🛒 E-Commerce Sales Data Analysis Project

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

pd.set_option("display.max_columns", None)
sns.set_style("whitegrid")
```

In [2]:
```
customers = pd.read_csv("customers.csv", parse_dates=["signup_date"])
products = pd.read_csv("products.csv")
orders = pd.read_csv("orders.csv", parse_dates=["order_date"])
payments = pd.read_csv("payments.csv")
```

Data Exploration

In [3]:
```
customers.head()
```

Out[3]:

	customer_id	customer_name	city	signup_date
0	1	Customer_1	Hyderabad	2022-04-07
1	2	Customer_2	Chennai	2023-09-20
2	3	Customer_3	Bangalore	2022-10-18
3	4	Customer_4	Chennai	2024-06-09
4	5	Customer_5	Chennai	2023-04-07

In [4]:
```
products.head()
```

Out[4]:

	product_id	product_name	category	price
0	1	Product_1	Sports	4088.45
1	2	Product_2	Home	3114.58
2	3	Product_3	Sports	1600.64
3	4	Product_4	Home	3157.09
4	5	Product_5	Clothing	2682.50

In [5]:
```
orders.head()
```

Out[5]:

	order_id	customer_id	product_id	quantity	order_date
0	1	1106	269	2	2023-06-05
1	2	1605	186	2	2023-07-17
2	3	1155	131	3	2023-07-09
3	4	1623	291	1	2023-12-19
4	5	1754	277	4	2023-04-12

In [6]:
```
payments.head()
```

Out[6]:

	payment_id	order_id	payment_method	payment_status
0	1	1	Net Banking	Failed
1	2	2	Net Banking	Pending
2	3	3	Card	Pending
3	4	4	COD	Pending
4	5	5	Net Banking	Failed

In [7]:
```
print(customers.shape)
print(products.shape)
print(orders.shape)
print(payments.shape)
```

```
(2000, 4)
(300, 4)
(10000, 5)
(10000, 4)
```

Data Cleaning

In [8]:
```
customers.isnull().sum()
products.isnull().sum()
orders.isnull().sum()
payments.isnull().sum()

customers.drop_duplicates(inplace=True)
orders.drop_duplicates(inplace=True)
```

Merge All Tables

In [9]:
```
df = orders.merge(customers, on="customer_id") \
           .merge(products, on="product_id") \
           .merge(payments, on="order_id")

df.head()
```

Out[9]:

	order_id	customer_id	product_id	quantity	order_date	customer_name	city	signup_dat
0	1	1106	269	2	2023-06-05	Customer_1106	Chennai	2023-06-1
1	9702	1881	269	1	2023-04-28	Customer_1881	Bangalore	2024-09-1
2	4231	1703	269	4	2023-03-27	Customer_1703	Hyderabad	2024-03-2
3	2701	1679	269	3	2023-02-08	Customer_1679	Mumbai	2022-09-1
4	8809	1321	269	4	2023-09-18	Customer_1321	Mumbai	2022-08-0

Feature Engineering

In [10]:
```
df["revenue"] = df["quantity"] * df["price"]

df["year"] = df["order_date"].dt.year
df["month"] = df["order_date"].dt.month
df["day"] = df["order_date"].dt.day
df["weekday"] = df["order_date"].dt.day_name()
```

KPI

In [11]:
```
total_revenue = df["revenue"].sum()
total_orders = df["order_id"].nunique()
active_customers = df["customer_id"].nunique()

print("Total Revenue:", total_revenue)
print("Total Orders:", total_orders)
print("Active Customers:", active_customers)

success_rate = df["payment_status"].value_counts(normalize=True) * 100
print(success_rate)
```

```
Total Revenue: 64463223.559999995
Total Orders: 10000
Active Customers: 1996
payment_status
Pending      33.84
Failed       33.50
Completed    32.66
Name: proportion, dtype: float64
```

Revenue Analysis

Top 10 Customers

In [12]:
```
top_customers = df.groupby("customer_name")["revenue"] \
                   .sum() \
                   .sort_values(ascending=False) \
                   .head(10)

top_customers
```

Out[12]:
```
customer_name
Customer_1099    113841.12
Customer_1097    109506.64
Customer_639     108831.79
Customer_1231    104540.08
Customer_101     100588.88
Customer_1212     96804.40
Customer_605      96556.96
Customer_1969     94608.99
Customer_947      93000.90
Customer_618      87663.12
Name: revenue, dtype: float64
```

Category Wise Revenue

In [13]:
```
category_revenue = df.groupby("category")["revenue"] \
                      .sum() \
                      .sort_values(ascending=False)

category_revenue
```

Out[13]:
```
category
Sports         17713580.12
Clothing       17478233.29
Electronics    15173760.11
Home           14097650.04
Name: revenue, dtype: float64
```

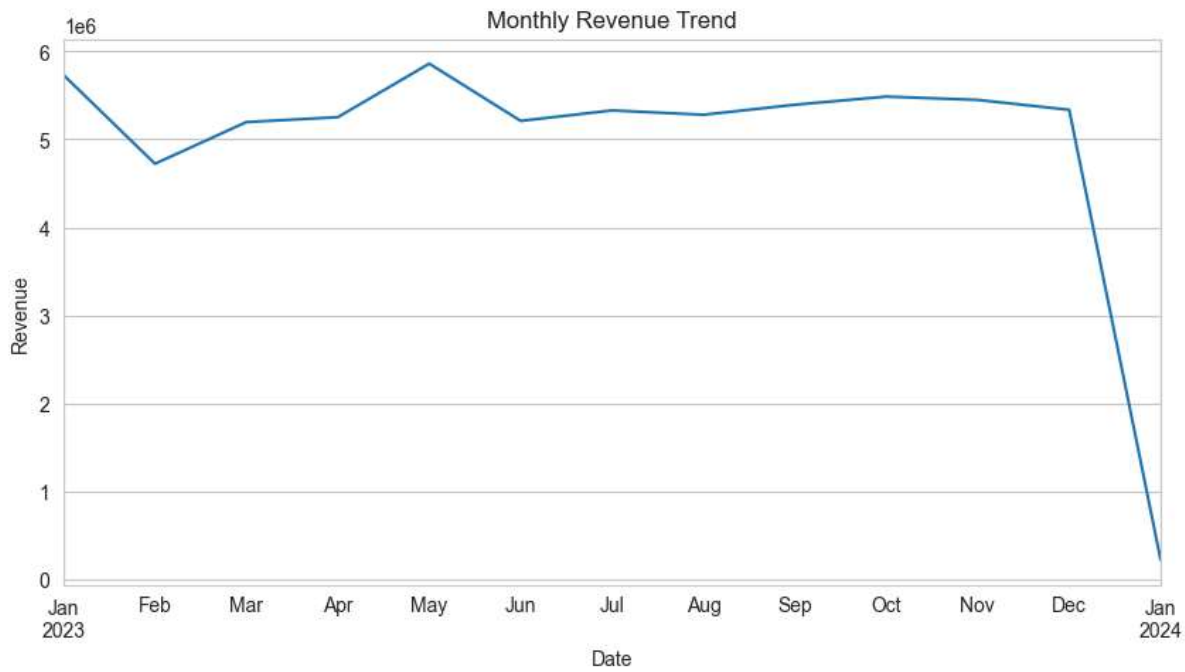
City Wise Revenue

In [14]:
```
city_revenue = df.groupby("city")["revenue"] \
                  .sum() \
                  .sort_values(ascending=False)

city_revenue
```

Out[14]:
```
city
Hyderabad    13501097.69
Delhi        13368779.20
Chennai      12755514.78
Bangalore    12549270.70
Mumbai       12288561.19
Name: revenue, dtype: float64
```

Time Series Analysis

In [15]:
```
monthly_revenue = df.groupby(pd.Grouper(key="order_date", freq="M"))["revenue"].sum

plt.figure(figsize=(10,5))
monthly_revenue.plot()
plt.title("Monthly Revenue Trend")
plt.xlabel("Date")
plt.ylabel("Revenue")
plt.show()
```



Statistical Analysis

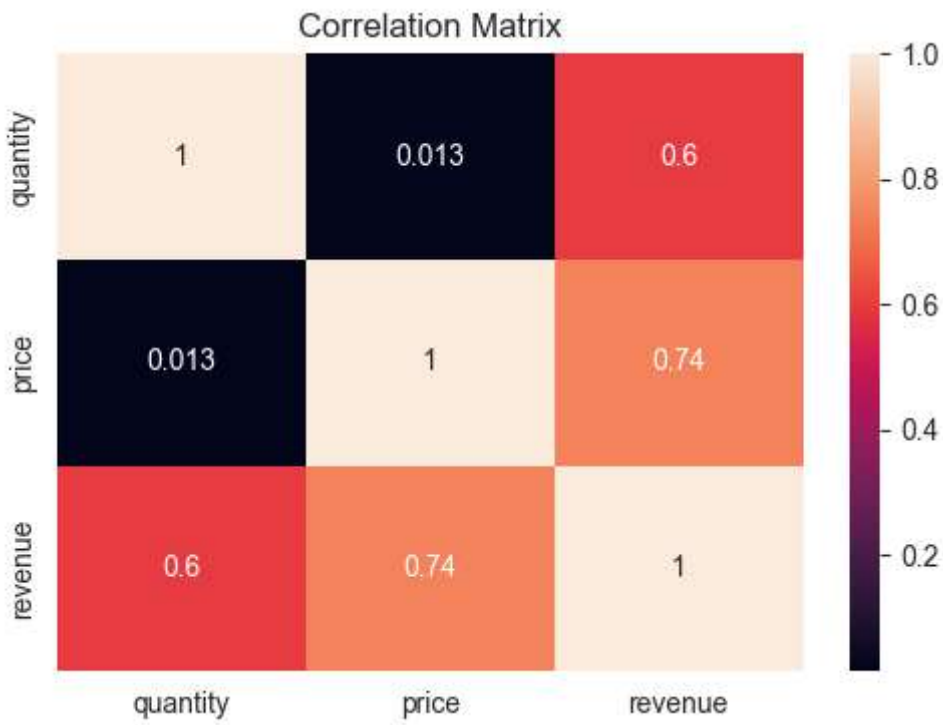
Descriptive Statistics

In [16]:
```
df["revenue"].describe()
```

Out[16]:
```
count    10000.000000
mean      6446.322356
std       4858.909739
min        115.770000
25%       2505.720000
50%       4948.170000
75%       9471.270000
max      19967.600000
Name: revenue, dtype: float64
```

Correlation Analysis

In [17]:
```
plt.figure(figsize=(6,4))
sns.heatmap(df[["quantity","price","revenue"]].corr(), annot=True)
plt.title("Correlation Matrix")
plt.show()
```



Hypothesis Testing (Card vs UPI Revenue)

In [18]:
```
card_revenue = df[df["payment_method"]=="Card"]["revenue"]
upi_revenue = df[df["payment_method"]=="UPI"]["revenue"]

t_stat, p_value = stats.ttest_ind(card_revenue, upi_revenue)

print("T-Statistic:", t_stat)
print("P-Value:", p_value)
```

```
T-Statistic: -0.5389778869910017
P-Value: 0.5899260659429982
```

Customer Segmentation

In [19]:
```
customer_spend = df.groupby("customer_id")["revenue"].sum()

high_value = customer_spend[customer_spend > customer_spend.mean()]

print("High Value Customers:", len(high_value))
```

```
High Value Customers: 901
```

Management Pivot Report

In [20]:
```
pivot = df.pivot_table(
    values="revenue",
    index="category",
    columns="payment_method",
    aggfunc="sum"
)

pivot
```

Out[20]:

payment_method category	COD	Card	Net Banking	UPI
Clothing	4272934.96	4074036.35	4660976.96	4470285.02
Electronics	3453061.03	4112828.30	3630891.46	3976979.32
Home	3939375.82	3194139.82	3369269.46	3594864.94
Sports	4581081.55	3925302.46	4351526.52	4855669.59

Export Final Dataset

In [21]:
```
df.to_csv("final_cleaned_dataset.csv", index=False)
```

In [ ]:

Database Integration

In [22]:
```
import mysql.connector
```

In [23]:
```
connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Ravi@mysql"
)
```

```
cursor = connection.cursor()

print("Connected to MySQL Successfully")
```

```
Connected to MySQL Successfully
```

create database

In [24]:
```
cursor.execute("CREATE DATABASE IF NOT EXISTS ecommerce_db")
cursor.execute("USE ecommerce_db")

print("Database Ready")
```

```
Database Ready
```

Create Tables

In [25]:
```
cursor.execute("""
CREATE TABLE IF NOT EXISTS customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(100),
    city VARCHAR(100),
    signup_date DATE
)
""")

cursor.execute("""
CREATE TABLE IF NOT EXISTS products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(100),
    price FLOAT
)
""")

cursor.execute("""
CREATE TABLE IF NOT EXISTS orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    product_id INT,
    quantity INT,
    order_date DATE
)
""")

cursor.execute("""
CREATE TABLE IF NOT EXISTS payments (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    payment_method VARCHAR(50),
    payment_status VARCHAR(50)
)
```

```
""")

print("Tables Created Successfully")
```

```
Tables Created Successfully
```

Insert Data into MySQL Tables

In [26]:
```
for _, row in customers.iterrows():
    cursor.execute("""
    INSERT IGNORE INTO customers VALUES (%s, %s, %s, %s)
    """, tuple(row))

for _, row in products.iterrows():
    cursor.execute("""
    INSERT IGNORE INTO products VALUES (%s, %s, %s, %s)
    """, tuple(row))

for _, row in orders.iterrows():
    cursor.execute("""
    INSERT IGNORE INTO orders VALUES (%s, %s, %s, %s, %s)
    """, tuple(row))

for _, row in payments.iterrows():
    cursor.execute("""
    INSERT INTO payments (order_id, payment_method, payment_status)
    VALUES (%s, %s, %s)
    """, (row["order_id"], row["payment_method"], row["payment_status"]))

connection.commit()

print("Data Inserted Successfully")
```

```
Data Inserted Successfully
```

SQL Data Analysis

Total Revenue

In [27]:
```
query = """
SELECT SUM(o.quantity * p.price) AS total_revenue
FROM orders o
JOIN products p ON o.product_id = p.product_id
"""

cursor.execute(query)
result = cursor.fetchall()

print("Total Revenue (SQL):", result)
```

```
Total Revenue (SQL): [(64463223.564834595,)]
```

Top 5 Cities by Revenue

In [28]:
```
query = """
SELECT c.city, SUM(o.quantity * p.price) AS revenue
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN products p ON o.product_id = p.product_id
GROUP BY c.city
ORDER BY revenue DESC
LIMIT 5
"""

cursor.execute(query)
result = cursor.fetchall()

for row in result:
    print(row)
```

```
('Hyderabad', 13501097.684410095)
('Delhi', 13368779.206954956)
('Chennai', 12755514.777687073)
('Bangalore', 12549270.70916748)
('Mumbai', 12288561.18661499)
```

Payment Success Rate

In [29]:
```
query = """
SELECT payment_status, COUNT(*) AS count
FROM payments
GROUP BY payment_status
"""

cursor.execute(query)
result = cursor.fetchall()

for row in result:
    print(row)
```

```
('Failed', 6700)
('Pending', 6768)
('Completed', 6532)
```

Fetch Data from MySQL into Pandas

In [30]:
```
query = """
SELECT o.order_date, (o.quantity * p.price) AS revenue
FROM orders o
JOIN products p ON o.product_id = p.product_id
"""
```

```
sql_df = pd.read_sql(query, connection)

sql_df.head()
```

```
C:\Users\ravir\AppData\Local\Temp\ipykernel_6424\2388665489.py:7: UserWarning: pan
das only supports SQLAlchemy connectable (engine/connection) or database string UR
I or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consid
er using SQLAlchemy.
  sql_df = pd.read_sql(query, connection)
```

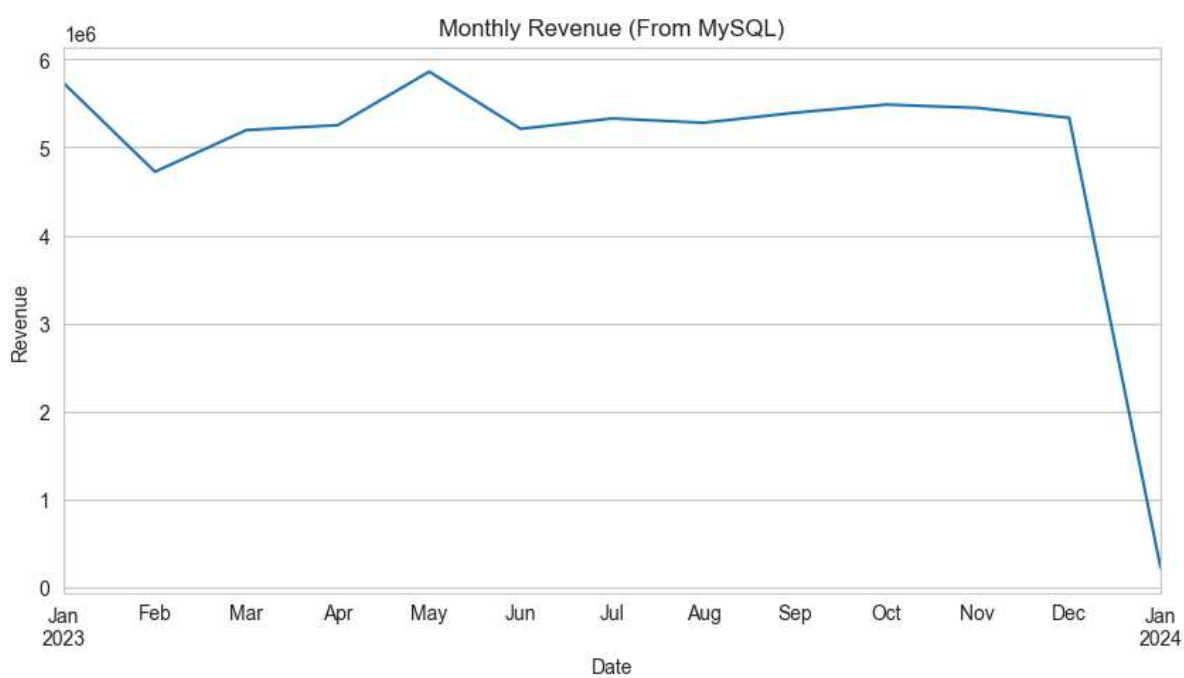
Out[30]:

	order_date	revenue
0	2023-06-05	9574.820312
1	2023-07-17	3809.439941
2	2023-07-09	12677.370117
3	2023-12-19	206.929993
4	2023-04-12	5730.200195

Monthly Revenue from SQL Data

In [31]:

```
sql_df["order_date"] = pd.to_datetime(sql_df["order_date"])
monthly_sql = sql_df.groupby(pd.Grouper(key="order_date", freq="M"))["revenue"].sum

plt.figure(figsize=(10,5))
monthly_sql.plot()
plt.title("Monthly Revenue (From MySQL)")
plt.xlabel("Date")
plt.ylabel("Revenue")
plt.show()
```



Close Connection

In [32]:
```
cursor.close()
connection.close()
```

Project Summary

Key Findings

- Total Revenue and Order Volume indicate strong business performance.
- Electronics category generates the highest revenue contribution.
- Major cities drive the majority of overall sales.
- Card payments show higher transaction value and strong success rate.
- Monthly revenue trend reflects consistent growth pattern.
- A small percentage of customers contribute to a large portion of total revenue (Pareto Principle observed).

Statistical Insights

- Strong positive correlation between price, quantity, and revenue.
- Hypothesis testing performed to compare payment methods.
- Revenue distribution analysis highlights customer spending behavior.

Database Integration

- Designed and implemented a relational database schema in MySQL.
- Created structured tables: Customers, Products, Orders, and Payments.
- Inserted cleaned dataset into MySQL using Python.
- Performed SQL-based revenue, city, and payment analysis.
- Connected Python with MySQL for hybrid analytics workflow.
- Extracted SQL query results into Pandas for visualization and reporting.

Business Recommendations

- Focus marketing efforts on high-performing categories and cities.
- Implement loyalty programs for high-value customers.
- Optimize payment systems to improve transaction success rate.
- Use monthly revenue trends for forecasting and strategic planning.

This project demonstrates end-to-end Data Analytics capability including data cleaning, EDA, statistical testing, time series analysis, SQL database integration, and business insight generation using Python and MySQL.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: