Notebook pricing_model_dataset.csv

```
from faker import Faker
import pandas as pd
import random

fake = Faker()

# Function to generate product data
def generate_product_data(num_products):
    products = []
    for _ in range(num_products):
        product_id = fake.uuid4()
        product_name = fake.catch_phrase()
        category = fake.random_element(elements=('Electronics', 'Clothing', 'Home & Kitchen', 'Books', 'Toys')
        price = round(random.uniform(10, 1000), 2)
        competitor_price = round(price * random.uniform(0.8, 1.2), 2)  # Generate competitor price
        products.append([product_id, product_name, category, price, competitor_price])
    return products

# Function to generate competitor data
def generate_competitor_data(num_competitors, num_products):
    competitors = []
    for _ in range(num_competitors):
        competitor_id = fake.uuid4()
        competitor_name = fake.company()
        products = random.sample(range(num_products), random.randint(5, num_products))  # Randomly select prod
        competitors.append([competitor_id, competitor_name, products])
    return competitors

# Generate product data
```

```
products = generate_product_data(1000)

# Generate competitor data
competitors = generate_competitor_data(10, len(products))

# Function to assign competitor prices to products
def assign_competitor_prices(products, competitors):
    for product in products:
        competitor_prices = []
        for competitor in competitors:
            if products.index(product) in competitor[2]:  # Check if product is in competitor's list
                competitor_prices.append(product[3] * random.uniform(0.8, 1.2))  # Generate competitor price
            else:
                competitor_prices.append(None)
        product.extend(competitor_prices)

# Assign competitor prices to products
assign_competitor_prices(products, competitors)

# Create DataFrame
columns = ['Product_ID', 'Product_Name', 'Category', 'Price'] + [f'Competitor_{i}_Price' for i in range(len(co
df = pd.DataFrame(products, columns=columns)

# Save DataFrame to CSV file
df.to_csv('ecommerce_data_with_competitors.csv', index=False)

print("E-commerce dataset with competitor data generated successfully!")
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/internals/construction.py in
_finalize_columns_and_data(content, columns, dtype)
    968     try:
--> 969         columns = _validate_or_indexify_columns(contents, columns)
    970     except AssertionError as err:

 5 frames
AssertionError: 14 columns passed, passed data had 15 columns

The above exception was the direct cause of the following exception:

ValueError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/internals/construction.py in
_finalize_columns_and_data(content, columns, dtype)
    970     except AssertionError as err:
    971         # GH#26429 do not raise user-facing AssertionError
--> 972         raise ValueError(err) from err
    973
    974     if len(contents) and contents[0].dtype == np.object_:

ValueError: 14 columns passed, passed data had 15 columns
```

Next steps: Explain error

```
pip install Faker
```

```
Collecting Faker
  Downloading Faker-24.3.0-py3-none-any.whl (1.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 9.0 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.4 in /usr/local/lib/python3.10/dist-packages (from Faker) (2.8
```

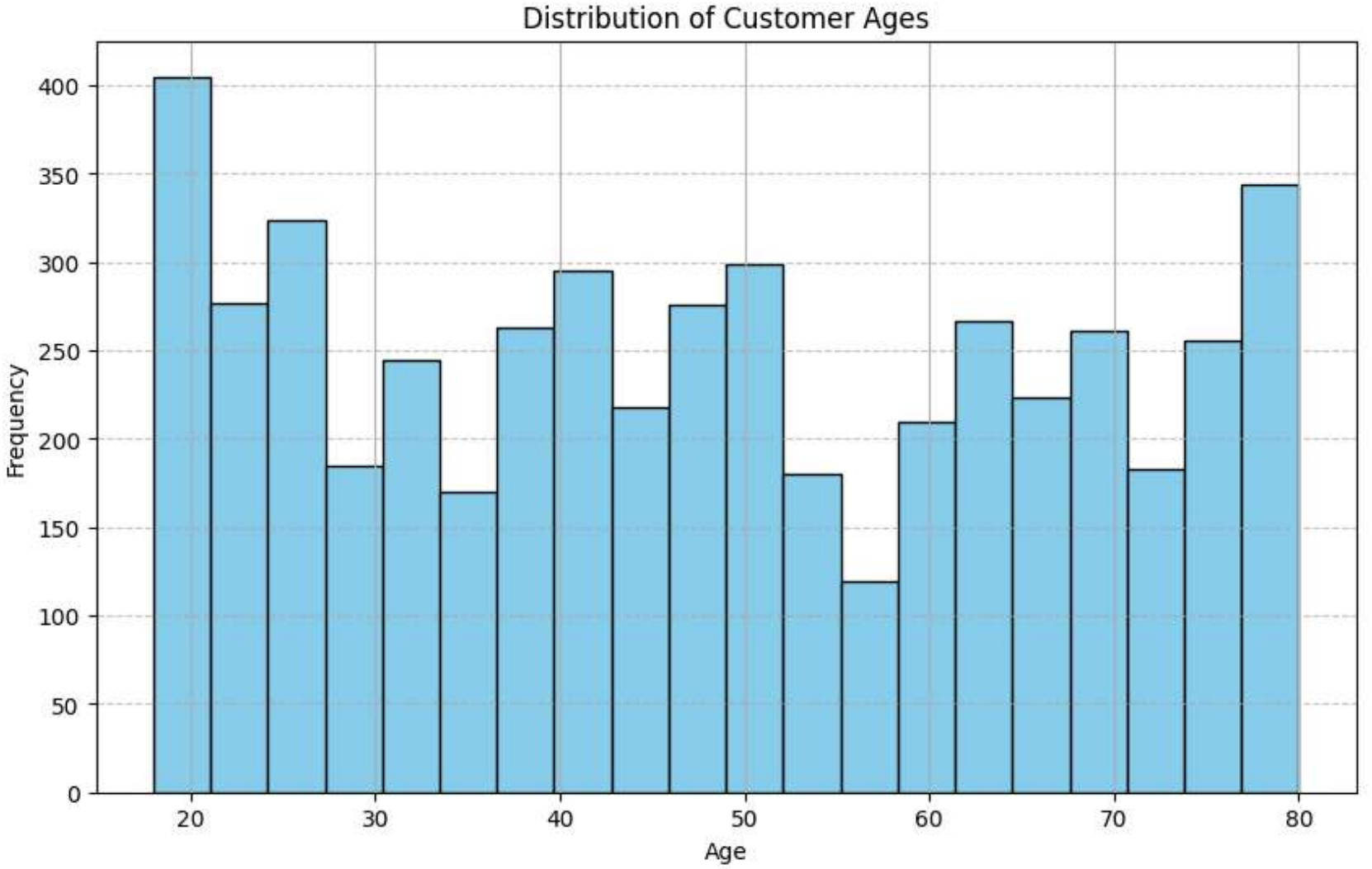
```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.4->F
Installing collected packages: Faker
Successfully installed Faker-24.3.0
```

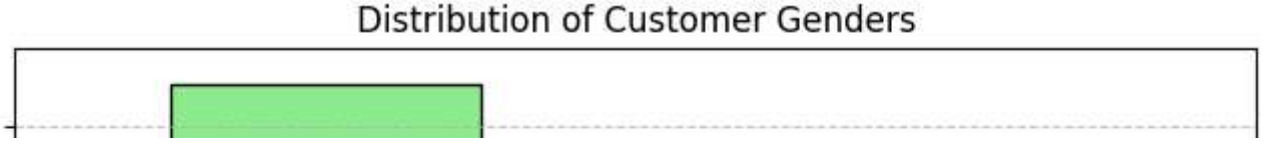
```
import pandas as pd
import numpy as np
import random
from faker import Faker
import datetime

fake = Faker()

# Function to generate product data
def generate_product_data(num_products):
    products = []
    for _ in range(num_products):
        product_id = fake.uuid4()
        product_name = fake.catch_phrase()
        category = fake.random_element(elements=('Electronics', 'Clothing', 'Home & Kitchen', 'Books', 'Toys'))
        price = round(random.uniform(10, 1000), 2)
        products.append([product_id, product_name, category, price])
    return products

# Function to generate customer data
def generate_customer_data(num_customers):
    customers = []
    for _ in range(num_customers):
        customer_id = fake.uuid4()
        name = fake.name()
        email = fake.email()
        age = random.randint(18, 80)
        gender = fake.random_element(elements=('Male', 'Female'))
        address = fake.address()
        customers.append([customer_id, name, email, age, gender, address])
    return customers
```

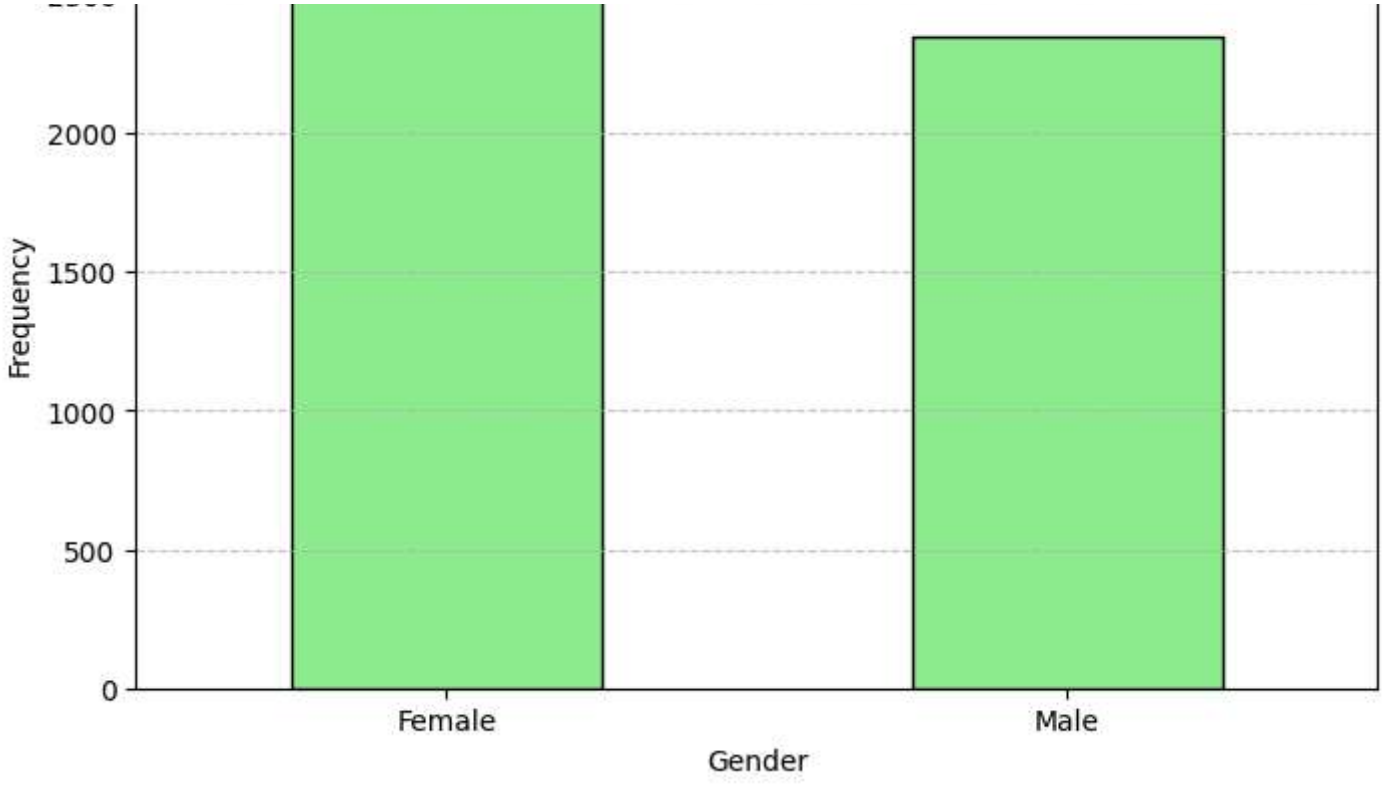
```
# Function to generate transaction data
def generate_transaction_data(num_transactions, products, customers):
    transactions = []
    for _ in range(num_transactions):
        transaction_id = fake.uuid4()
        product_id, product_name, category, price = random.choice(products)
        customer_id, name, email, age, gender, address = random.choice(customers)
        quantity = random.randint(1, 5)
        transaction_date = fake.date_time_this_year()
        transactions.append([transaction_id, product_id, product_name, category, customer_id, name, email, age, gend
    return transactions

# Generate product data
products = generate_product_data(1000)

# Generate customer data
customers = generate_customer_data(500)

# Generate transaction data
transactions = generate_transaction_data(5000, products, customers)

# Create DataFrame
columns = ['Transaction_ID', 'Product_ID', 'Product_Name', 'Category', 'Customer_ID', 'Customer_Name', 'Customer_Ema
df = pd.DataFrame(transactions, columns=columns)

# Save DataFrame to CSV file
df.to_csv('pricing_model_dataset.csv', index=False)

print("Pricing model dataset generated successfully!")
```

```
Pricing model dataset generated successfully!
```

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

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Explore the distribution of customer ages
plt.figure(figsize=(10, 6))
df['Customer_Age'].hist(bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Customer Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Explore the distribution of customer genders
plt.figure(figsize=(8, 5))
df['Customer_Gender'].value_counts().plot(kind='bar', color='lightgreen', edgecolor='black')
plt.title('Distribution of Customer Genders')
plt.xlabel('Gender')
plt.ylabel('Frequency')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```


Distribution of Customer Ages
Frequency
Age
0
50
100
150
200
250
300
350
400
20
30
40
50
60
70
80


Distribution of Customer Genders
2500


2000
1500
1000
500
0
Frequency
Female
Male
Gender

```
import pandas as pd

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Extract location information from addresses
df['Location'] = df['Customer_Address'].apply(lambda address: address.split()[-1])

# Group customers by location and calculate the average age
location_age_distribution = df.groupby('Location')['Customer_Age'].mean().reset_index()

# Display the location-wise average age distribution
print(location_age_distribution)
```

```
     Location  Customer_Age
0       00724          73.0
1       00848          76.0
2       00923          66.0
3       01161          66.0
4       01414          44.0
..        ...           ...
495     97985          19.0
496     98653          31.0
497     98682          46.0
498     98894          33.0
499     99240          55.0

[500 rows x 2 columns]
```

```
import pandas as pd

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Extract location information from addresses
df['Location'] = df['Customer_Address'].apply(lambda address: address.split()[-1])

# Group customers by location and calculate the average age
location_avg_age = df.groupby('Location')['Customer_Age'].mean().reset_index()

# Display the location-wise average age distribution
print(location_avg_age)
```

```
    Location  Customer_Age
0      00724          73.0
1      00848          76.0
2      00923          66.0
3      01161          66.0
4      01414          44.0
..       ...           ...
495    97985          19.0
496    98653          31.0
497    98682          46.0
498    98894          33.0
499    99240          55.0

[500 rows x 2 columns]
```

```
import pandas as pd

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Extract location information from addresses
df['Location'] = df['Customer_Address'].apply(lambda address: address.split()[-1])

# Group customers by location and calculate the average age
location_avg_age = df.groupby('Location')['Customer_Age'].mean().reset_index()

# Display the location-wise average age distribution
print(location_avg_age)
```

```
     Location  Customer_Age
0       00724          73.0
1       00848          76.0
2       00923          66.0
3       01161          66.0
4       01414          44.0
..        ...           ...
495     97985          19.0
496     98653          31.0
497     98682          46.0
498     98894          33.0
499     99240          55.0

[500 rows x 2 columns]
```

```
import pandas as pd

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Define a function to extract city names from addresses
def extract_city(address):
    # Split the address by commas and extract the first element
    parts = address.split(',')
    if len(parts) >= 2:
        return parts[-2].strip()  # Assuming the city name is the second-to-last element after splitting
    else:
        return None

# Apply the function to extract city names
df['City'] = df['Customer_Address'].apply(extract_city)

# Group customers by city and calculate the average age
city_avg_age = df.groupby('City')['Customer_Age'].mean().reset_index()

# Display the city-wise average age distribution
print(city_avg_age)
```

```
                                                City  Customer_Age
0              000 Catherine Dam Suite 071\nJustinberg          40.0
1    0034 Steven Fields Apt. 050\nPort Jenniferborough          34.0
2                       007 Donna Falls\nNorth Kristin          49.0
3                    008 Miles Plains\nLake Jessicaton          36.0
4            0132 Evans Island Suite 991\nEnglishburgh          72.0
..                                                 ...           ...
456                                          PSC 8098          20.0
457                                          PSC 8746          66.0
```

```
458                                    PSC 8844          18.0
459                                    PSC 9485          40.0
460                                    PSC 9855          33.0

[461 rows x 2 columns]
```

```
import pandas as pd

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Define a function to extract city names from addresses
def extract_city(address):
    # Split the address by commas and extract the first element
    parts = address.split(',')
    if len(parts) >= 2:
        city = parts[-2].strip()  # Assuming the city name is the second-to-last element after splitting
        return city
    else:
        return None

# Apply the function to extract city names
df['City'] = df['Customer_Address'].apply(extract_city)

# Group customers by city and calculate the average age
city_avg_age = df.groupby('City')['Customer_Age'].mean().reset_index()

# Display the city-wise average age distribution
print(city_avg_age)
```

```
                                                  City  Customer_Age
0              000 Catherine Dam Suite 071\nJustinberg          40.0
1    0034 Steven Fields Apt. 050\nPort Jenniferborough          34.0
2                        007 Donna Falls\nNorth Kristin          49.0
3                    008 Miles Plains\nLake Jessicaton          36.0
4            0132 Evans Island Suite 991\nEnglishburgh          72.0
..                                                 ...           ...
456                                          PSC 8098          20.0
457                                          PSC 8746          66.0
458                                          PSC 8844          18.0
459                                          PSC 9485          40.0
460                                          PSC 9855          33.0

[461 rows x 2 columns]
```

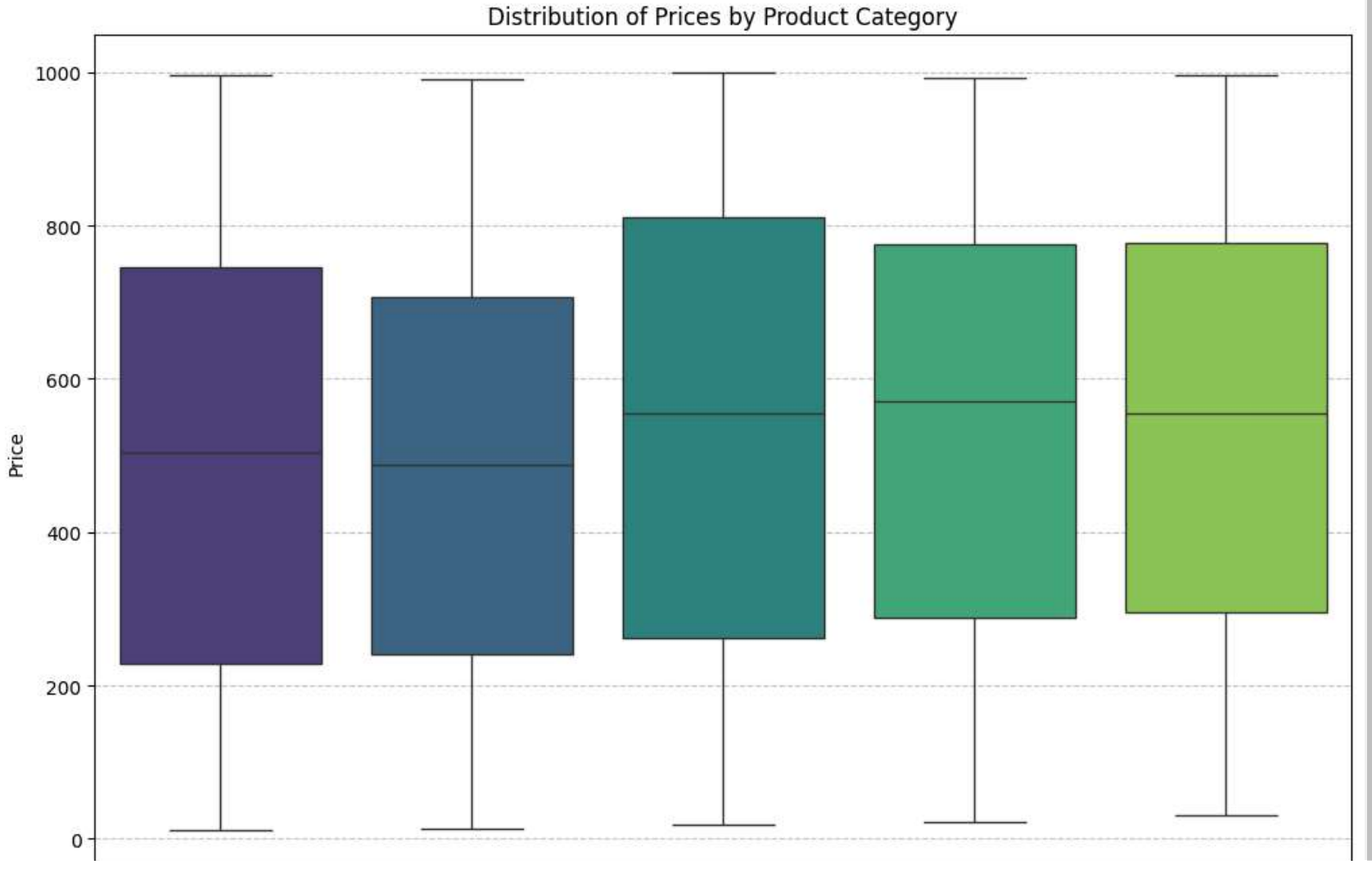
```
import pandas as pd
import re

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Define a function to extract city names from addresses using regex
def extract_city(address):
    # Use regex to find the city name
    match = re.search(r'([A-Za-z\s]+), [A-Z]{2} \d{5}', address)
    if match:
        return match.group(1).strip()
    else:
        return None

# Apply the function to extract city names
df['City'] = df['Customer_Address'].apply(extract_city)

# Group customers by city and calculate the average age
city_avg_age = df.groupby('City')['Customer_Age'].mean().reset_index()

# Display the city-wise average age distribution
print(city_avg_age)
```

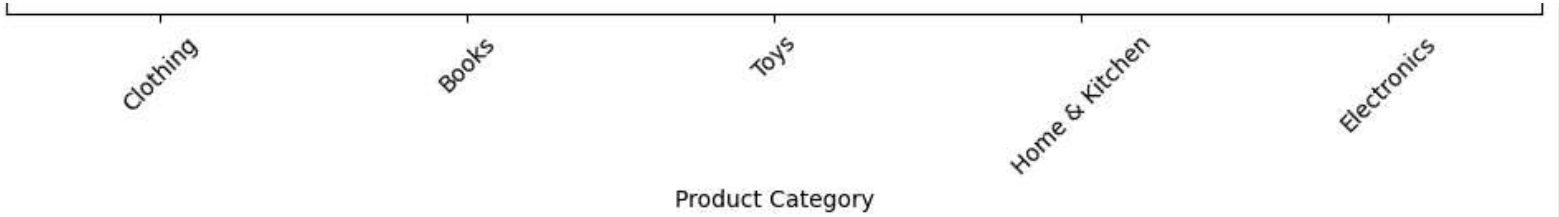
```
                            City  Customer_Age
0                     Aaronmouth          22.0
1                      Aaronview          45.0
2        Adams Valleys\nLisastad          77.0
3      Adrian Cliff\nNew Deborah          50.0
4                     Adrianland          26.0
..                           ...           ...
435                Williamstown          18.0
```

```
436   Wilson Shore\nGabrielaport          42.0
437   Wilson Viaduct\nMorenofurt          31.0
438                   Wilsonfurt          35.0
439  Wood Falls\nEast Thomasberg          75.0

[440 rows x 2 columns]
```

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

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Box plot of prices by product category
plt.figure(figsize=(12, 8))
sns.boxplot(data=df, x='Category', y='Price', palette='viridis')
plt.title('Distribution of Prices by Product Category')
plt.xlabel('Product Category')
plt.ylabel('Price')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Box plot of prices by customer gender
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x='Customer_Gender', y='Price', palette='magma')
plt.title('Distribution of Prices by Customer Gender')
plt.xlabel('Customer Gender')
plt.ylabel('Price')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
<ipython-input-12-1be015194c0b>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variab

  sns.boxplot(data=df, x='Category', y='Price', palette='viridis')
```

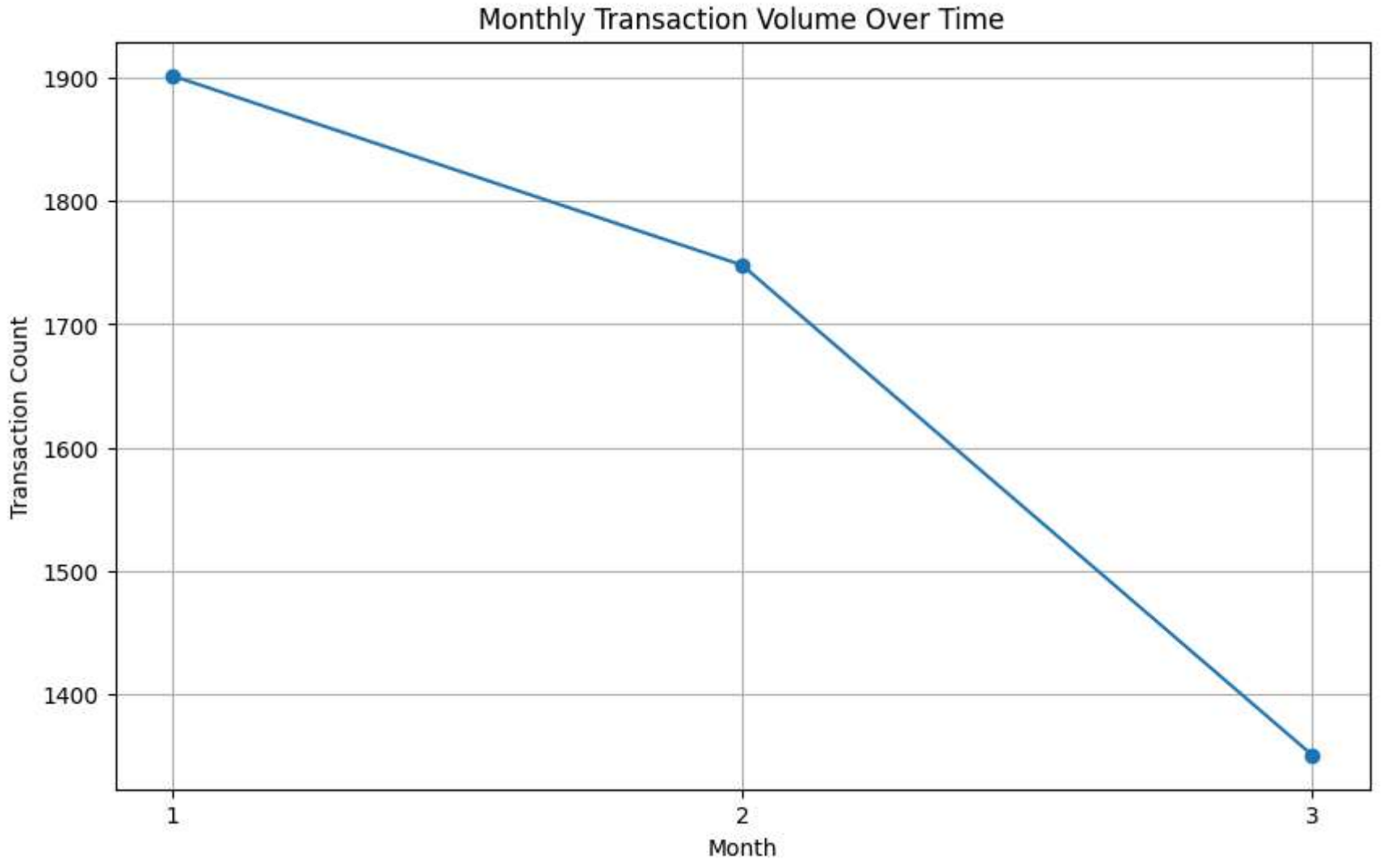





```
<ipython-input-12-1be015194c0b>:20: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variab

  sns.boxplot(data=df, x='Customer_Gender', y='Price', palette='magma')
```


0
Female
Male
Customer Gender

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

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Convert 'Transaction_Date' column to datetime format
df['Transaction_Date'] = pd.to_datetime(df['Transaction_Date'])

# Extract month and year from 'Transaction_Date'
df['Month'] = df['Transaction_Date'].dt.month
df['Year'] = df['Transaction_Date'].dt.year

# Group transactions by month and year
monthly_sales = df.groupby(['Year', 'Month']).size().reset_index(name='Transaction_Count')

# Plot monthly transaction volume
plt.figure(figsize=(10, 6))
plt.plot(monthly_sales['Transaction_Count'], marker='o', linestyle='-')
plt.title('Monthly Transaction Volume Over Time')
plt.xlabel('Month')
plt.ylabel('Transaction Count')
plt.xticks(range(len(monthly_sales)), monthly_sales['Month'])
plt.grid(True)
plt.show()
```


Monthly Transaction Volume Over Time
Transaction Count
1900
1800
1700
1600
1500
1400
1
2
3
Month

```
Start coding or generate with AI.
```

```
import pandas as pd

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Convert 'Transaction_Date' column to datetime format
df['Transaction_Date'] = pd.to_datetime(df['Transaction_Date'])

# Calculate total revenue per customer
total_revenue_per_customer = df.groupby('Customer_ID')['Price'].sum().reset_index()
total_revenue_per_customer.rename(columns={'Price': 'Total_Revenue'}, inplace=True)

# Calculate purchase frequency per customer
purchase_frequency_per_customer = df.groupby('Customer_ID').size().reset_index(name='Purchase_Frequency')

# Calculate average purchase value per customer
average_purchase_value_per_customer = df.groupby('Customer_ID')['Price'].mean().reset_index()
average_purchase_value_per_customer.rename(columns={'Price': 'Average_Purchase_Value'}, inplace=True)

# Merge the calculated metrics into a single DataFrame
customer_metrics = pd.merge(total_revenue_per_customer, purchase_frequency_per_customer, on='Customer_ID')
customer_metrics = pd.merge(customer_metrics, average_purchase_value_per_customer, on='Customer_ID')

# Estimate customer lifespan (assuming it's the difference between the first and last transaction dates)
customer_lifespan = df.groupby('Customer_ID')['Transaction_Date'].apply(lambda x: (x.max() - x.min()).days).reset_in

# Merge customer lifespan into the DataFrame
customer_metrics = pd.merge(customer_metrics, customer_lifespan, on='Customer_ID')

# Calculate CLV using historical CLV method
customer_metrics['CLV'] = customer_metrics['Average_Purchase_Value'] * customer_metrics['Purchase_Frequency'] * cust
```

```
# Segment customers based on CLV scores (e.g., into quartiles)
clv_quartiles = pd.qcut(customer_metrics['CLV'], q=4, labels=['Low', 'Medium', 'High', 'Very High'])
customer_metrics['CLV_Segment'] = clv_quartiles

# Display the calculated metrics and CLV segments
print(customer_metrics)
```

```
                              Customer_ID  Total_Revenue  Purchase_Frequency  \
0    007ffb7e-2060-4255-ab43-0410a0591b7d        7711.86                  11
1    00c767df-a733-4727-a0b2-ef8b3c3fd096        4481.87                  12
2    024a2339-5e18-4b2a-9061-d7003a9fe634        2509.45                   7
3    0299f029-a918-427a-b0d5-e263f972e947        6096.26                  12
4    030fb6c3-3054-4dd5-a857-ebf596b6ee92         385.72                   2
..                                    ...            ...                 ...
495  fc9231f1-9fe2-48d7-8cb0-73013a788f23        7879.25                  15
496  fd305e40-3690-47ba-bf02-93eafcbd8167        3840.52                   9
497  fd738902-0d99-4433-8d77-c6f1c9520a47        3638.64                   8
498  fe08632e-9336-44e9-b8c2-c9ece114c6bd        6936.32                  13
499  fe9af796-0d54-4183-ae14-1444e72b913d        4568.05                  11

     Average_Purchase_Value  Customer_Lifespan        CLV CLV_Segment
0                701.078182                 77  593813.22   Very High
1                373.489167                 72  322694.64      Medium
2                358.492857                 70  175661.50         Low
3                508.021667                 74  451123.24   Very High
4                192.860000                 27   10414.44         Low
..                      ...                ...        ...         ...
495              525.283333                 78  614581.50   Very High
496              426.724444                 68  261155.36      Medium
497              454.830000                 44  160100.16         Low
498              533.563077                 82  568778.24   Very High
499              415.277273                 56  255810.80      Medium
```

```
[500 rows x 7 columns]
```

```
import pandas as pd

# Load the dataset
df = pd.read_csv('pricing_model_dataset.csv')

# Calculate CLV/LTV for each customer (e.g., total revenue)
customer_clv = df.groupby('Customer_ID')['Price'].sum().reset_index()
total_clv = customer_clv['Price'].sum()

# Calculate CLV/LTV as percentage
customer_clv['CLV_Percentage'] = (customer_clv['Price'] / total_clv) * 100

# Display the CLV/LTV percentage for each customer
print(customer_clv)
```

```
                             Customer_ID    Price  CLV_Percentage
0    007ffb7e-2060-4255-ab43-0410a0591b7d  7711.86        0.297418
1    00c767df-a733-4727-a0b2-ef8b3c3fd096  4481.87        0.172849
2    024a2339-5e18-4b2a-9061-d7003a9fe634  2509.45        0.096780
3    0299f029-a918-427a-b0d5-e263f972e947  6096.26        0.235110
4    030fb6c3-3054-4dd5-a857-ebf596b6ee92   385.72        0.014876
..                                    ...      ...             ...
495  fc9231f1-9fe2-48d7-8cb0-73013a788f23  7879.25        0.303873
496  fd305e40-3690-47ba-bf02-93eafcbd8167  3840.52        0.148115
497  fd738902-0d99-4433-8d77-c6f1c9520a47  3638.64        0.140329
498  fe08632e-9336-44e9-b8c2-c9ece114c6bd  6936.32        0.267508
499  fe9af796-0d54-4183-ae14-1444e72b913d  4568.05        0.176173

[500 rows x 3 columns]
```

```
import pandas as pd
from pygam import GAM, s

# Load the dataset
data = pd.read_csv('pricing_model_dataset.csv')

# Filter customers under 35 years old
data_under_35 = data[data['Customer_Age'] < 35]

# Prepare features and target variable
X = data_under_35[['Demand', 'Product_Attribute']]
y = data_under_35['Sales']

# Define and train the GAM model
gam = GAM(s(0) + s(1)).fit(X, y)

# Print model summary
print(gam.summary())

# Make predictions
predictions = gam.predict(X)

# Evaluate model performance (if applicable)
# You can use appropriate metrics such as MSE, R-squared, etc.

# Visualize relationships between input variables and sales (if applicable)
# You can use partial dependence plots or other visualization techniques.

# Use the trained model for price optimization and deployment.
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-18-a74e56de01c7> in <cell line: 11>()
      9
     10 # Prepare features and target variable
---> 11 X = data_under_35[['Demand', 'Product_Attribute']]
     12 y = data_under_35['Sales']
     13

 2 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in _raise_if_missing(self, key, indexer,
axis_name)
   6128                 if use_interval_msg:
   6129                     key = list(key)
-> 6130                 raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   6131
   6132             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())

KeyError: "None of [Index(['Demand', 'Product_Attribute'], dtype='object')] are in the [columns]"
```

Next steps: Explain error

```
pip install pygam
```

```
Collecting pygam
  Downloading pygam-0.9.1-py3-none-any.whl (522 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 522.0/522.0 kB 9.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.25 in /usr/local/lib/python3.10/dist-packages (from pygam) (1.25.2)
Requirement already satisfied: progressbar2<5.0.0,>=4.2.0 in /usr/local/lib/python3.10/dist-packages (from pygam
Requirement already satisfied: scipy<1.12,>=1.11.1 in /usr/local/lib/python3.10/dist-packages (from pygam) (1.11
Requirement already satisfied: python-utils>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from progressbar2
Requirement already satisfied: typing-extensions>3.10.0.2 in /usr/local/lib/python3.10/dist-packages (from pytho
```

```
Installing collected packages: pygam
Successfully installed pygam-0.9.1
```

```
import pandas as pd
from pygam import GAM, s, l

# Load the dataset
data = pd.read_csv('pricing_model_dataset.csv')

# Filter customers below age 35
data_under_35 = data[data['Customer_Age'] < 35]

# Prepare features and target variable
X = data_under_35[['Category', 'Price', 'Quantity', 'Customer_Address']]
y = data_under_35['Sales']

# Define and train the GAM model
gam = GAM(s(0) + l(1) + l(2) + s(3)).fit(X, y)

# Print model summary
print(gam.summary())

# Make predictions
predictions = gam.predict(X)

# Evaluate model performance (if applicable)
# You can use appropriate metrics such as MSE, R-squared, etc.

# Visualize relationships between input variables and sales (if applicable)
# You can use partial dependence plots or other visualization techniques.

# Use the trained model for price optimization and deployment.
```