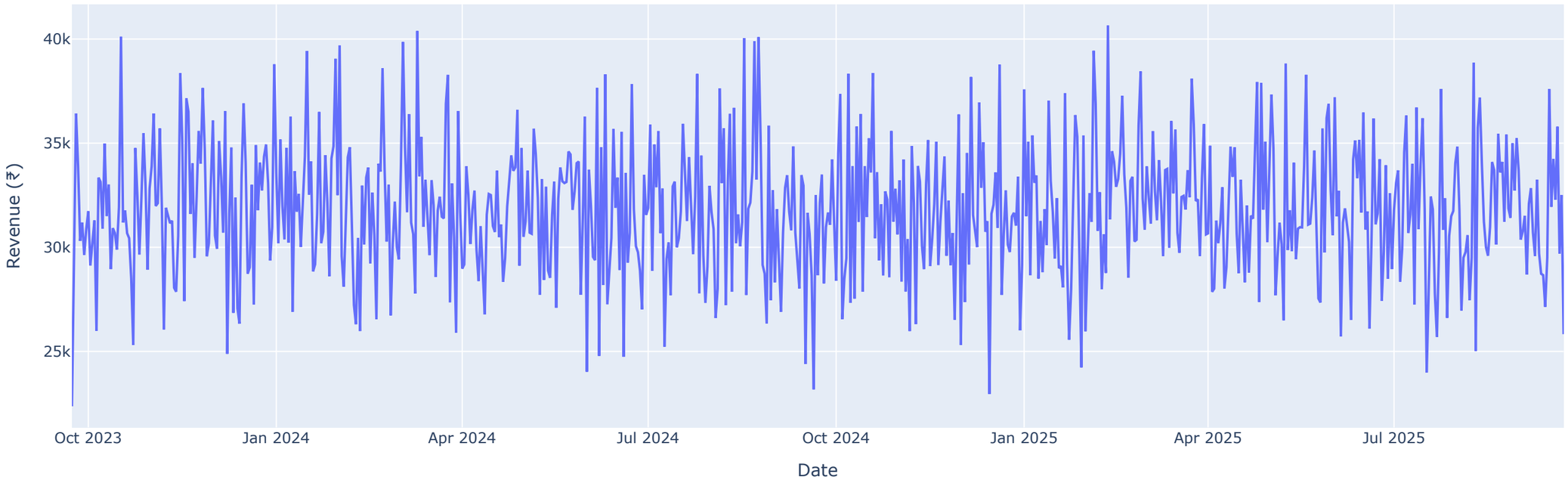

```
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning:

Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
```


Daily Revenue Trends and Rolling Averages from Delta Lake Sales Data

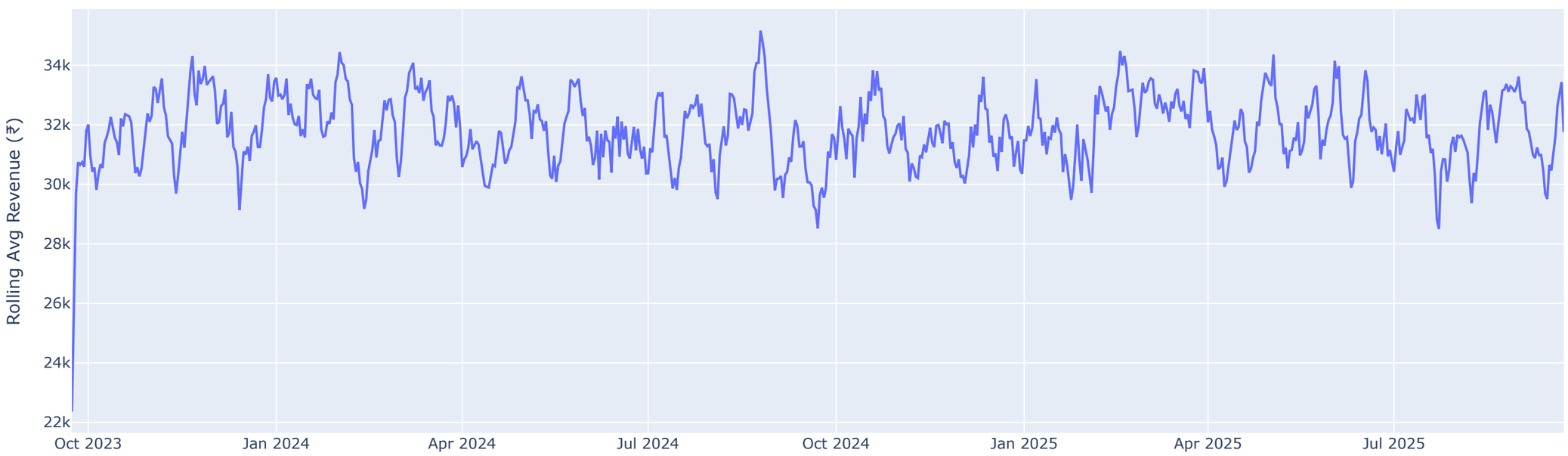
Customer Pipeline

```
import pandas as pd
import numpy as np
import plotly.express as px
from pyspark.sql import SparkSession
from delta import configure_spark_with_delta_pip

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["ARROW_PRE_0_15_IPC_FORMAT"] = "1"
builder = SparkSession.builder \
    .appName("SalesTimeseries") \
    .master("local[*]") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")
spark = configure_spark_with_delta_pip(builder).getOrCreate()
BASE_DELTA = "/tmp/pp-010"
pdf_sales = spark.read.format("delta").load(f"{BASE_DELTA}/sales").toPandas()
pdf_sales['ts'] = pd.to_datetime(pdf_sales['ts'])
daily = pdf_sales.groupby(pdf_sales['ts'].dt.date)['total_price'].sum().reset_index()
daily.columns = ['date', 'total_price']
daily['date'] = pd.to_datetime(daily['date'])
daily = daily.sort_values('date')
daily['rolling_7d'] = daily['total_price'].rolling(window=7, min_periods=1).mean()

fig = px.line(daily, x='date', y='total_price', title='📈 Daily Revenue')
fig.update_layout(xaxis_title='Date', yaxis_title='Revenue (₹)')
fig.show()

fig2 = px.line(daily, x='date', y='rolling_7d', title='📊 7-Day Rolling Avg Revenue')
fig2.update_layout(xaxis_title='Date', yaxis_title='Rolling Avg Revenue (₹)')
fig2.show()
```

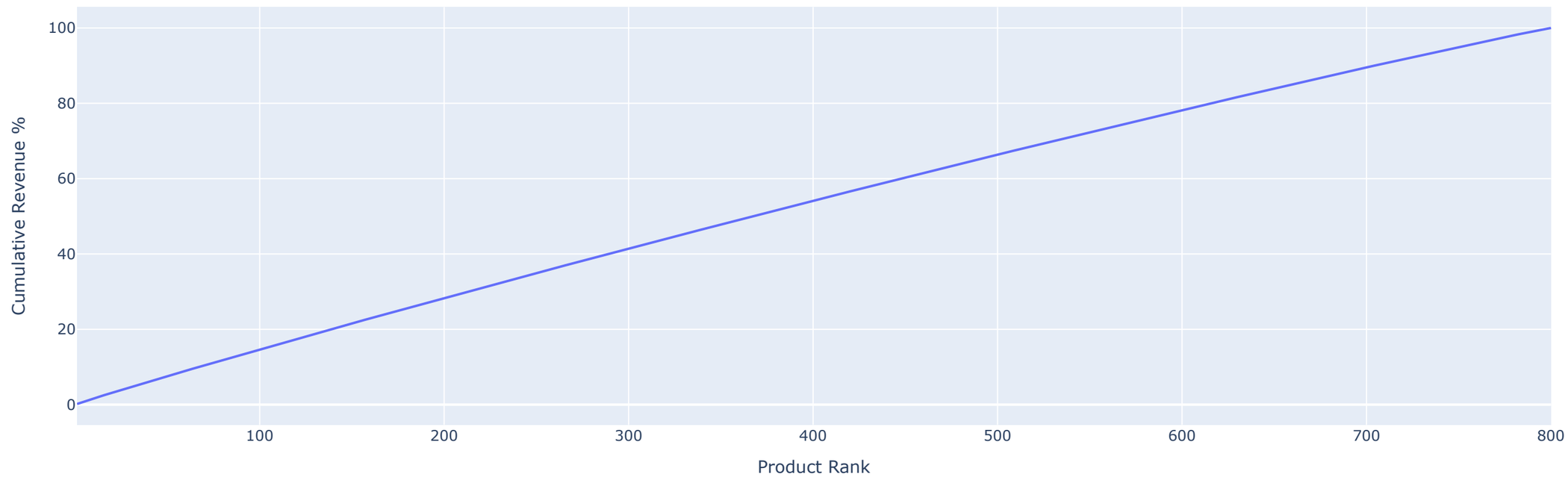
Daily Revenue



7-Day Rolling Avg Revenue



```
import pandas as pd
import plotly.express as px

pdf_sales = spark.read.format("delta").load(f"{BASE_DELTA}/sales").toPandas()
pdf_products = spark.read.format("delta").load(f"{BASE_DELTA}/products").toPandas()
merged = pdf_sales.merge(pdf_products, on='product_id', how='left')

prod_rev = merged.groupby('product_name')['total_price'].sum().reset_index()
prod_rev = prod_rev.sort_values('total_price', ascending=False)
prod_rev['cum_pct'] = prod_rev['total_price'].cumsum() / prod_rev['total_price'].sum() * 100

fig = px.line(
    prod_rev.reset_index(),
    x=prod_rev.reset_index().index + 1,
    y='cum_pct',
    title='Product Revenue Cumulative % (Pareto)',
    labels={'index': 'Product Rank', 'cum_pct': 'Cumulative %'}
)
fig.update_layout(xaxis_title='Product Rank', yaxis_title='Cumulative Revenue %')
fig.show()
```

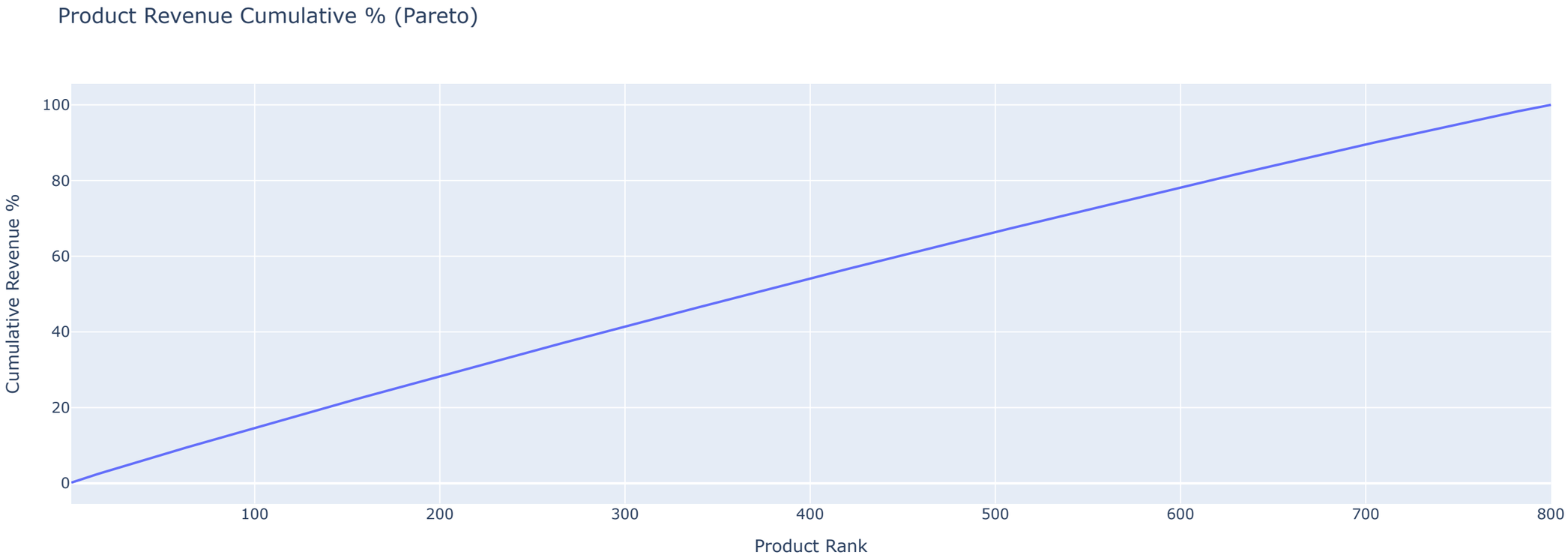


RFM and KMeans segmentation

```
pdf_sales['date'] = pd.to_datetime(pdf_sales['ts'])
ref_date = pdf_sales['date'].max() + pd.Timedelta(days=1)
rfm = pdf_sales.groupby('customer_id').agg(recency=('date', lambda x: (ref_date - x.max()).days),
                                           frequency=('transaction_id', 'count'),
                                           monetary=('total_price','sum')).reset_index()
from sklearn.preprocessing import StandardScaler
X = rfm[['recency','frequency','monetary']].fillna(0)
scaler = StandardScaler()
Xs = scaler.fit_transform(X)
kmeans = KMeans(n_clusters=4, random_state=42).fit(Xs)
rfm['segment'] = kmeans.labels_

fig = px.scatter(rfm, x='recency', y='monetary', size='frequency', color='segment', title='RFM KMeans Segments')
fig.show()
fig2 = px.histogram(rfm, x='segment', title='Customer Segment Counts')
fig2.show()
```

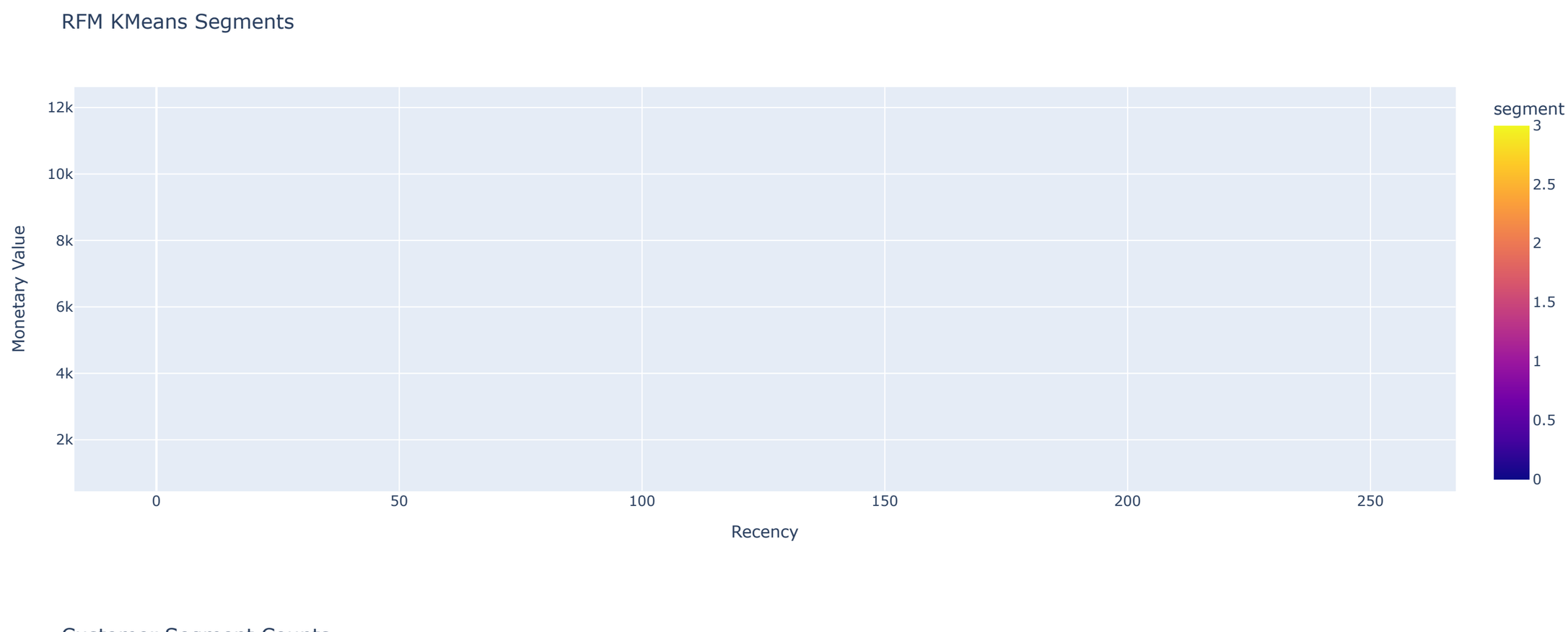
Show hidden output

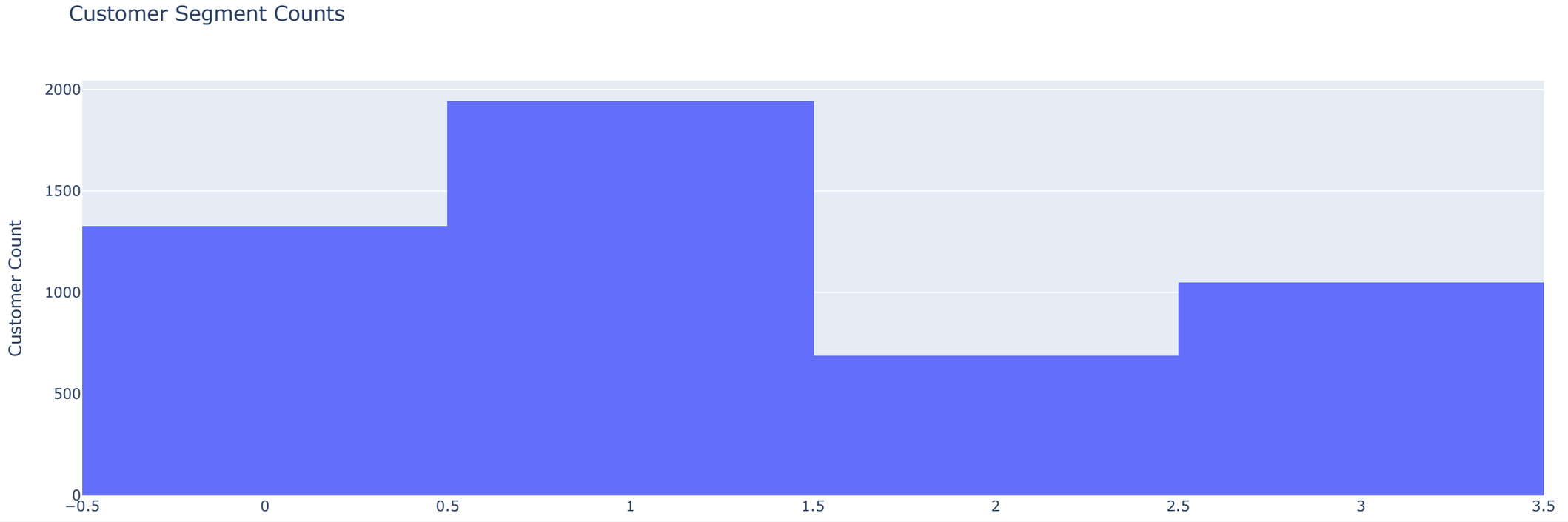
```
import pandas as pd
import plotly.express as px

pdf_sales = spark.read.format("delta").load(f"{BASE_DELTA}/sales").toPandas()
pdf_products = spark.read.format("delta").load(f"{BASE_DELTA}/products").toPandas()
merged = pdf_sales.merge(pdf_products, on='product_id', how='left')

prod_rev = merged.groupby('product_name')['total_price'].sum().reset_index()
prod_rev = prod_rev.sort_values('total_price', ascending=False)
prod_rev['cum_pct'] = prod_rev['total_price'].cumsum() / prod_rev['total_price'].sum() * 100

fig = px.line(
    prod_rev.reset_index(),
    x=prod_rev.reset_index().index + 1,
    y='cum_pct',
    title='Product Revenue Cumulative % (Pareto)',
    labels={'index': 'Product Rank', 'cum_pct': 'Cumulative %'}
)
fig.update_layout(xaxis_title='Product Rank', yaxis_title='Cumulative Revenue %')
fig.show()
```




```
import pandas as pd
import numpy as np
import plotly.express as px
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

pdf_sales['date'] = pd.to_datetime(pdf_sales['ts'])
ref_date = pdf_sales['date'].max() + pd.Timedelta(days=1)

rfm = pdf_sales.groupby('customer_id').agg(
    recency=('date', lambda x: (ref_date - x.max()).days),
    frequency=('transaction_id', 'count'),
    monetary=('total_price', 'sum')
).reset_index()

X = rfm[['recency', 'frequency', 'monetary']].fillna(0)
scaler = StandardScaler()
Xs = scaler.fit_transform(X)
kmeans = KMeans(n_clusters=4, random_state=42)
rfm['segment'] = kmeans.fit_predict(Xs)

fig = px.scatter(
    rfm, x='recency', y='monetary', size='frequency', color='segment',
    title='RFM KMeans Segments',
    labels={'recency': 'Recency (days)', 'monetary': 'Monetary Value (₹)', 'frequency': 'Frequency'}
```

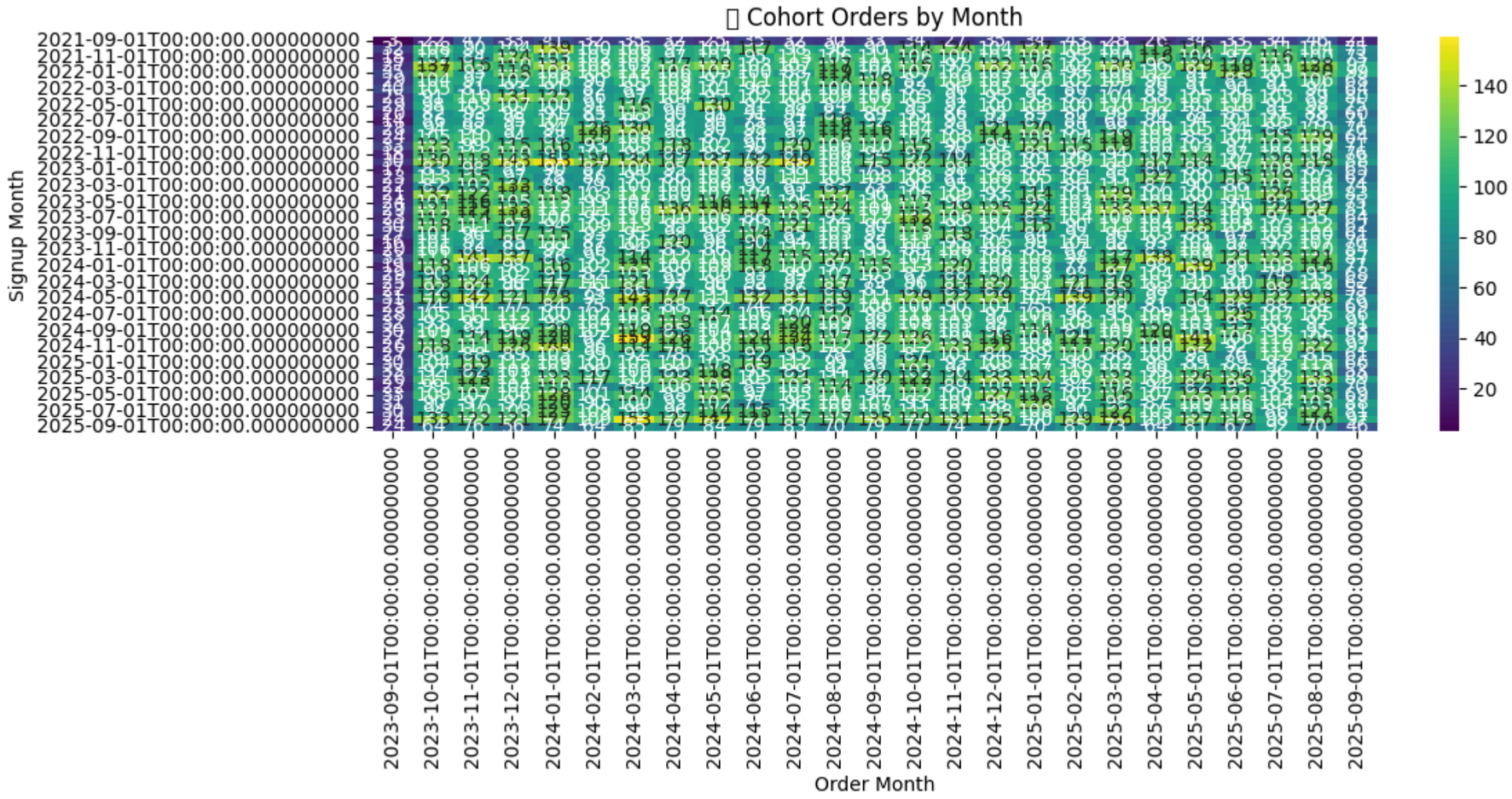
```
)
fig.update_layout(xaxis_title='Recency', yaxis_title='Monetary Value')
fig.show()
fig2 = px.histogram(
    rfm, x='segment', title='Customer Segment Counts',
    labels={'segment': 'Segment', 'count': 'Number of Customers'}
)
fig2.update_layout(xaxis_title='Segment', yaxis_title='Customer Count')
fig2.show()
```


RFM KMeans Segments
Monetary Value
12k
10k
8k
6k
4k
2k
0
50
100
150
200
250
Recency
segment
3
2.5
2
1.5
1
0.5
0

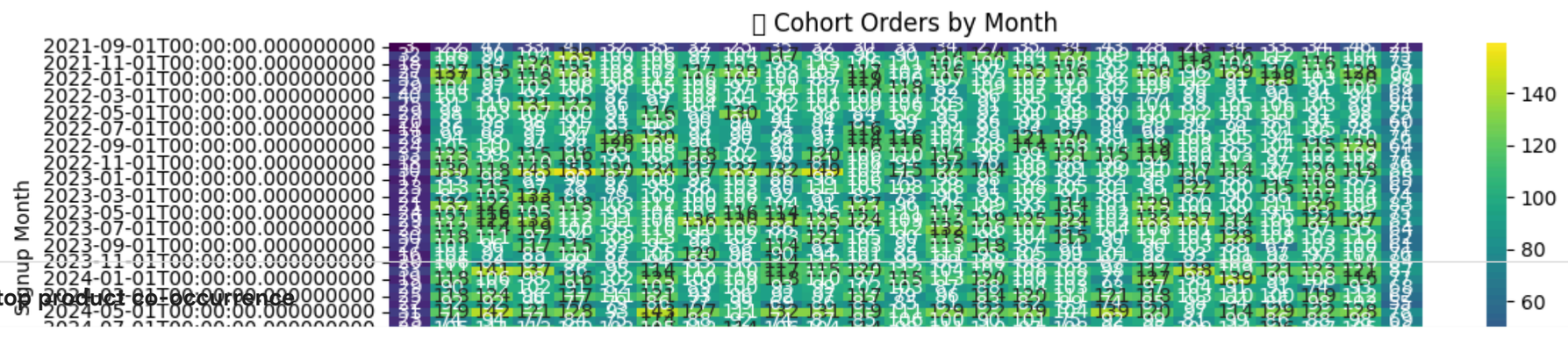

Customer Segment Counts
Customer Count
2000
1500
1000
500
0
−0.5
0
0.5
1
1.5
2
2.5
3
3.5

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

VIS_DIR = "/tmp/visuals"
os.makedirs(VIS_DIR, exist_ok=True)
from IPython.display import Image, display
def show_image(path):
    display(Image(filename=path))

pdf_customers = spark.read.format("delta").load(f"{BASE_DELTA}/customers").toPandas()
pdf_sales = spark.read.format("delta").load(f"{BASE_DELTA}/sales").toPandas()
pdf_customers['signup_month'] = pd.to_datetime(pdf_customers['signup_date']).dt.to_period('M').dt.to_timestamp()
pdf_sales['date'] = pd.to_datetime(pdf_sales['ts'])

pdf_sales2 = pdf_sales.merge(
    pdf_customers[['customer_id', 'signup_month']],
    on='customer_id',
    how='left'
)

cohort = pdf_sales2.groupby(
    ['signup_month', pdf_sales2['date'].dt.to_period('M').dt.to_timestamp()]
)['transaction_id'].nunique().reset_index()

cohort.columns = ['signup_month', 'order_month', 'orders']
pivot = cohort.pivot(index='signup_month', columns='order_month', values='orders').fillna(0)

plt.figure(figsize=(12, 6))
sns.heatmap(pivot, cmap='viridis', annot=True, fmt='g')
plt.title('📊 Cohort Orders by Month')
plt.xlabel('Order Month')
plt.ylabel('Signup Month')
plt.tight_layout()

output_path = f"{VIS_DIR}/21_cohort.png"
plt.savefig(output_path)
show_image(output_path)
```

```
/tmp/ipython-input-2386378136.py:43: UserWarning:

Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.

/tmp/ipython-input-2386378136.py:47: UserWarning:

Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
```

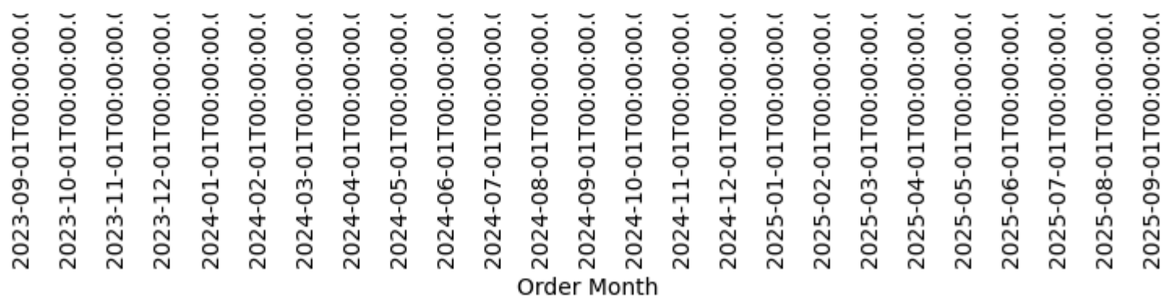


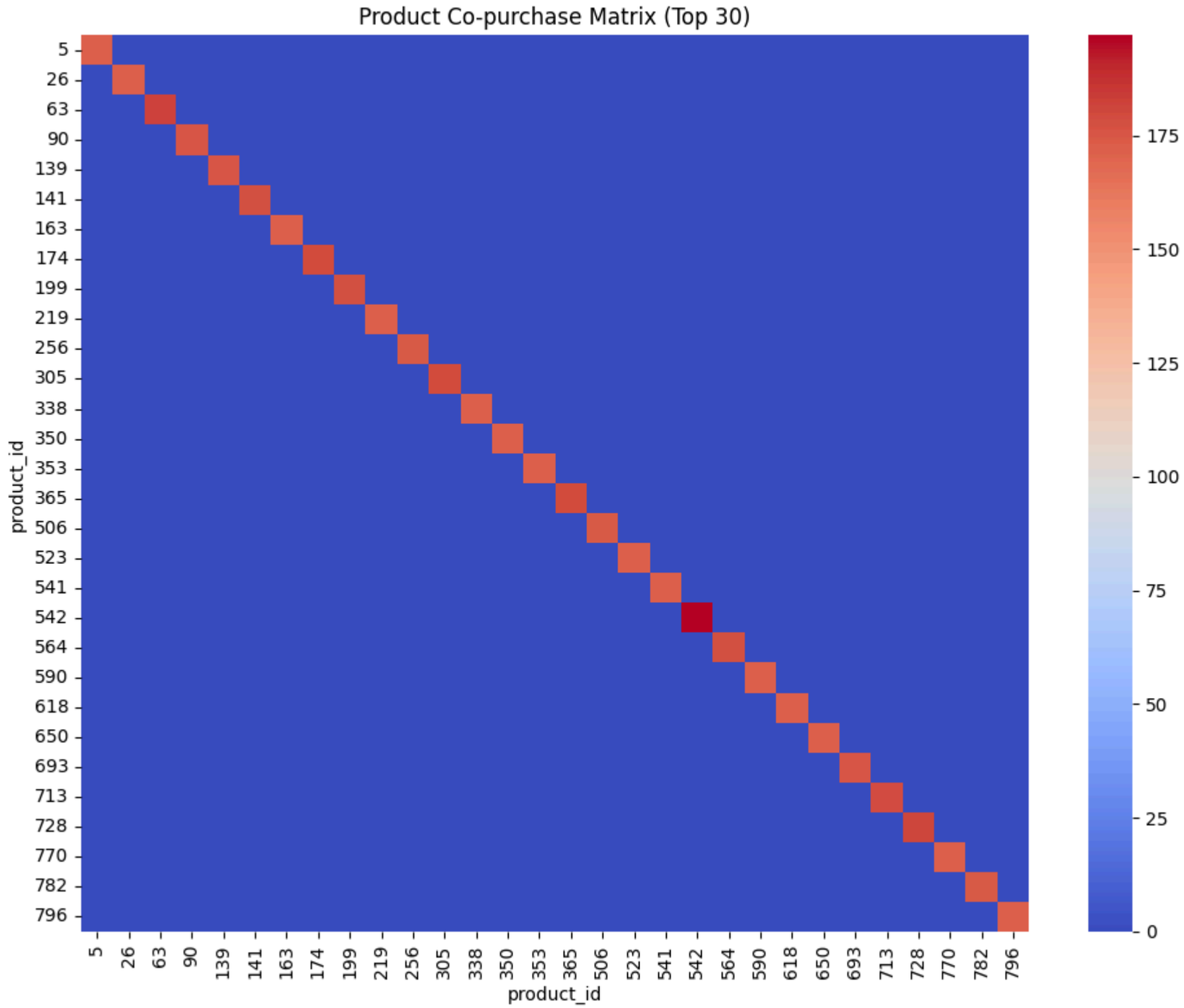

```
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning:

Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.
```



top product co-occurrence

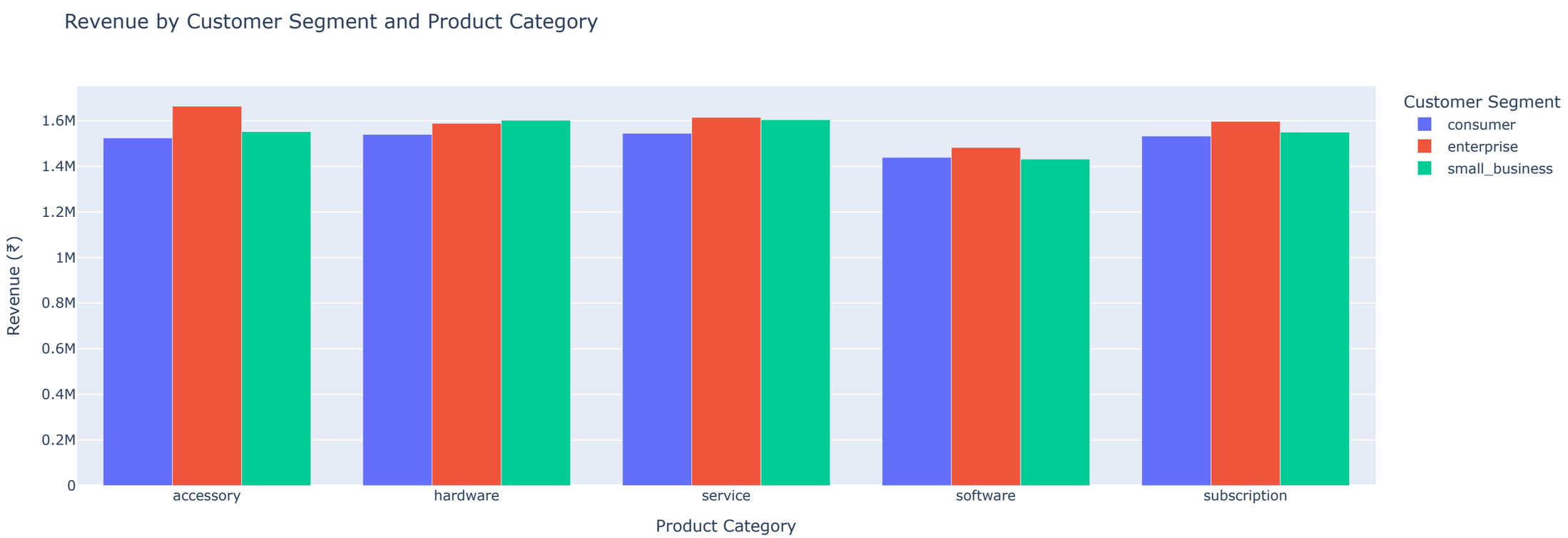

```
top_pids = merged.groupby('product_id')['transaction_id'].count().sort_values(ascending=False).head(30).index.tolist()
sample = merged[merged['product_id'].isin(top_pids)]
pivot = sample.groupby(['transaction_id','product_id']).size().unstack(fill_value=0)
co = pivot.T.dot(pivot)
plt.figure(figsize=(10,8)); sns.heatmap(co, cmap='coolwarm'); plt.title('Product Co-purchase Matrix (Top 30)'); plt.tight_layout()
plt.savefig(f"{VIS_DIR}/22_cooccurrence.png"); show_image(f"{VIS_DIR}/22_cooccurrence.png")
```




Product Co-purchase Matrix (Top 30)
product_id
5
26
63
90
139
141
163
174
199
219
256
305
338
350
353
365
506
523
541
542
564
590
618
650
693
713
728
770
782
796
175
150
125
100
75
50
25
0


Product Co-purchase Matrix (Top 30)
5
26
63
90
139
141
163
175

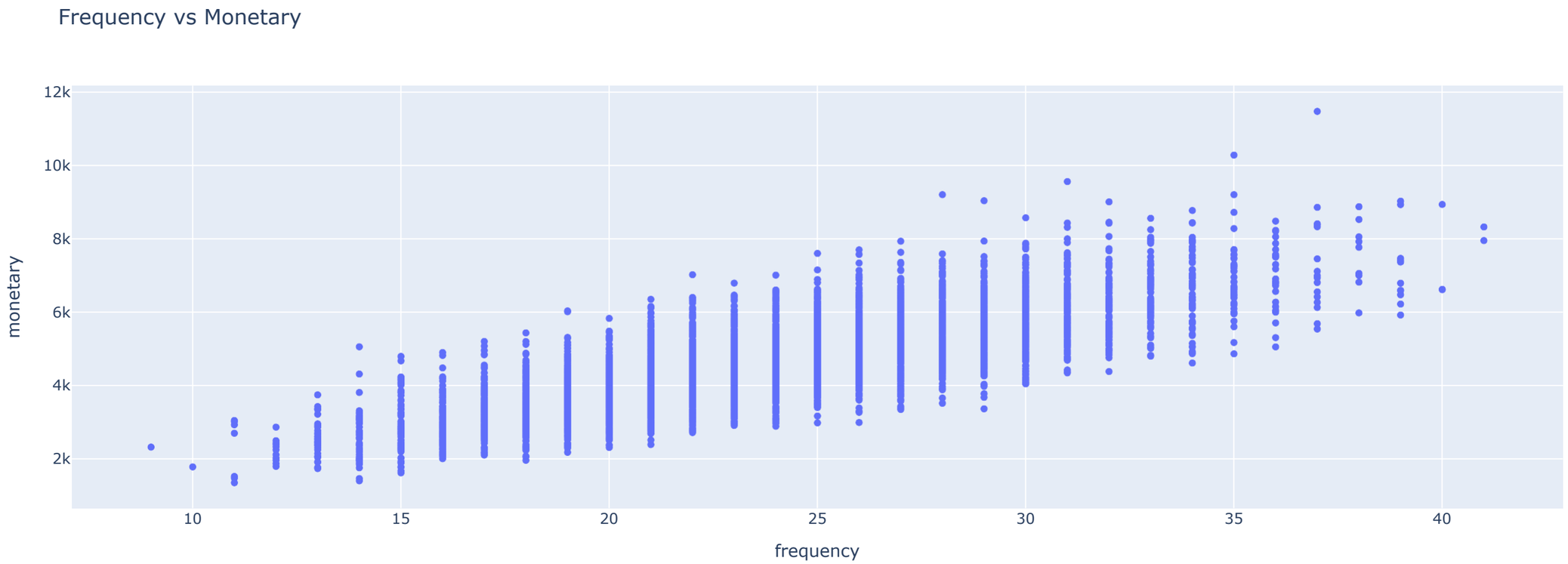
```
import pandas as pd
import plotly.express as px

pd_sales = spark.read.format("delta").load(f"{BASE_DELTA}/sales").toPandas()
pd_customers = spark.read.format("delta").load(f"{BASE_DELTA}/customers").toPandas()
pd_products = spark.read.format("delta").load(f"{BASE_DELTA}/products").toPandas()

df = pd_sales.merge(
    pd_customers[['customer_id', 'segment']],
    on='customer_id',
    how='left'
).merge(
    pd_products[['product_id', 'category']],
    on='product_id',
    how='left'
)

agg = df.groupby(['segment', 'category'])['total_price'].sum().reset_index()

fig = px.bar(
    agg,
    x='category',
    y='total_price',
    color='segment',
    barmode='group',
    title='Revenue by Customer Segment and Product Category',
    labels={'total_price': 'Revenue (₹)', 'category': 'Product Category', 'segment': 'Customer Segment'}
)
fig.update_layout(xaxis_title='Product Category', yaxis_title='Revenue (₹)')
fig.show()
```



> **correlation between frequency and monetary**

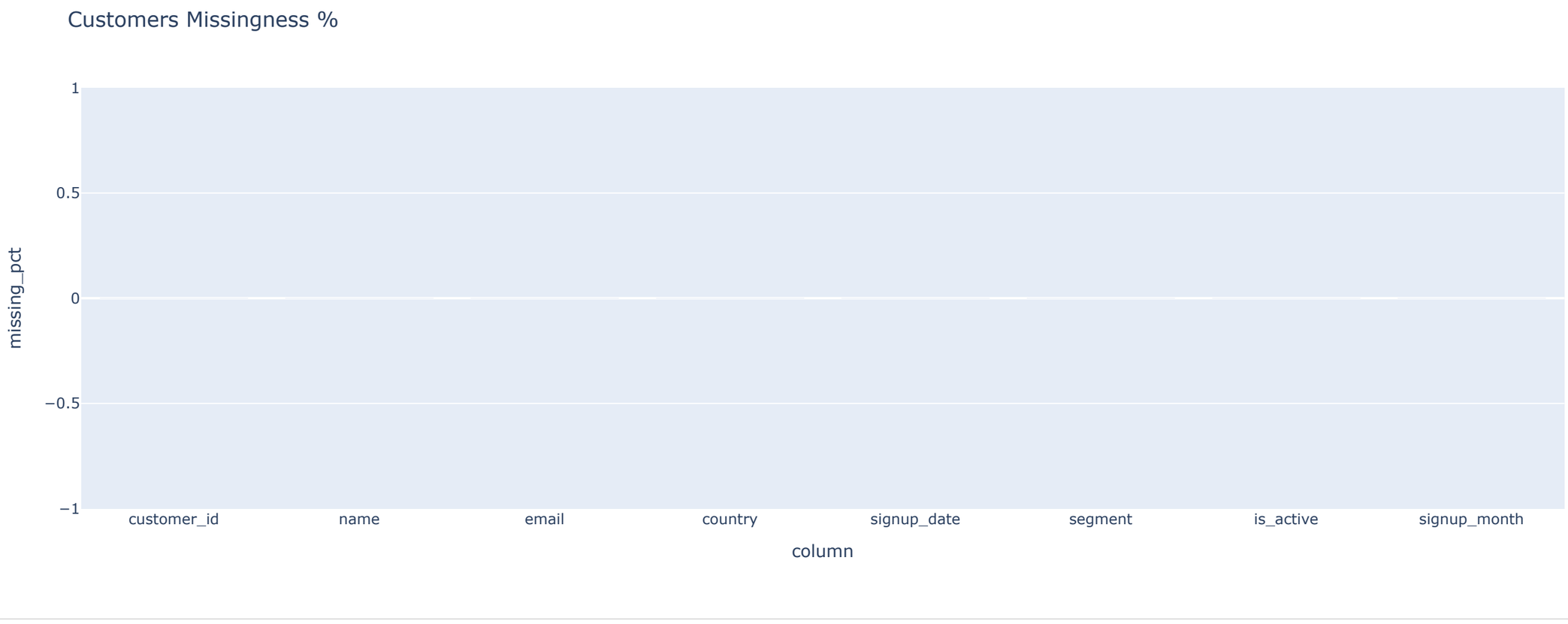
```
rfm_corr = rfm[['frequency','monetary']].corr().iloc[0,1]
print("Correlation frequency vs monetary:", rfm_corr)
fig = px.scatter(rfm, x='frequency', y='monetary', title='Frequency vs Monetary')
fig.show()
```

```
Correlation frequency vs monetary: 0.7848096293294047
```

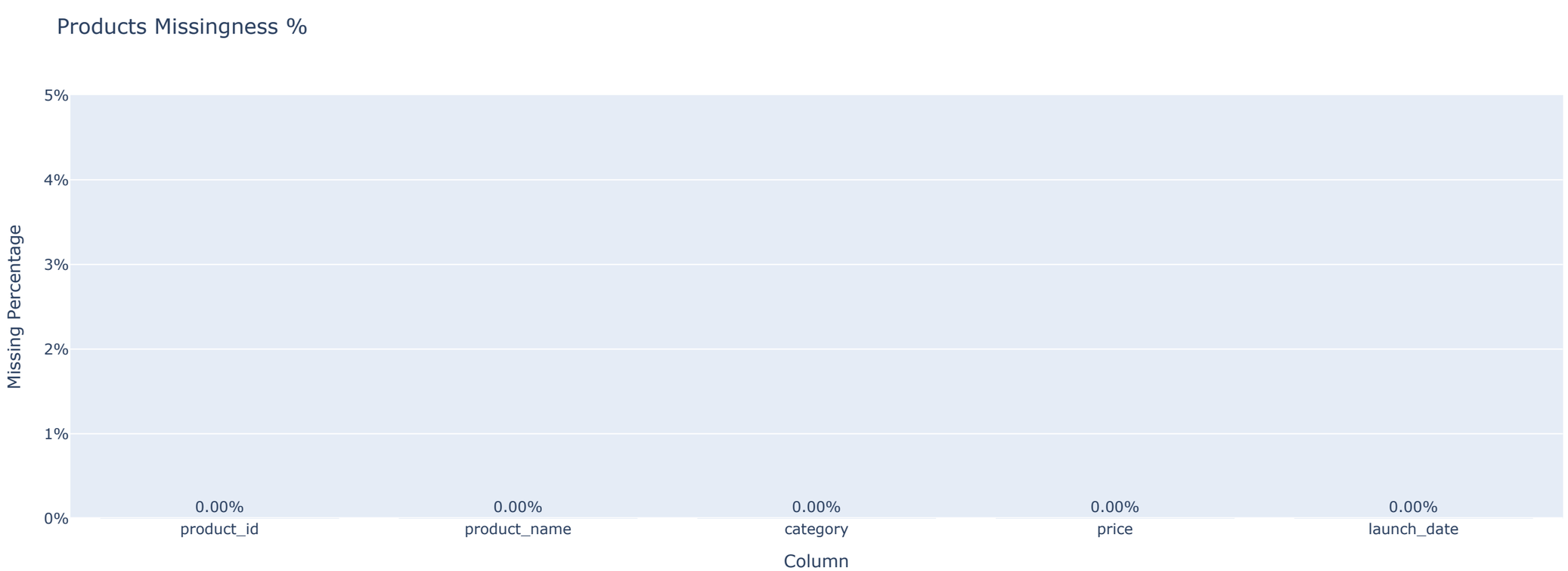


missingness per column in customers and products

```
miss_customers = pdf_customers.isnull().mean().sort_values(ascending=False).reset_index()
miss_customers.columns = ['column','missing_pct']
fig = px.bar(miss_customers, x='column', y='missing_pct', title='Customers Missingness %')
fig.show()
```




```
pdf_products = spark.read.format("delta").load(f"{BASE_DELTA}/products").toPandas()
miss_products = pdf_products.isnull().mean().reset_index()
miss_products.columns = ['column', 'missing_pct']

import plotly.express as px
fig = px.bar(
    miss_products,
    x='column',
    y='missing_pct',
    title='Products Missingness %',
    text=miss_products['missing_pct'].apply(lambda x: f"{x:.2%}")
)
fig.update_traces(textposition='outside')
fig.update_layout(
    yaxis=dict(range=[0, 0.05]),  # Force visible range
    yaxis_tickformat='.0%',
    xaxis_title='Column',
    yaxis_title='Missing Percentage'
)
fig.show()
```




```
miss_only = miss_products[miss_products['missing_pct'] > 0]
if miss_only.empty:
    print("✅ No missing values in products dataset.")
else:
    display(miss_only)
```

```
✅ No missing values in products dataset.
```

API monitoring snapshot (we simulated metrics)

```
if 'metrics_df' not in globals():
    metrics_df = pd.DataFrame([{"endpoint":"seed","latency_ms":abs(np.random.normal(120,30))} for _ in range(300)])
fig = px.box(metrics_df, y='latency_ms', title='API Latency Boxplot')
fig.show()

fig2 = px.histogram(metrics_df, x='latency_ms', nbins=40, title='API Latency Histogram')
fig2.show()
```


API Latency Boxplot
201
200.5
200
latency_ms