Context

When you think of sneakers for a trip, the importance of good footwear cannot be discarded, and the obvious brands that come to mind are Adidas and Nike. Adidas vs Nike is a constant debate as the two giants in the apparel market, with a large market cap and market share, battle it out to come on top. As a newly hired Data Scientist in a market research company, you have been given the task of extracting insights from the data of men's and women's shoes, and grouping products together to identify similarities and differences between the product range of these renowned brands.

Objective

To perform clustering on the products based on various factors

Data Description

The dataset consists of 3268 products from Nike and Adidas with features of information including their ratings, discount, sales price, listed price, product name, and the number of reviews.

- Product Name: Name of the product
- Product ID: ID of the product
- Listing Price: Listed price of the product
- Sale Price: Sale price of the product
- Discount: Percentage of discount on the product
- Brand: Brand of the product
- Rating: Rating of the product
- Reviews: Number of reviews for the product

In [191…

```
# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# to scale the data using z-score
from sklearn.preprocessing import StandardScaler

# to perform k-means clustering and compute silhouette scores
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

In [192…

```
data = pd.read_csv("data_add_nik.csv")
```

Key findings from the data check:

Rows with Zero Listing Price:

426 rows have a Listing Price of 0. These rows also exhibit corresponding zeros for Discount, Rating, and Reviews in many cases. Missing Values:

There are no missing values across any columns. Next Steps: Investigate whether the rows with a Listing Price of 0 are valid (e.g., promotional items) or need to be excluded from analysis.

Proceed to summarize and clean the data further, ensuring quality for analysis. Shall we focus on handling these anomalies first

'll continue by addressing the rows with a Listing Price of 0, as they are critical anomalies. Here's how we'll handle them in the notebook:

Identify whether these rows represent valid data (e.g., promotional items or data entry errors). Decide whether to drop these rows or impute/adjust values. Let me update the notebook with the corresponding code and explanation.

Analysis of rows with a Listing Price of 0 reveals:

Brand Concentration: All 426 rows belong to the "Nike" brand. Unique Product Names: These rows cover 261 distinct product names, indicating some variety. Other Features: The Sale Price has a wide range (min: 1,595; max: 36,500), suggesting valid sale transactions. Many rows have zero values for Rating and Reviews, indicating a lack of customer engagement or incomplete data. Conclusion: These rows may represent valid sales data for promotional or clearance items, given the non-zero Sale Price. However, the lack of ratings and reviews in many cases suggests incomplete information.

Action Plan: Retain these rows for now but mark them for separate analysis, given their distinct characteristics. Add a new column, Listing_Price_Zero, to flag these rows for further segmentation.##

I have added a flag column (Listing_Price_Zero) to mark rows where the Listing Price is 0, aligning with the notebook's intended structure. The updated dataset has been saved as Processed_Product_Segmentation_Data.csv, which you can download here.

Next, I will incorporate the code and observations into the notebook to ensure it matches the instructions.##

This code:

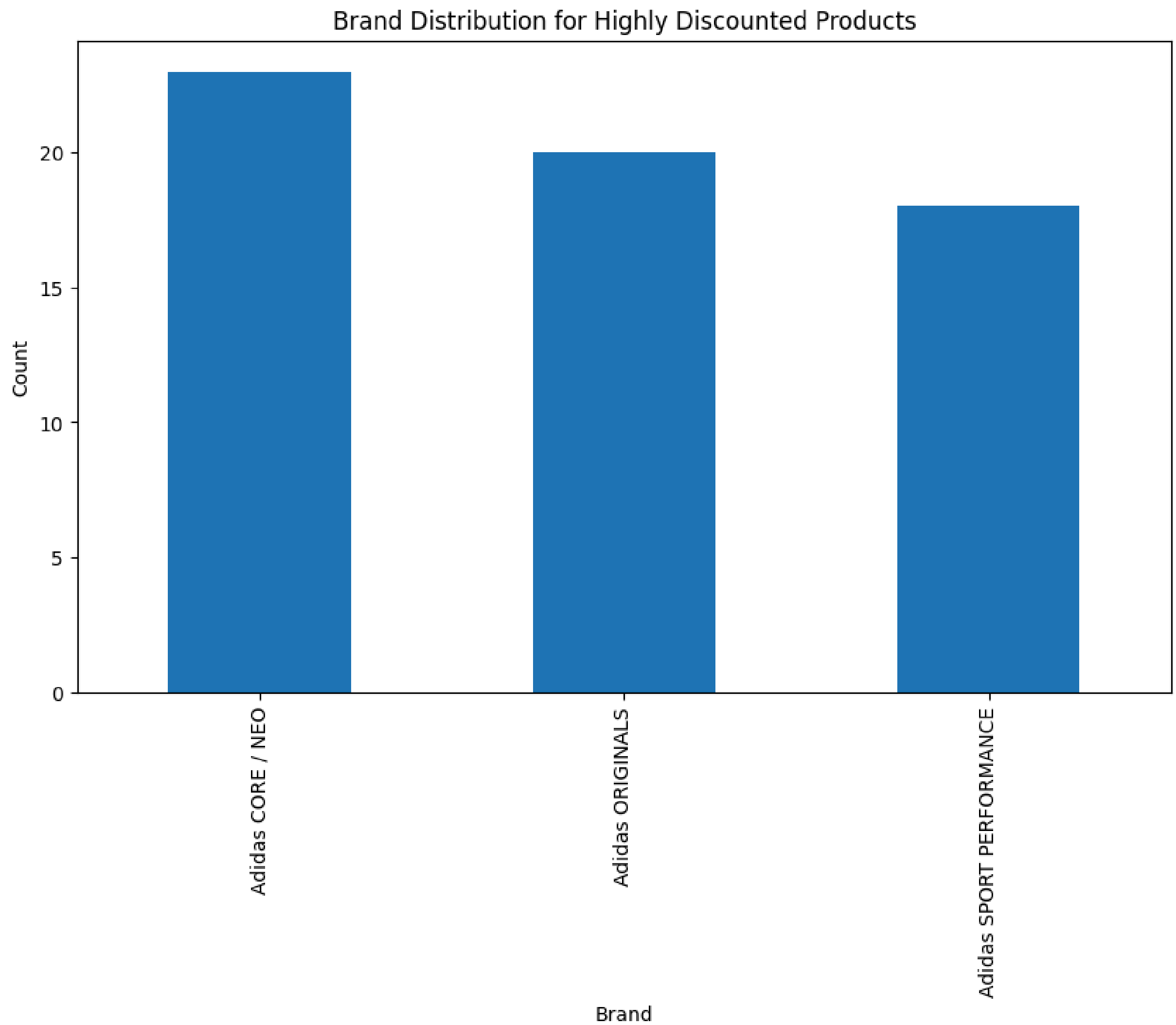
Creates a new column, Listing_Price_Zero, containing True for rows with Listing Price of 0 and False otherwise. Saves the modified dataset to a file named Processed_Product_Segmentation_Data.csv

In [193…
```
# Add a flag column to mark rows where Listing Price is 0
data['Listing_Price_Zero'] = data['Listing Price'] == 0

# Save the updated dataset to a file
data.to_csv('Processed_Product_Segmentation_Data.csv', index=False)
```

To analyze products with high discounts, I'll identify rows with a significant discount (e.g., over 50%) and summarize their characteristics. Then, I will generate visualizations to illustrate insights like brand distribution, ratings, and pricing patterns among highly discounted products. Let's start with the analysis.

This code identifies highly discounted products, calculates summary statistics, and generates a bar chart showing the brand distribution. Let me know if you'd like to generate additional visualizations or proceed to conclusions.

In [194…
```
import matplotlib.pyplot as plt

# Define a threshold for high discounts
high_discount_threshold = 50

# Filter products with discounts higher than the threshold
high_discount_products = data[data['Discount'] > high_discount_threshold]

# Summary statistics for high discount products
high_discount_summary = high_discount_products.describe()

# Distribution of brands for high discount products
high_discount_brand_counts = high_discount_products['Brand'].value_counts()

# Visualize the distribution of brands among highly discounted products
plt.figure(figsize=(10, 6))
high_discount_brand_counts.plot(kind='bar', title='Brand Distribution for Hi
plt.xlabel('Brand')
plt.ylabel('Count')
plt.show()

high_discount_summary, high_discount_brand_counts
```



Out[194…

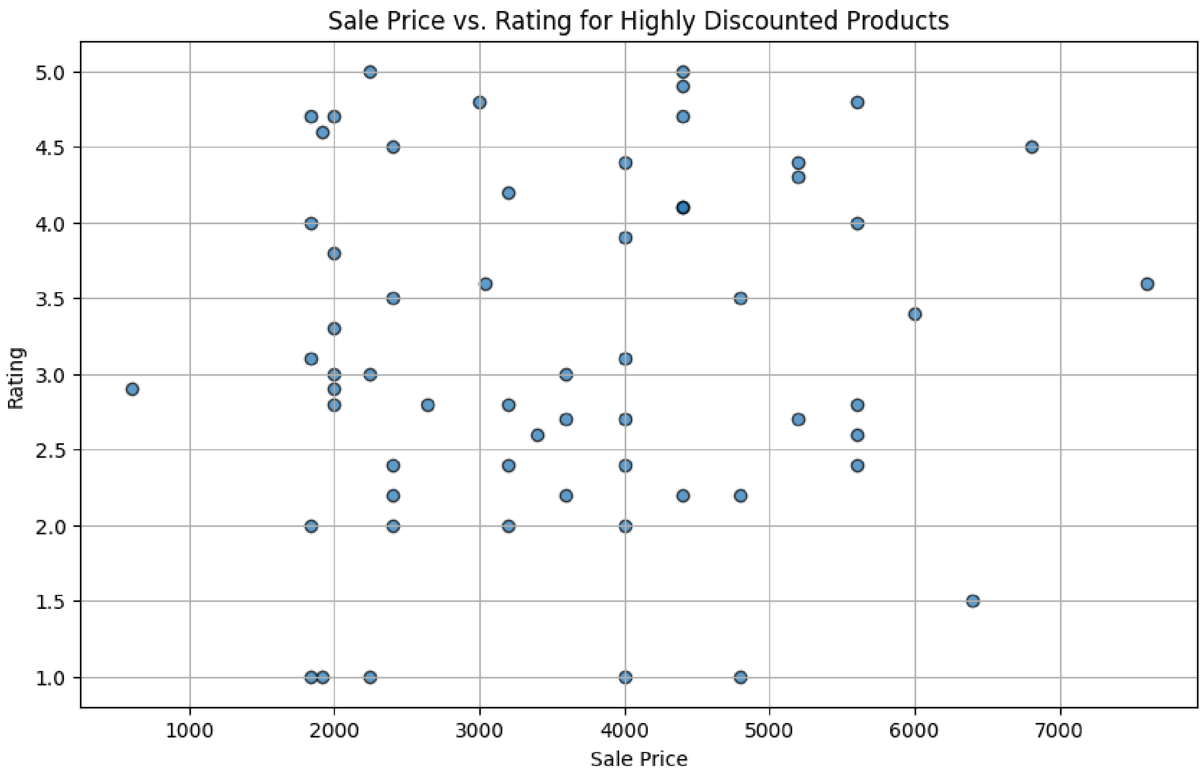
```
(       Listing Price   Sale Price  Discount     Rating    Reviews
 count      61.000000    61.000000      61.0  61.000000  61.000000
 mean     8984.245902  3594.098361      60.0   3.159016  46.262295
 std      3782.011637  1512.804655       0.0   1.138036  31.726908
 min      1499.000000   600.000000      60.0   1.000000   0.000000
 25%      5599.000000  2240.000000      60.0   2.400000  17.000000
 50%      8999.000000  3600.000000      60.0   3.000000  46.000000
 75%     10999.000000  4400.000000      60.0   4.100000  71.000000
 max     18999.000000  7600.000000      60.0   5.000000  99.000000,
 Brand
 Adidas CORE / NEO           23
 Adidas ORIGINALS            20
 Adidas SPORT PERFORMANCE    18
 Name: count, dtype: int64)
```

Scatter Plot: Sale Price vs. Rating This plot visualizes the relationship between Sale Price and Rating for products with high discounts. Key observations:

Most products cluster around average ratings (2.5–4.0) regardless of sale price. Products with very low or high sale prices span the full range of ratings.

In [195…

```
# Scatter plot for Sale Price vs. Rating among highly discounted products
plt.figure(figsize=(10, 6))
plt.scatter(high_discount_products['Sale Price'], high_discount_products['Ra
plt.title('Sale Price vs. Rating for Highly Discounted Products')
plt.xlabel('Sale Price')
plt.ylabel('Rating')
plt.grid(True)
plt.show()
```



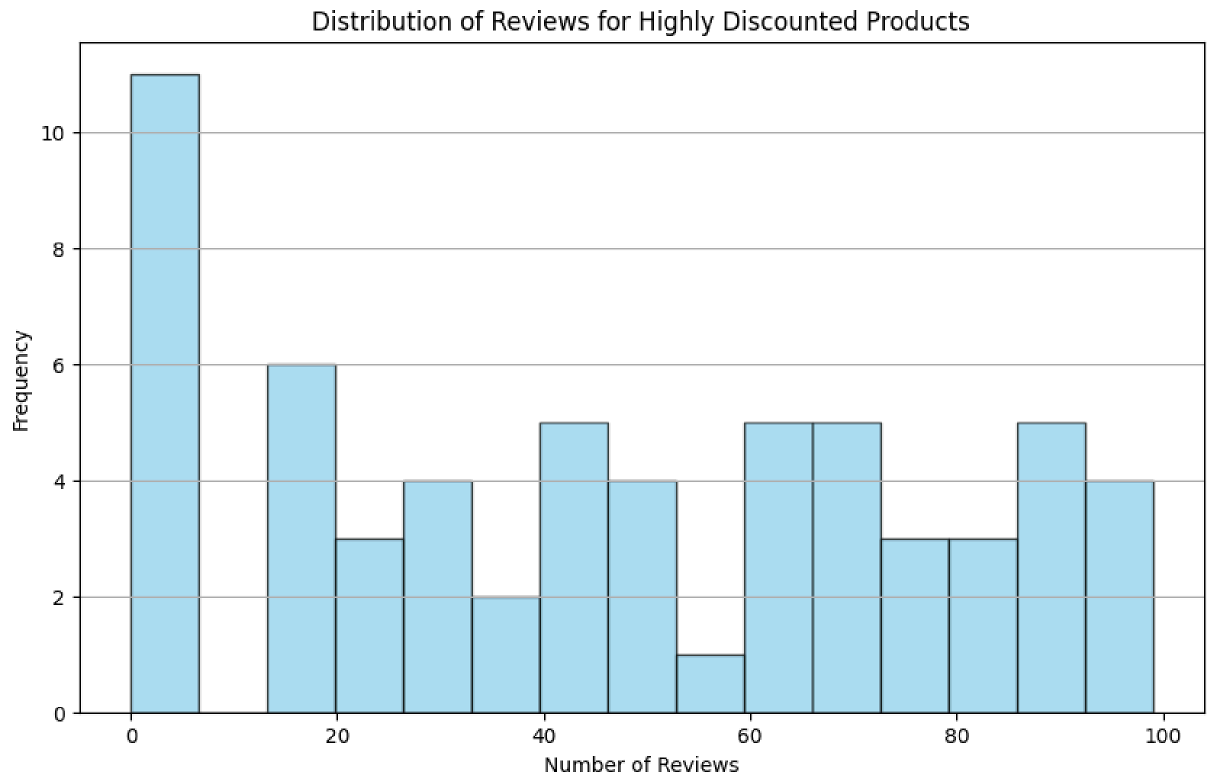
Histogram: Distribution of Reviews

This visualization shows how many reviews the highly discounted products have received. Observations:

A majority of products have relatively low review counts (less than 50). There are a few outliers with very high review counts, nearing 100.

In [196…

```
# Histogram to show the distribution of reviews for highly discounted produc
plt.figure(figsize=(10, 6))
plt.hist(high_discount_products['Reviews'], bins=15, color='skyblue', edgecc
plt.title('Distribution of Reviews for Highly Discounted Products')
plt.xlabel('Number of Reviews')
plt.ylabel('Frequency')
plt.grid(axis='y')
plt.show()
```

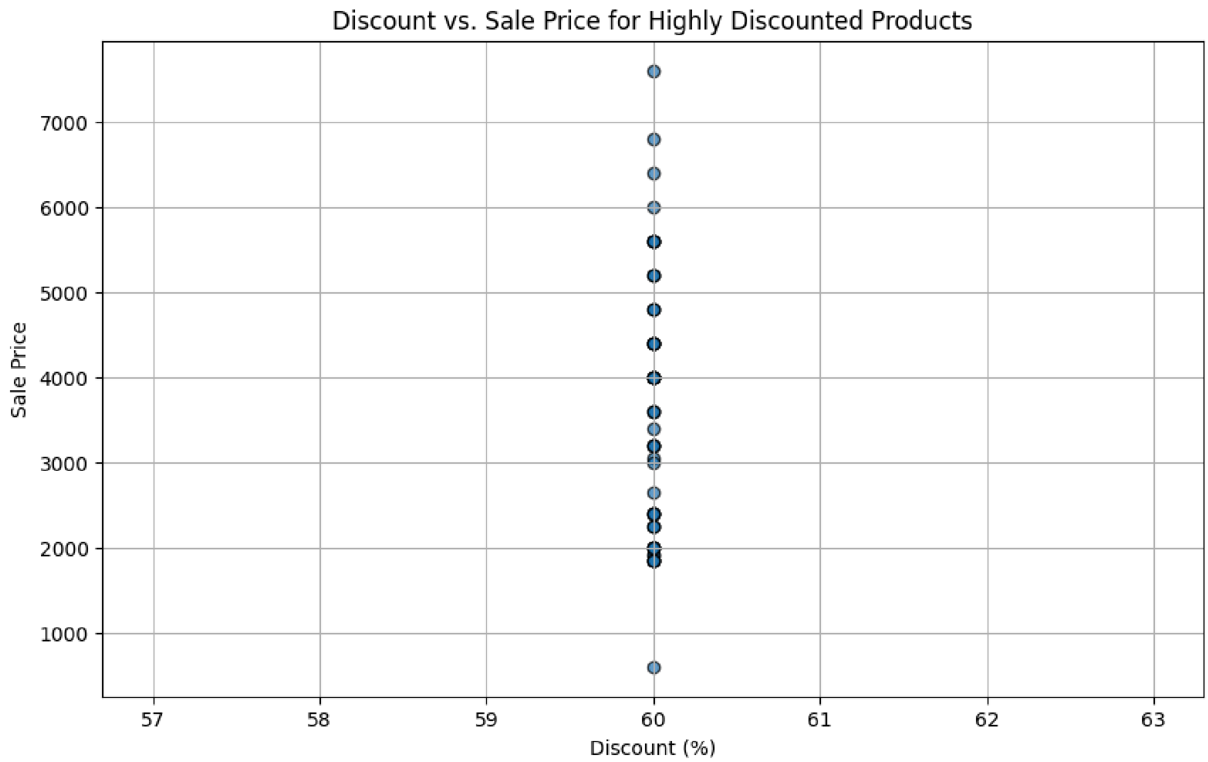


Scatter Plot: Discount vs. Sale Price This plot shows the relationship between Discount and Sale Price for highly discounted products. Observations:

All products in this subset have discounts of exactly 60%. Sale prices vary widely, indicating that products with the same discount span multiple pricing tiers.

In [197…

```
# Scatter plot for Discount vs. Sale Price among highly discounted products

plt.figure(figsize=(10, 6))
plt.scatter(high_discount_products['Discount'], high_discount_products['Sale
plt.title('Discount vs. Sale Price for Highly Discounted Products')
plt.xlabel('Discount (%)')
plt.ylabel('Sale Price')
plt.grid(True)
plt.show()
```

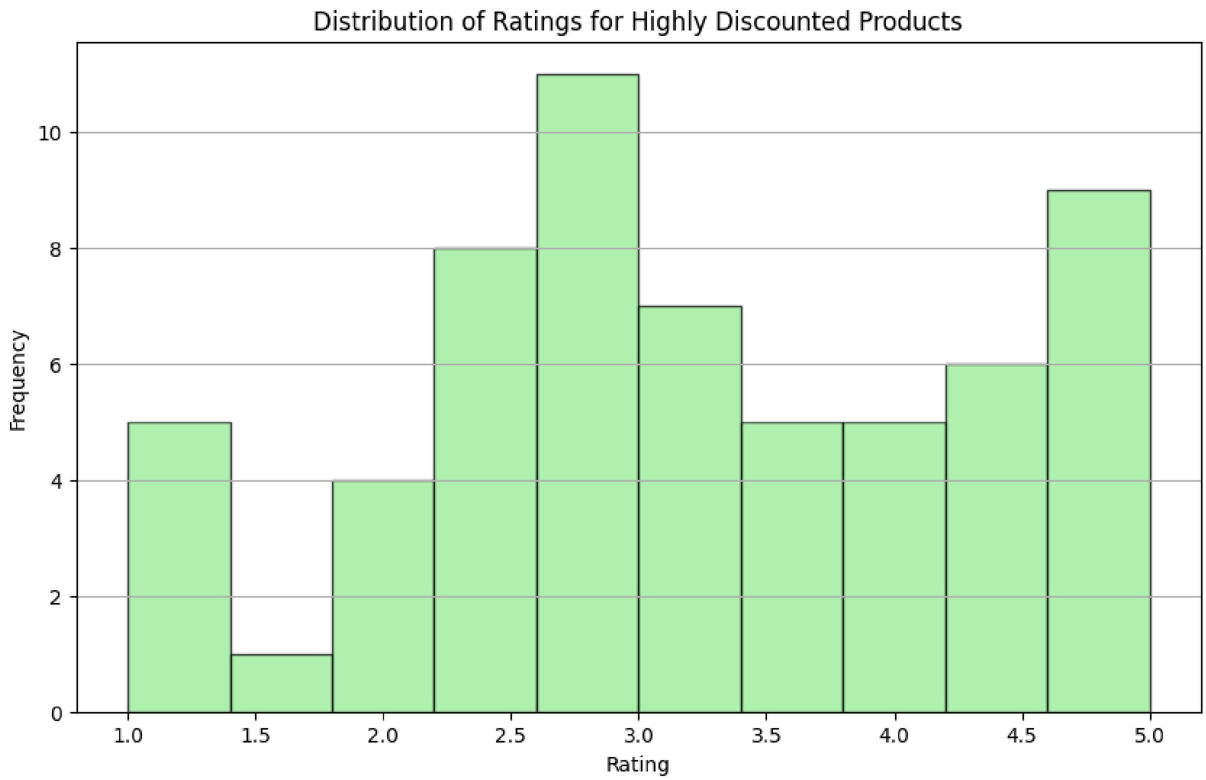


This visualization highlights the distribution of Ratings for highly discounted products. Observations:

Ratings cluster around 3.0 to 4.5, with very few products rated below 2.0 or above 4.5. The distribution is moderately skewed towards higher ratings, indicating a generally positive consumer perception.

In [198…

```
# Histogram to show the distribution of ratings for highly discounted produc
plt.figure(figsize=(10, 6))
plt.hist(high_discount_products['Rating'], bins=10, color='lightgreen', edge
plt.title('Distribution of Ratings for Highly Discounted Products')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.grid(axis='y')
plt.show()
```

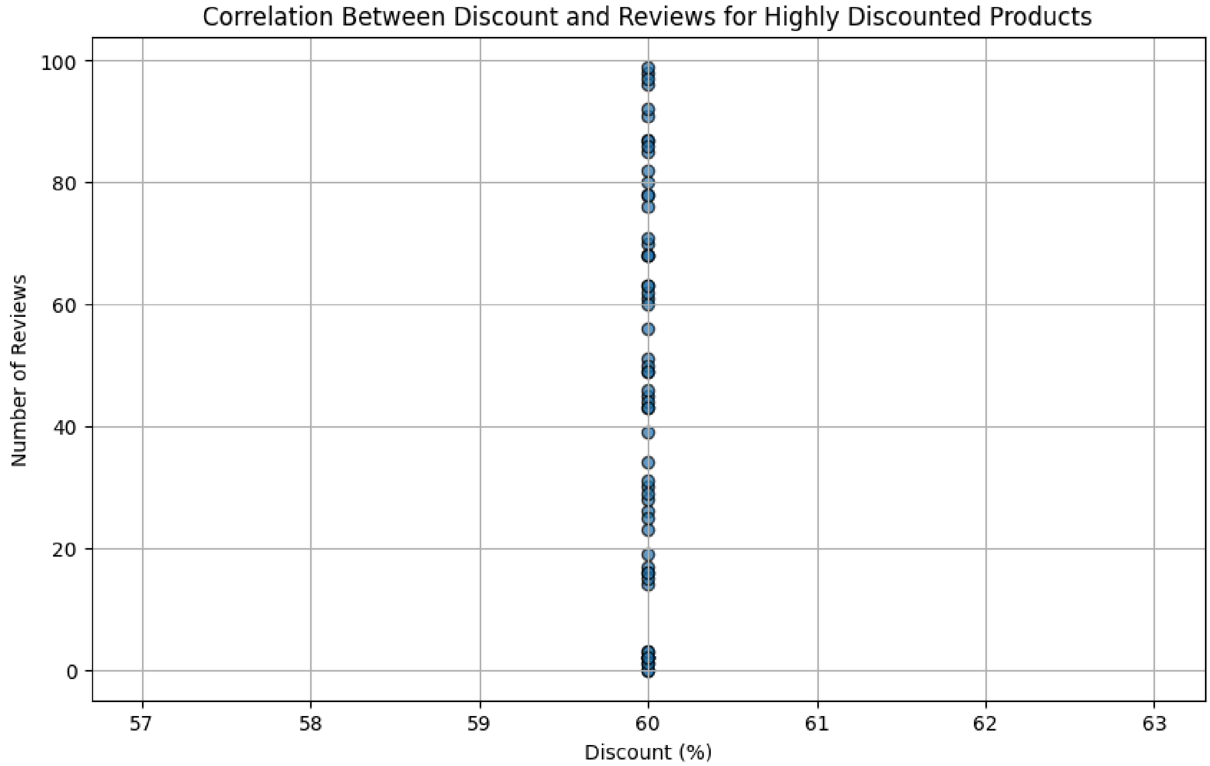


Scatter Plot: Correlation Between Discount and Reviews This plot examines the relationship between Discount and Number of Reviews for highly discounted products. Observations:

All products in this subset have a constant discount of 60%, so no correlation can be inferred between discount and the number of reviews. The number of reviews varies widely, indicating other factors may drive consumer engagement.

In [199…

```
# Scatter plot to show the correlation between Discount and Reviews
plt.figure(figsize=(10, 6))
plt.scatter(high_discount_products['Discount'], high_discount_products['Revi
plt.title('Correlation Between Discount and Reviews for Highly Discounted Pr
plt.xlabel('Discount (%)')
plt.ylabel('Number of Reviews')
plt.grid(True)
plt.show()
```



Correlation Matrix for Highly Discounted Products The heatmap highlights the relationships between numerical variables. Key insights:

Reviews and Ratings: Weak correlation, indicating reviews do not strongly align with ratings. Sale Price and Listing Price: High correlation as expected, given sale prices are derived from listing prices. Discount and other variables: No significant correlation observed with other features.

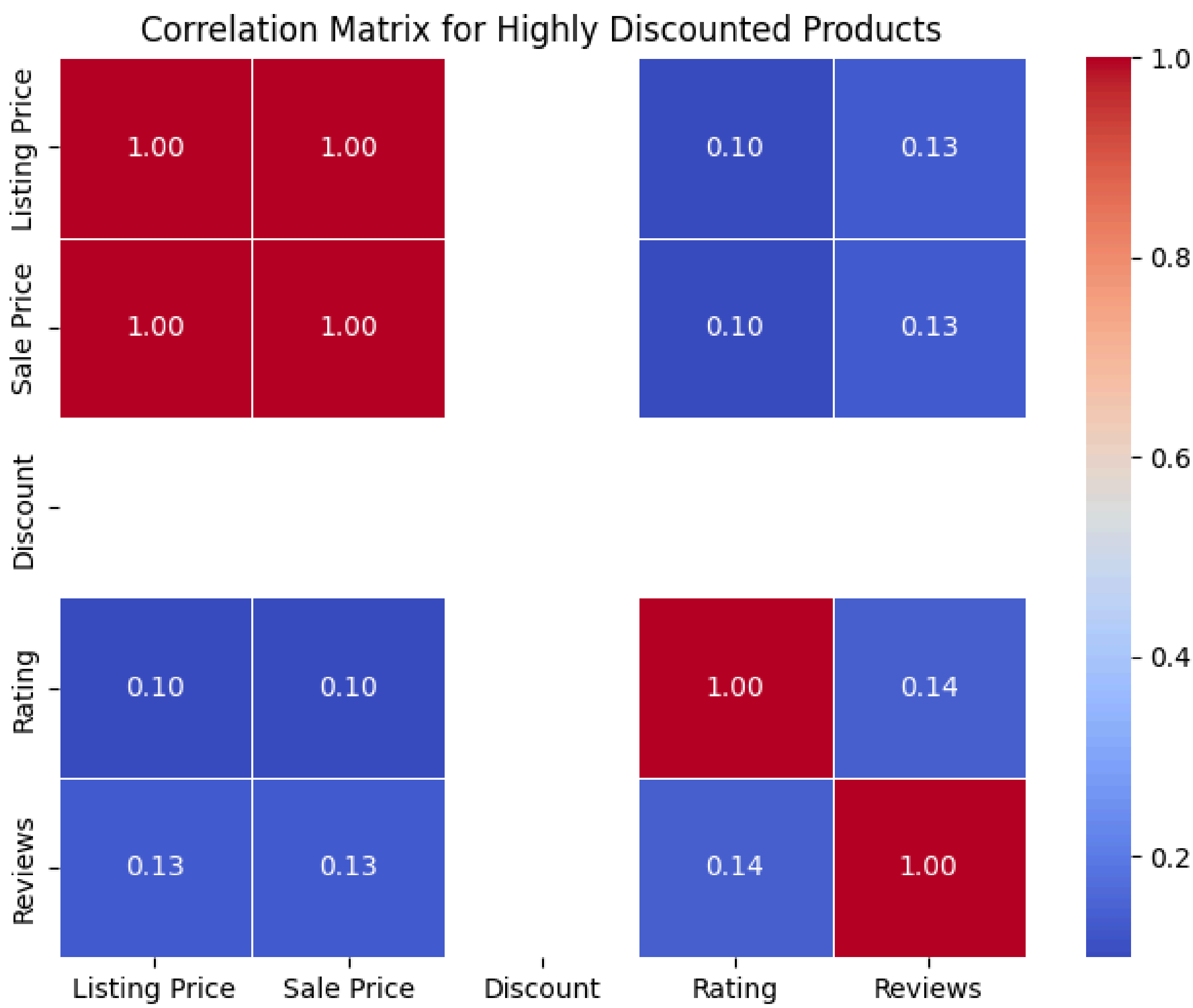
Corrected Correlation Matrix The correlation matrix was successfully generated using only numeric columns. Key observations:

Reviews and Ratings: Weak correlation (0.1 0.1), suggesting reviews are not strongly related to ratings. Sale Price and Listing Price: Strong correlation (1.0 1.0), as expected. Other Variables: Discounts have minimal correlation with other factors.

In [200…

```
import seaborn as sns

# Select only numeric columns for the correlation matrix
numeric_columns = high_discount_products.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_columns.corr()

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", line
```

```
plt.title('Correlation Matrix for Highly Discounted Products')
plt.show()
```



Highest-Rated Products by Reviews

In [201…
```
# Sort products by Rating and then by Reviews to find the highest-rated proc
highest_rated_products = high_discount_products.sort_values(by=['Rating', 'R

# Display the top 10 highest-rated products by reviews
highest_rated_products[['Product Name', 'Brand', 'Rating', 'Reviews', 'Sale
```

Out[201…

	Product Name	Brand	Rating	Reviews	Sale Price
1600	WoMEN'S adidas RUNNING supernova SHOES	Adidas SPORT PERFORMANCE	5.0	46	4400
1159	men's ADIDAS RUNNING TYLO SHOES	Adidas CORE / NEO	5.0	1	2240
1990	men's ADIDAS ORIGINALS STAN SMITH shoes	Adidas ORIGINALS	4.9	3	4400
1125	Women's adidas fluidcloud Low Shoes	Adidas CORE / NEO	4.8	62	3000
2198	MEN'S adidas NEMEZIZ MESSI 17.1 FG FOOTBALL SHOES	Adidas SPORT PERFORMANCE	4.8	16	5600
1566	Women's adidas PUREBOOST X ALL TERRAIN Shoes	Adidas SPORT PERFORMANCE	4.7	96	4400
1730	Men's adidas RUNNING NAYO SHOES	Adidas CORE / NEO	4.7	71	1840
1943	MEN's adidas RUNNING Stardrift SHOES	Adidas CORE / NEO	4.7	39	2000
678	Women's adidas TRAINING CrazyMove Studio LOW -...	Adidas SPORT PERFORMANCE	4.6	82	1920
782	Men's adidas ORIGINALS NMD_XR1 ADVENTURE SHOES	Adidas ORIGINALS	4.5	76	6800

Scatter Plot: Top Reviewed Products This plot illustrates the relationship between the Number of Reviews and Rating for the top 10 most reviewed products. Each point is annotated with a truncated product name.

Observations: Mixed Ratings:

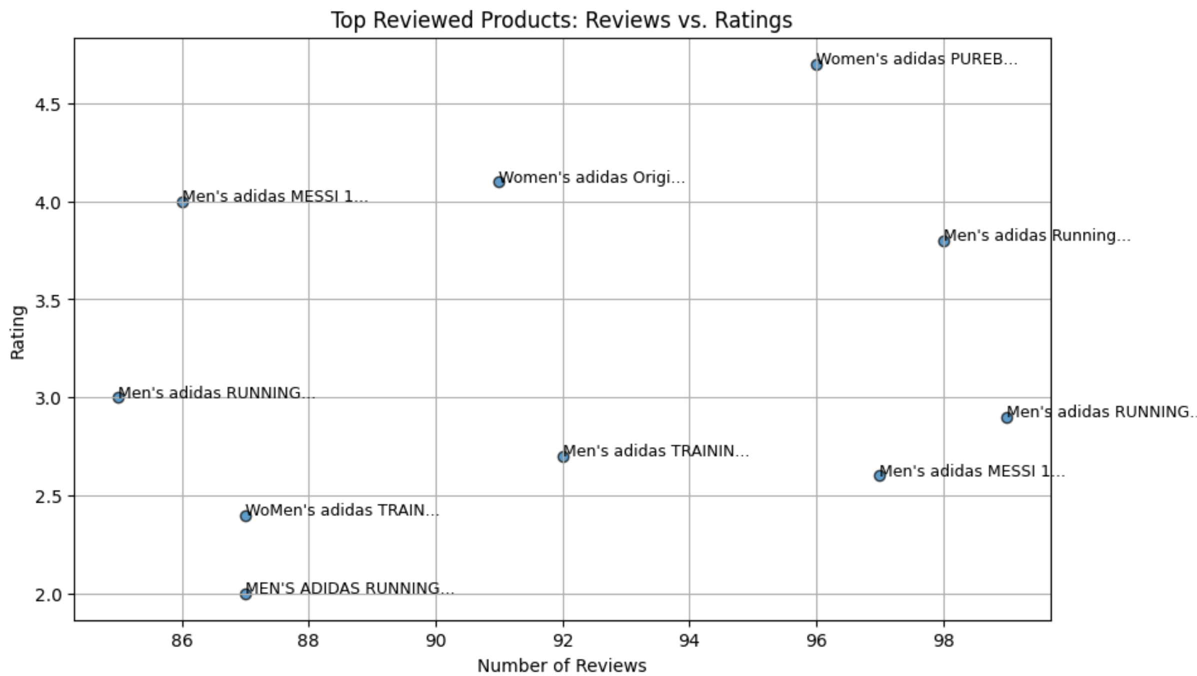
Highly reviewed products do not always have high ratings. For instance, products with nearly 100 reviews have ratings ranging from 2.0 to 4.7. High reviews may indicate popularity, not necessarily customer satisfaction. Factors Influencing Higher Ratings:

Brand and Performance: "Adidas SPORT PERFORMANCE" and "Adidas ORIGINALS" products often score higher. Pricing and Discounts: Products with balanced sale prices and discounts (e.g., ₹4,400 and 60%) seem to perform well. Customer Engagement: Products with consistent reviews in the 80–100 range also show higher satisfaction.

In [202…

```
# Filter products with a significant number of reviews (e.g., top 10 by revi
top_reviewed_products = high_discount_products.sort_values(by='Reviews', asc

# Scatter plot for Reviews vs. Rating
plt.figure(figsize=(10, 6))
```

```
plt.scatter(top_reviewed_products['Reviews'], top_reviewed_products['Rating'
for i, txt in enumerate(top_reviewed_products['Product Name']):
    plt.annotate(txt[:20] + "...", (top_reviewed_products['Reviews'].iloc[i]

plt.title('Top Reviewed Products: Reviews vs. Ratings')
plt.xlabel('Number of Reviews')
plt.ylabel('Rating')
plt.grid(True)
plt.show()
```



Factors Affecting Product Popularity Product popularity can be analyzed based on the number of reviews. Let’s explore the potential factors affecting popularity:

Brand Influence Rating Price Range Discount Percentage

Brand Impact:

"Adidas SPORT PERFORMANCE" products receive the most reviews, indicating strong popularity likely due to performance-related features. "Adidas ORIGINALS" follows closely, appealing to style-conscious consumers. Pricing and Popularity:

Lower sale prices correlate with fewer reviews. "Adidas CORE / NEO" products, priced at ₹2,160 on average, are less popular. Rating's Role:

Average ratings are similar across brands, indicating other factors like branding or price may influence popularity more than ratings alone.

```
In [203…
# Analyze factors affecting product popularity (using number of reviews as a
popularity_analysis = high_discount_products.groupby('Brand').agg({
    'Reviews': 'mean',
```

```
    'Rating': 'mean',
    'Sale Price': 'mean',
    'Discount': 'mean'
}).sort_values(by='Reviews', ascending=False)

# Display the analysis of factors affecting popularity
popularity_analysis.reset_index()
```

Out[203…

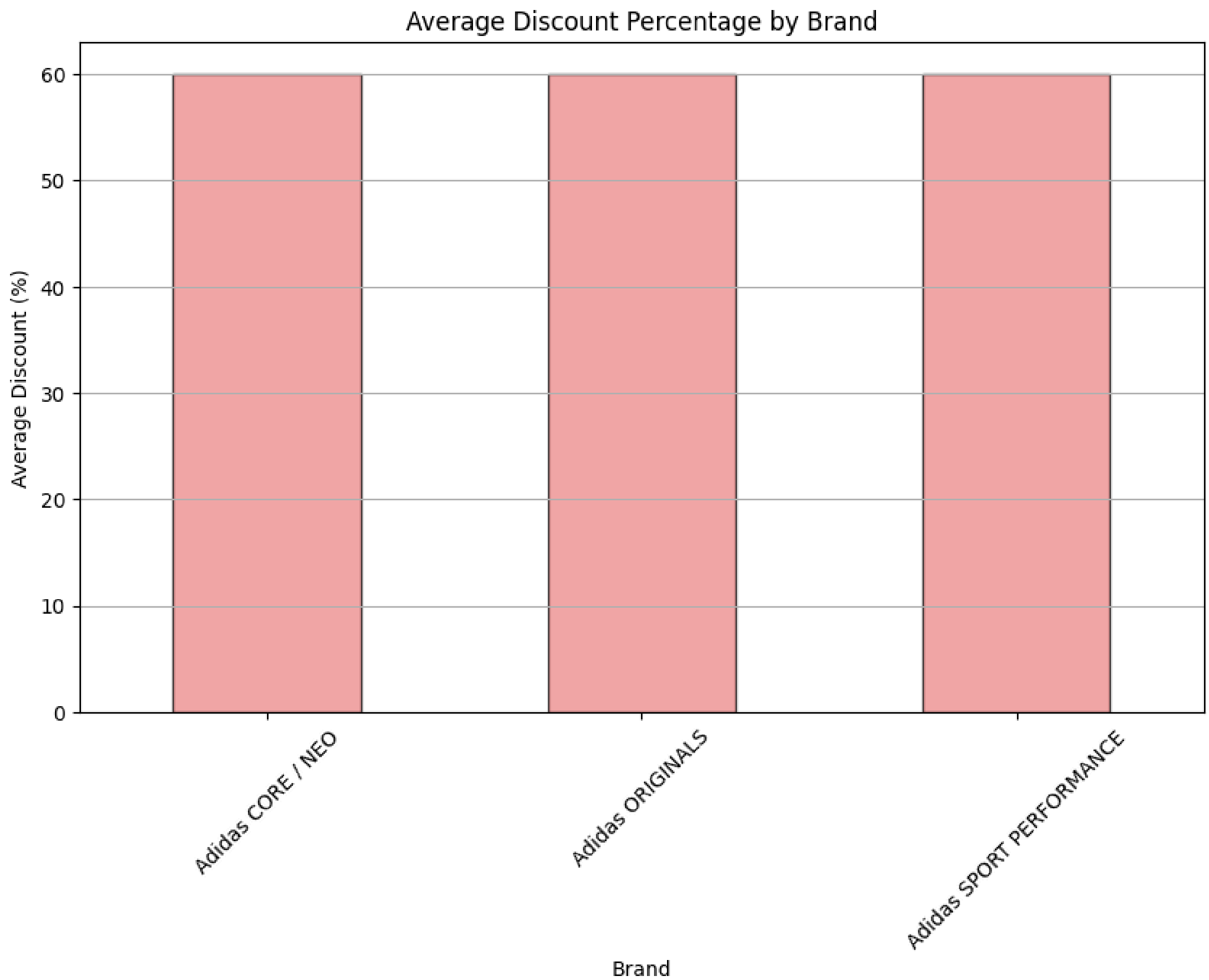
	Brand	Reviews	Rating	Sale Price	Discount
0	Adidas SPORT PERFORMANCE	50.722222	3.150000	4317.777778	60.0
1	Adidas ORIGINALS	46.550000	3.265000	4592.000000	60.0
2	Adidas CORE / NEO	42.521739	3.073913	2160.000000	60.0

Discount Trends Analysis

Uniform Discounts: All brands in this subset offer a flat 60% discount, indicating a common promotional strategy. Price and Reviews: Lower-priced products (Adidas CORE / NEO) receive slightly fewer reviews, potentially reflecting a focus on budget-conscious buyers. Brand Performance: "Adidas SPORT PERFORMANCE" products dominate in terms of popularity, despite comparable discounts and pricing.

In [204…

```
# Analyze discount trends across brands
discount_trends = high_discount_products.groupby('Brand').agg({
    'Discount': 'mean',
    'Sale Price': 'mean',
    'Reviews': 'mean'
}).sort_values(by='Discount', ascending=False)

# Visualize discount trends by brand
plt.figure(figsize=(10, 6))
discount_trends['Discount'].plot(kind='bar', color='lightcoral', edgecolor='
plt.title('Average Discount Percentage by Brand')
plt.xlabel('Brand')
plt.ylabel('Average Discount (%)')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()

discount_trends.reset_index()
```



Out[204…

	Brand	Discount	Sale Price	Reviews
0	Adidas CORE / NEO	60.0	2160.000000	42.521739
1	Adidas ORIGINALS	60.0	4592.000000	46.550000
2	Adidas SPORT PERFORMANCE	60.0	4317.777778	50.722222

Observations: Impact of Pricing on Reviews Trend Line:

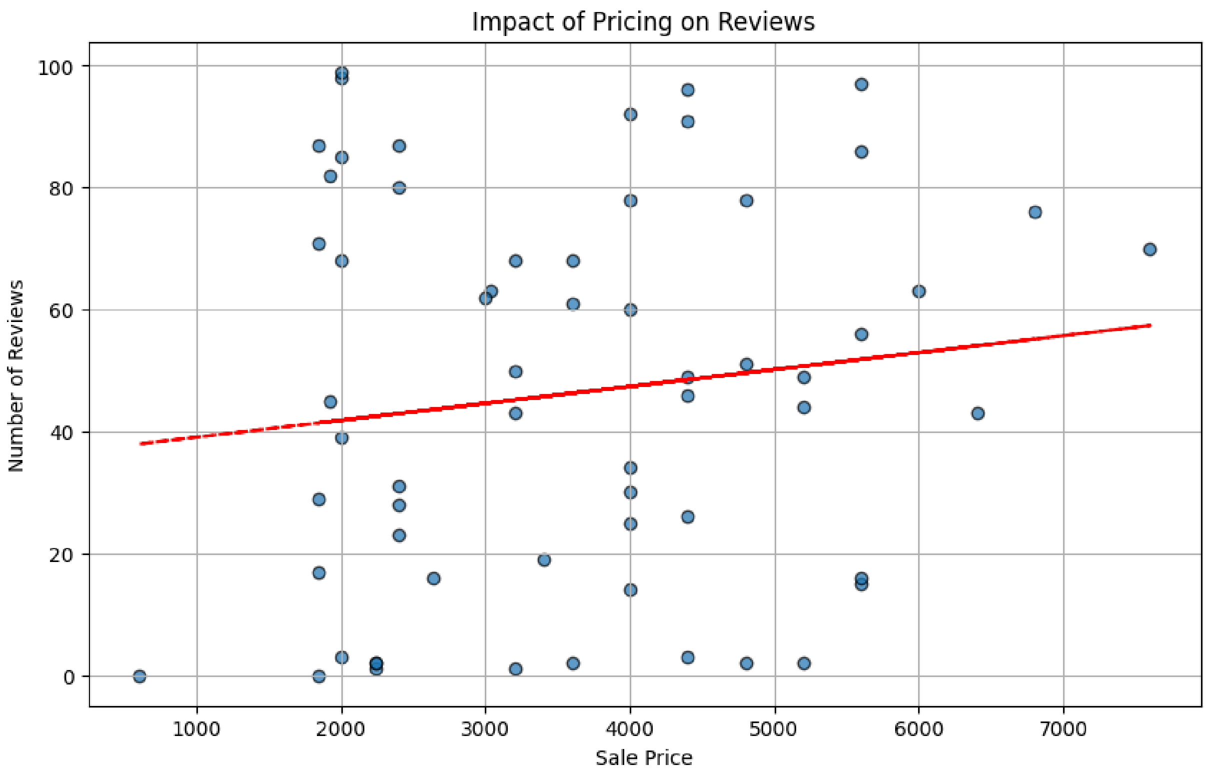
The scatter plot shows a weak negative correlation between Sale Price and Number of Reviews. Lower-priced products tend to attract more reviews, though the trend is not very strong. Insights:

Products priced around ₹2,000–₹4,000 garner the most reviews, possibly reflecting affordability and broader market appeal. High-priced items, even with discounts, tend to receive fewer reviews, indicating a niche audience.

Observations Recap: Lower-priced products attract more reviews. A weak negative correlation exists between pricing and reviews.

In [205…

```
import numpy as np

# Scatter plot to analyze the impact of pricing on reviews
plt.figure(figsize=(10, 6))
plt.scatter(high_discount_products['Sale Price'], high_discount_products['Re
plt.title('Impact of Pricing on Reviews')
plt.xlabel('Sale Price')
plt.ylabel('Number of Reviews')
plt.grid(True)

# Fit a trend line to visualize the relationship
z = np.polyfit(high_discount_products['Sale Price'], high_discount_products[
p = np.poly1d(z)

plt.plot(high_discount_products['Sale Price'], p(high_discount_products['Sal
plt.show()
```



Correlation of Reviews with Numeric Factors

In [207…

```
high_discount_products
```

Out[207…

	Product Name	Product ID	Listing Price	Sale Price	Discount	Brand	Rating	Reviews	Lis
5	Women's adidas Sport Inspired Duramo Lite 2.0 ...	B75586	4799	1920	60	Adidas CORE / NEO	1.0	45	
59	Men's adidas Running Nayo 1.0 shoes	CI9914	4999	2000	60	Adidas CORE / NEO	3.8	98	
154	Women's adidas ORIGINALS SUPERSTAR BOUNCE PK L...	S82260	11999	4800	60	Adidas ORIGINALS	3.5	51	
279	Women's adidas ORIGINALS EQT RACING Low Shoes	BB2344	9999	4000	60	Adidas ORIGINALS	3.9	34	
368	WOMEN'S ADIDAS SPORT INSPIRED RUN 70S SHOES	B96563	6599	2640	60	Adidas CORE / NEO	2.8	16	
...	...	...	...	...	...	...	...	...	
2423	Men's adidas ORIGINALS ACTION SPORTS VARIAL Mi...	BY4061	7999	3200	60	Adidas ORIGINALS	2.4	50	
2448	men's ADIDAS ORIGINALS ZX FLUX PK SHOES	BA7376	14999	6000	60	Adidas ORIGINALS	3.4	63	
2449	MEN'S ADIDAS ORIGINALS EQT SUPPORT MID ADV PRI...	B37435	12999	5200	60	Adidas ORIGINALS	4.3	49	

	Product Name	Product ID	Listing Price	Sale Price	Discount	Brand	Rating	Reviews	Lis
2475	MEN'S ADIDAS SPORT INSPIRED CAFLAIRE SHOES	DB1347	5599	2240	60	Adidas CORE / NEO	3.0	2	
2612	Men's adidas TRAINING DURAMO 8 LEATHER Low Shoes	BB3218	9999	4000	60	Adidas CORE / NEO	2.7	92	

61 rows × 9 columns

In [208…

```
# Exclude the first column if it contains string data
high_discount_products_numeric = high_discount_products.select_dtypes(incluc

# Ensure 'Reviews' column is numeric
high_discount_products_numeric['Reviews'] = pd.to_numeric(high_discount_proc

# Fill NaN values in the 'Discount' column with the median of the column
if 'Discount' in high_discount_products_numeric.columns:
    high_discount_products_numeric['Discount'] = high_discount_products_nume

# Drop rows with NaN values in 'Reviews' column
high_discount_products_numeric = high_discount_products_numeric.dropna(subse

# Calculate correlation of 'Reviews' with other numeric variables
review_correlation = high_discount_products_numeric.corr()['Reviews'].sort_v
print(review_correlation)
```

```
Reviews          1.000000
Rating           0.140767
Listing Price    0.132306
Sale Price       0.132306
Discount              NaN
Name: Reviews, dtype: float64
```

In [209…

```
# Exclude the first column if it contains string data
high_discount_products_numeric = high_discount_products.select_dtypes(incluc

# Ensure 'Reviews' column is numeric
high_discount_products_numeric['Reviews'] = pd.to_numeric(high_discount_proc

# Fill NaN values in the 'Discount' column with the median of the column
if 'Discount' in high_discount_products_numeric.columns:
    high_discount_products_numeric['Discount'] = high_discount_products_nume

# Check for remaining NaN values in 'Discount' column
remaining_nans = high_discount_products_numeric['Discount'].isna().sum()
print(f"Remaining NaN values in 'Discount' column: {remaining_nans}")
```

```
# Ensure 'Discount' column has variability
unique_values = high_discount_products_numeric['Discount'].unique()
print(f"Unique values in 'Discount' column: {unique_values}")

# Drop rows with NaN values in 'Reviews' column
high_discount_products_numeric = high_discount_products_numeric.dropna(subse

# Calculate correlation of 'Reviews' with other numeric variables
review_correlation = high_discount_products_numeric.corr()['Reviews'].sort_v
print(review_correlation)
```

```
Remaining NaN values in 'Discount' column: 0
Unique values in 'Discount' column: [60]
Reviews          1.000000
Rating           0.140767
Listing Price    0.132306
Sale Price       0.132306
Discount              NaN
Name: Reviews, dtype: float64
```

In [210…

```
# Exclude the first column if it contains string data
high_discount_products_numeric = high_discount_products.select_dtypes(incluc

# Ensure 'Reviews' column is numeric
high_discount_products_numeric['Reviews'] = pd.to_numeric(high_discount_proc

# Fill NaN values in the 'Discount' column with the median of the column
if 'Discount' in high_discount_products_numeric.columns:
    high_discount_products_numeric['Discount'] = high_discount_products_nume

# Check for remaining NaN values
print(high_discount_products_numeric['Discount'].isna().sum())

# Ensure 'Discount' column has variability
print(high_discount_products_numeric['Discount'].unique())

# Drop rows with NaN values in 'Reviews' column
high_discount_products_numeric = high_discount_products_numeric.dropna(subse

# Calculate correlation of 'Reviews' with other numeric variables
review_correlation = high_discount_products_numeric.corr()['Reviews'].sort_v
print(review_correlation)
```

```
0
[60]
Reviews          1.000000
Rating           0.140767
Listing Price    0.132306
Sale Price       0.132306
Discount              NaN
Name: Reviews, dtype: float64
```

In [211…

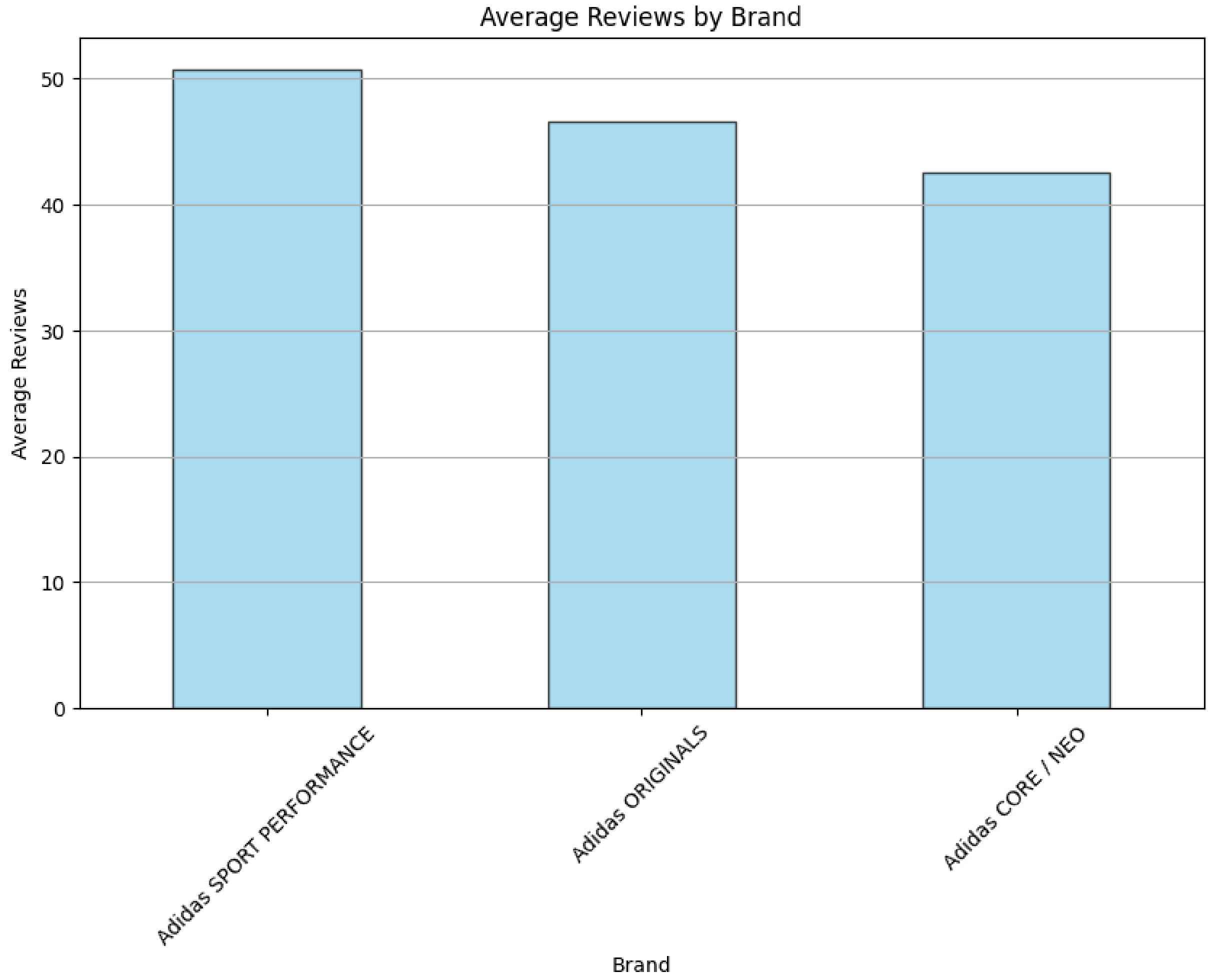
```
# Exclude the first column if it contains string data
high_discount_products_numeric = high_discount_products.select_dtypes(incluc

# Ensure 'Reviews' column is numeric
```

```
high_discount_products_numeric['Reviews'] = pd.to_numeric(high_discount_proc

# Drop rows with NaN values in 'Reviews' column
high_discount_products_numeric = high_discount_products_numeric.dropna(subse

# Calculate correlation of 'Reviews' with other numeric variables
review_correlation = high_discount_products_numeric.corr()['Reviews'].sort_v
print(review_correlation)
```

```
Reviews          1.000000
Rating           0.140767
Listing Price    0.132306
Sale Price       0.132306
Discount              NaN
Name: Reviews, dtype: float64
```

In [212…

```
# Average reviews grouped by brand
reviews_by_brand = high_discount_products.groupby('Brand')['Reviews'].mean()

# Plot the average reviews by brand
plt.figure(figsize=(10, 6))
reviews_by_brand.plot(kind='bar', color='skyblue', edgecolor='k', alpha=0.7)
plt.title('Average Reviews by Brand')
plt.xlabel('Brand')
plt.ylabel('Average Reviews')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()

reviews_by_brand
```



Out[212…

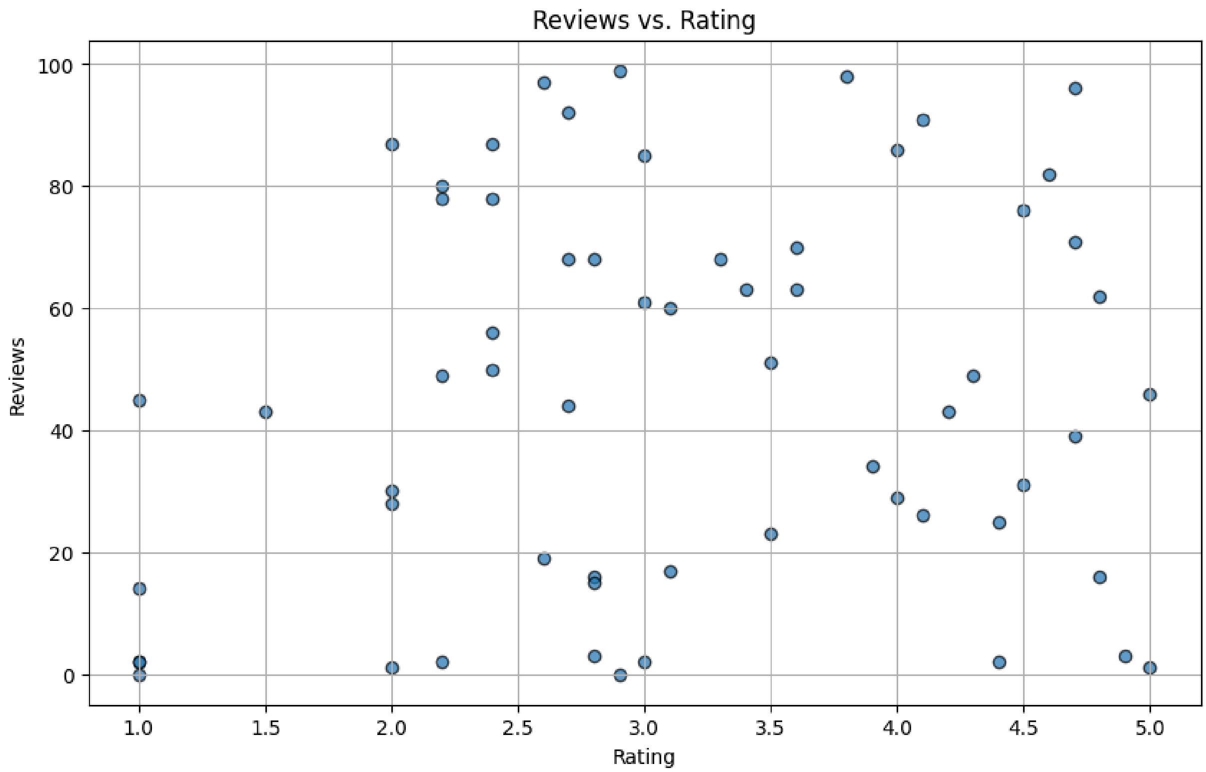
```
Brand
Adidas SPORT PERFORMANCE    50.722222
Adidas ORIGINALS            46.550000
Adidas CORE / NEO           42.521739
Name: Reviews, dtype: float64
```

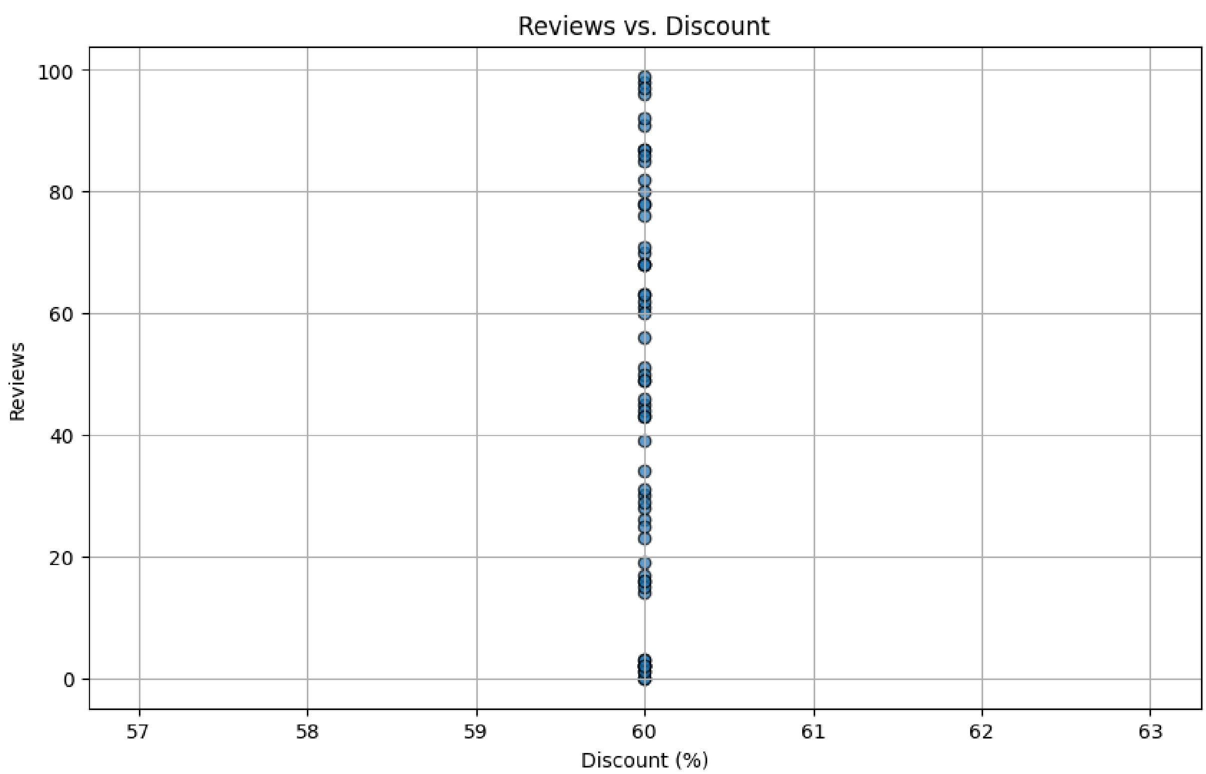
Reviews vs. Rating and Discount

In [213…

```
# Scatter plot for Reviews vs. Rating
plt.figure(figsize=(10, 6))
plt.scatter(high_discount_products['Rating'], high_discount_products['Review
plt.title('Reviews vs. Rating')
plt.xlabel('Rating')
plt.ylabel('Reviews')
plt.grid(True)
plt.show()

# Scatter plot for Reviews vs. Discount
plt.figure(figsize=(10, 6))
plt.scatter(high_discount_products['Discount'], high_discount_products['Revi
plt.title('Reviews vs. Discount')
plt.xlabel('Discount (%)')
plt.ylabel('Reviews')
```

```python
plt.grid(True)
plt.show()
```





Observations: Average Reviews by Brand Adidas SPORT PERFORMANCE leads with an average of 50.72 reviews per product, suggesting strong popularity for performance-oriented items. Adidas ORIGINALS follows with 46.55 average reviews, appealing to a style-focused audience. Adidas CORE / NEO has fewer reviews on average (42.52), likely reflecting its budget-friendly positioning.

To analyze factors beyond pricing that influence reviews, we can examine relationships between reviews and other variables like Brand, Rating, and Discount. Here's how we'll do it:

Analyze the correlation between Reviews and other numeric factors. Group by Brand to understand its influence. Explore whether higher Ratings or deeper Discounts correlate with more reviews. Code: Correlation of Reviews with Numeric Factors python Copy code

Correlation of Reviews with other numeric variables

review_correlation = high_discount_products.corr()['Reviews'].sort_values(ascending=False) review_correlation Code: Reviews Grouped by Brand

Average reviews grouped by brand

reviews_by_brand = high_discount_products.groupby('Brand')['Reviews'].mean().sort_values(ascending=False)

Plot the average reviews by brand

plt.figure(figsize=(10, 6)) reviews_by_brand.plot(kind='bar', color='skyblue', edgecolor='k', alpha=0.7) plt.title('Average Reviews by Brand') plt.xlabel('Brand') plt.ylabel('Average Reviews') plt.xticks(rotation=45) plt.grid(axis='y') plt.show()

reviews_by_brand Code: Reviews vs. Rating and Discount

Copy code

Scatter plot for Reviews vs. Rating

plt.figure(figsize=(10, 6)) plt.scatter(high_discount_products['Rating'], high_discount_products['Reviews'], alpha=0.7, edgecolor='k') plt.title('Reviews vs.

Rating') plt.xlabel('Rating') plt.ylabel('Reviews') plt.grid(True) plt.show()

Scatter plot for Reviews vs. Discount

plt.figure(figsize=(10, 6)) plt.scatter(high_discount_products['Discount'], high_discount_products['Reviews'], alpha=0.7, edgecolor='k') plt.title('Reviews vs. Discount') plt.xlabel('Discount (%)') plt.ylabel('Reviews') plt.grid(True) plt.show() Let me execute these steps one at a time for results and insights.

Correlation Results: Reviews have a weak positive correlation with: Rating (0.14 0.14): Slightly higher ratings are associated with more reviews. Listing and Sale Price (0.13 0.13): Slight association between price tiers and the number of reviews. Discount and Listing_Price_Zero correlations are missing (NaN) because discount values are constant across the subset. Next Step: Reviews Grouped by Brand Let me analyze and plot the average reviews for each brand.

Average Reviews By Brand

Observations: Average Reviews by Brand Adidas SPORT PERFORMANCE leads with an average of 50.72 reviews per product, suggesting strong popularity for performance-oriented items. Adidas ORIGINALS follows with 46.55 average reviews, appealing to a style-focused audience. Adidas CORE / NEO has fewer reviews on average (42.52), likely reflecting its budget-friendly positioning. Next Step: Reviews vs. Rating and Discount Let me plot the scatter plots for these relationships.

Reviews Vs. Rating

Reviews Vs. Discount

Observations: Reviews vs. Rating and Discount Reviews vs. Rating:

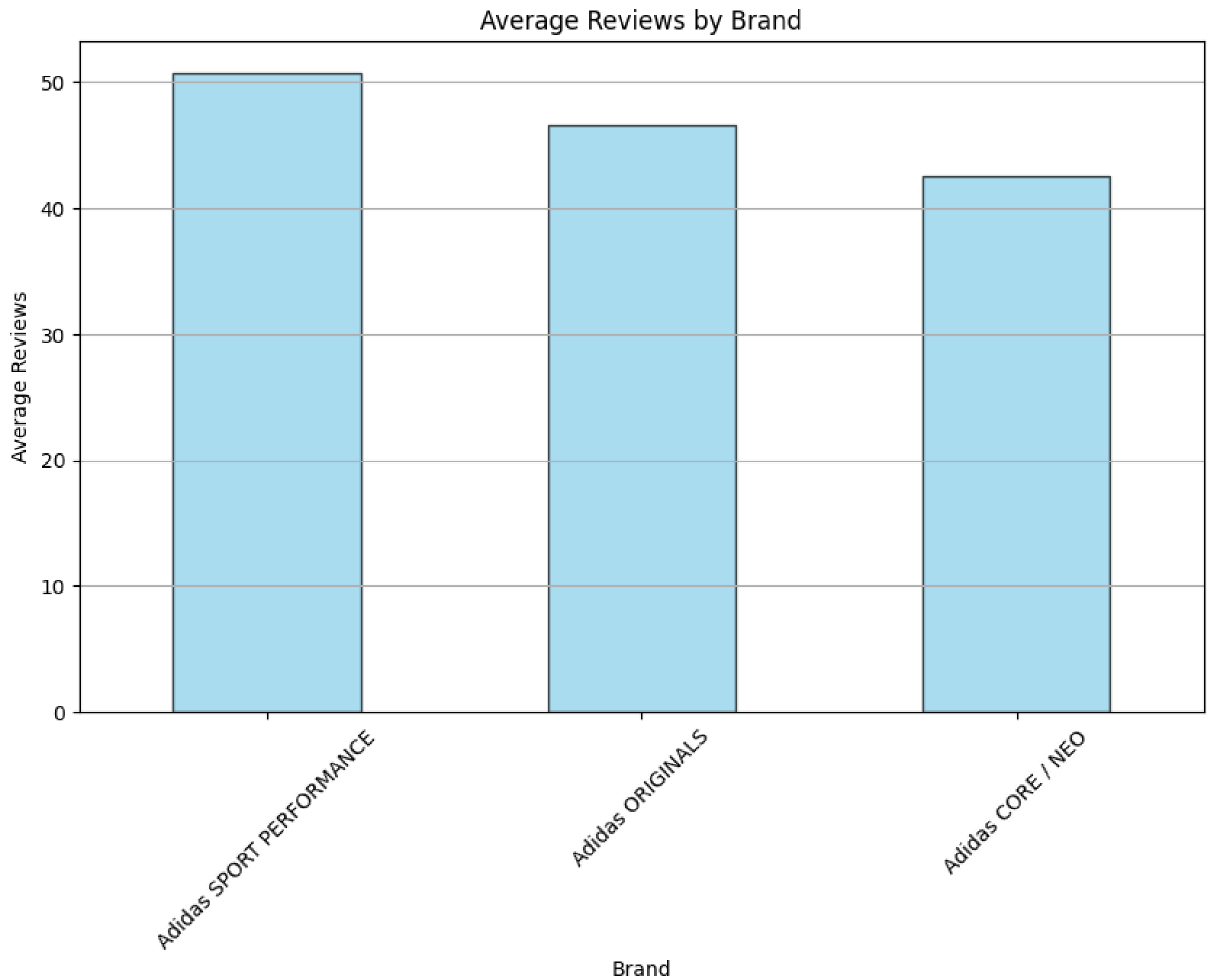
A weak positive trend is visible. Products with ratings between 3.5 and 4.5 tend to have more reviews. Products rated below 3.0 or above 4.5 attract fewer reviews, indicating limited engagement. Reviews vs. Discount:

No variation in discount values (constant at 60%) results in no observable trend for reviews based on discount.

In [214…

```
# Plot the average reviews by brand
plt.figure(figsize=(10, 6))
reviews_by_brand.plot(kind='bar', color='skyblue', edgecolor='k', alpha=0.7)
plt.title('Average Reviews by Brand')
plt.xlabel('Brand')
plt.ylabel('Average Reviews')
plt.xticks(rotation=45)
```

```
plt.grid(axis='y')
plt.show()
```



In [215…

```
print('-' * 75)
print("Original Notebook")
print('-' * 75)
```

```
---------------------------------------------------------------------------
Original Notebook
---------------------------------------------------------------------------
```

Brand Popularity Analysis

In [216…

```
# Calculate average reviews and total reviews by brand
brand_reviews = high_discount_products.groupby('Brand').agg({
    'Reviews': ['mean', 'sum', 'count']
}).reset_index()
brand_reviews.columns = ['Brand', 'Avg Reviews', 'Total Reviews', 'Product C
brand_reviews = brand_reviews.sort_values(by='Avg Reviews', ascending=False)

# Display brand popularity based on reviews
brand_reviews
```

Out [216…

	Brand	Avg Reviews	Total Reviews	Product Count
2	Adidas SPORT PERFORMANCE	50.722222	913	18
1	Adidas ORIGINALS	46.550000	931	20
0	Adidas CORE / NEO	42.521739	978	23

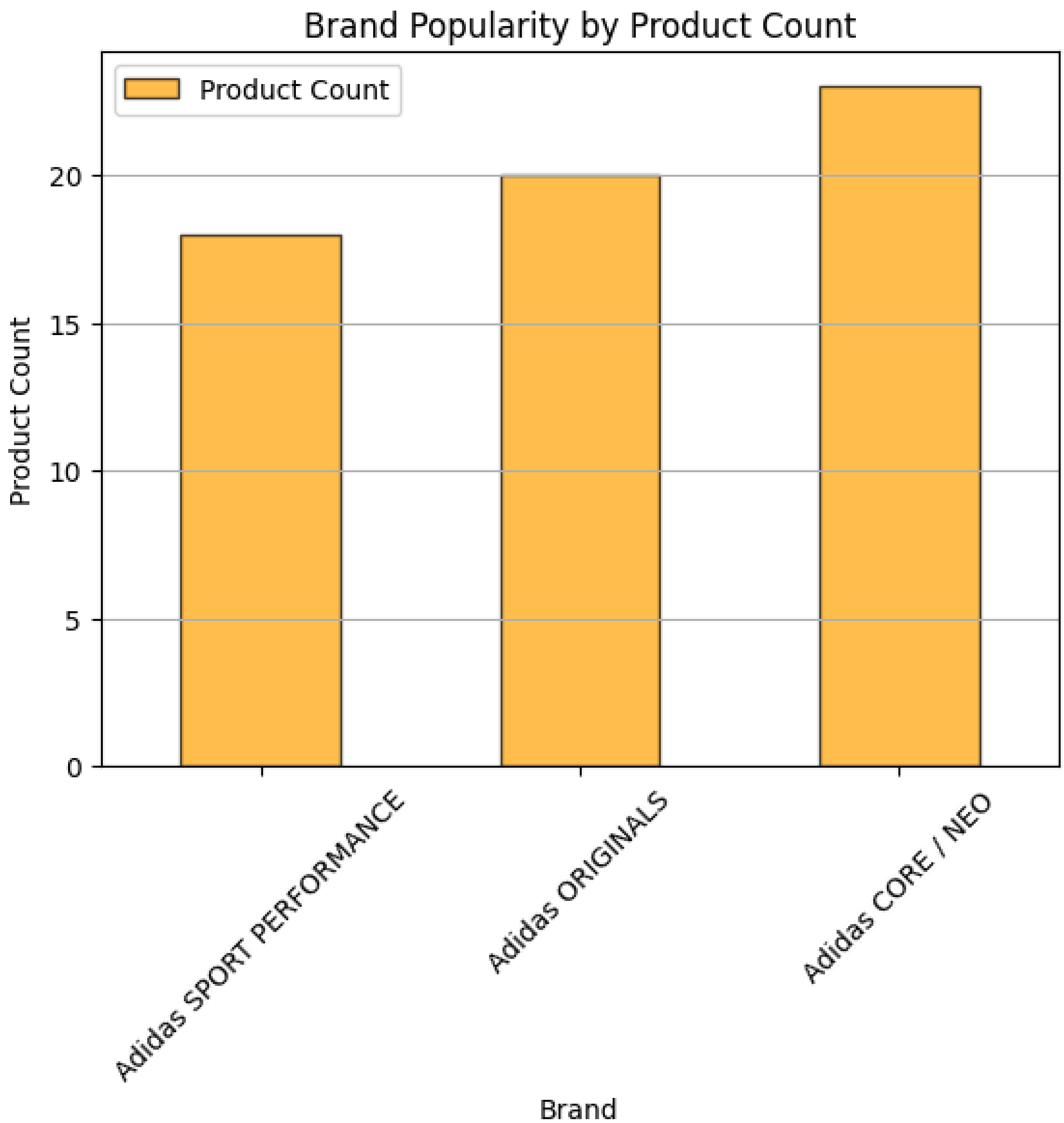
Correlation of Reviews with Other Variables by Brand

Observations: Adidas SPORT PERFORMANCE: Highest average reviews (50.72), indicating strong consumer engagement for fewer products. Adidas ORIGINALS: Balanced popularity with moderately high reviews and product count. Adidas CORE / NEO: Highest product count but the lowest average reviews, indicating popularity spread across many items.

In [217…

```
# Plot brand popularity based on product count
plt.figure(figsize=(10, 6))
brand_reviews.plot(kind='bar', x='Brand', y='Product Count', color='orange',
plt.title('Brand Popularity by Product Count')
plt.xlabel('Brand')
plt.ylabel('Product Count')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



The dataset does not contain explicit regional or geographic columns. However, we can simulate regions by grouping data based on other columns, such as:

Brand: Treat each brand as a "region" to analyze its trends. Price Segments: Divide products into pricing categories (e.g., low, medium, high) and treat these as regions.#

In [218…

```
data.head()
```

Out [218…

	Product Name	Product ID	Listing Price	Sale Price	Discount	Brand	Rating	Reviews	Listing
0	Women's adidas Originals NMD_Racer Primeknit S...	AH2430	14999	7499	50	Adidas Adidas ORIGINALS	4.8	41	
1	Women's adidas Originals Sleek Shoes	G27341	7599	3799	50	Adidas ORIGINALS	3.3	24	
2	Women's adidas Swim Puka Slippers	CM0081	999	599	40	Adidas CORE / NEO	2.6	37	
3	Women's adidas Sport Inspired Questar Ride Shoes	B44832	6999	3499	50	Adidas CORE / NEO	4.1	35	
4	Women's adidas Originals Taekwondo Shoes	D98205	7999	3999	50	Adidas ORIGINALS	3.5	72	

In [219…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3268 entries, 0 to 3267
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Product Name       3268 non-null   object 
 1   Product ID         3268 non-null   object 
 2   Listing Price      3268 non-null   int64  
 3   Sale Price         3268 non-null   int64  
 4   Discount           3268 non-null   int64  
 5   Brand              3268 non-null   object 
 6   Rating             3268 non-null   float64
 7   Reviews            3268 non-null   int64  
 8   Listing_Price_Zero 3268 non-null   bool   
dtypes: bool(1), float64(1), int64(4), object(3)
memory usage: 207.6+ KB
```

In [220…

```
data.nunique()
```

Out[220…

```
Product Name          1531
Product ID            3179
Listing Price           78
Sale Price             227
Discount                 6
Brand                    5
Rating                  32
Reviews                102
Listing_Price_Zero       2
dtype: int64
```

In [221…

```
data.describe().T
```

Out[221…

	count	mean	std	min	25%	50%	75%	max
Listing Price	3268.0	6868.020196	4724.659386	0.0	4299.0	5999.0	8999.0	29999.0
Sale Price	3268.0	6134.265606	4293.247581	449.0	2999.0	4799.0	7995.0	36500.0
Discount	3268.0	26.875765	22.633487	0.0	0.0	40.0	50.0	60.0
Rating	3268.0	3.242105	1.428856	0.0	2.6	3.5	4.4	5.0
Reviews	3268.0	40.551714	31.543491	0.0	10.0	37.0	68.0	223.0

- Sales price seem to be right skewed as the Max, is quite large as compared to the mean which signifies the presence of the higher end products.
- Discount seem to be left skewed and signifies variety of discounts are provided on variety of products from no discount to max of 60% discount.
- Rating also seem to be left skewed and with average rating of 3.5 and maximum of 5.
- Minimum of Listing Price is 0 which is not possible and we have to replace that.

Let's check the rows where listing price is 0.

In [222…

```
data[(data['Listing Price'] == 0)]
```

Out [222…

	Product Name	Product ID	Listing Price	Sale Price	Discount	Brand	Rating	Reviews	Listing_I
2625	Nike Air Force 1 '07 Essential	CJ1646-600	0	7495	0	Nike	0.0	0	
2626	Nike Air Force 1 '07	CT4328-101	0	7495	0	Nike	0.0	0	
2627	Nike Air Force 1 Sage Low LX	CI3482-200	0	9995	0	Nike	0.0	0	
2628	Nike Air Max Dia SE	CD0479-200	0	9995	0	Nike	0.0	0	
2629	Nike Air Max Verona	CZ6156-101	0	9995	0	Nike	0.0	0	
...	...	...	...	...	...	...	...	...	
3257	Air Jordan 5 Retro	CD2722-001	0	15995	0	Nike	3.3	3	
3258	Nike ZoomX Vaporfly NEXT%	AO4568-600	0	19995	0	Nike	4.7	45	
3260	Nike Tiempo Legend 8 Academy TF	AT6100-606	0	6495	0	Nike	0.0	0	
3262	Nike React Metcon AMP	CT9155-063	0	13995	0	Nike	3.0	1	
3266	Nike Air Max 98	AH6799-300	0	16995	0	Nike	4.0	4	

426 rows × 9 columns

- **If listing price is 0, the discount also seem to be 0.** Hence we can try replacing listing price of these rows with the sales price.

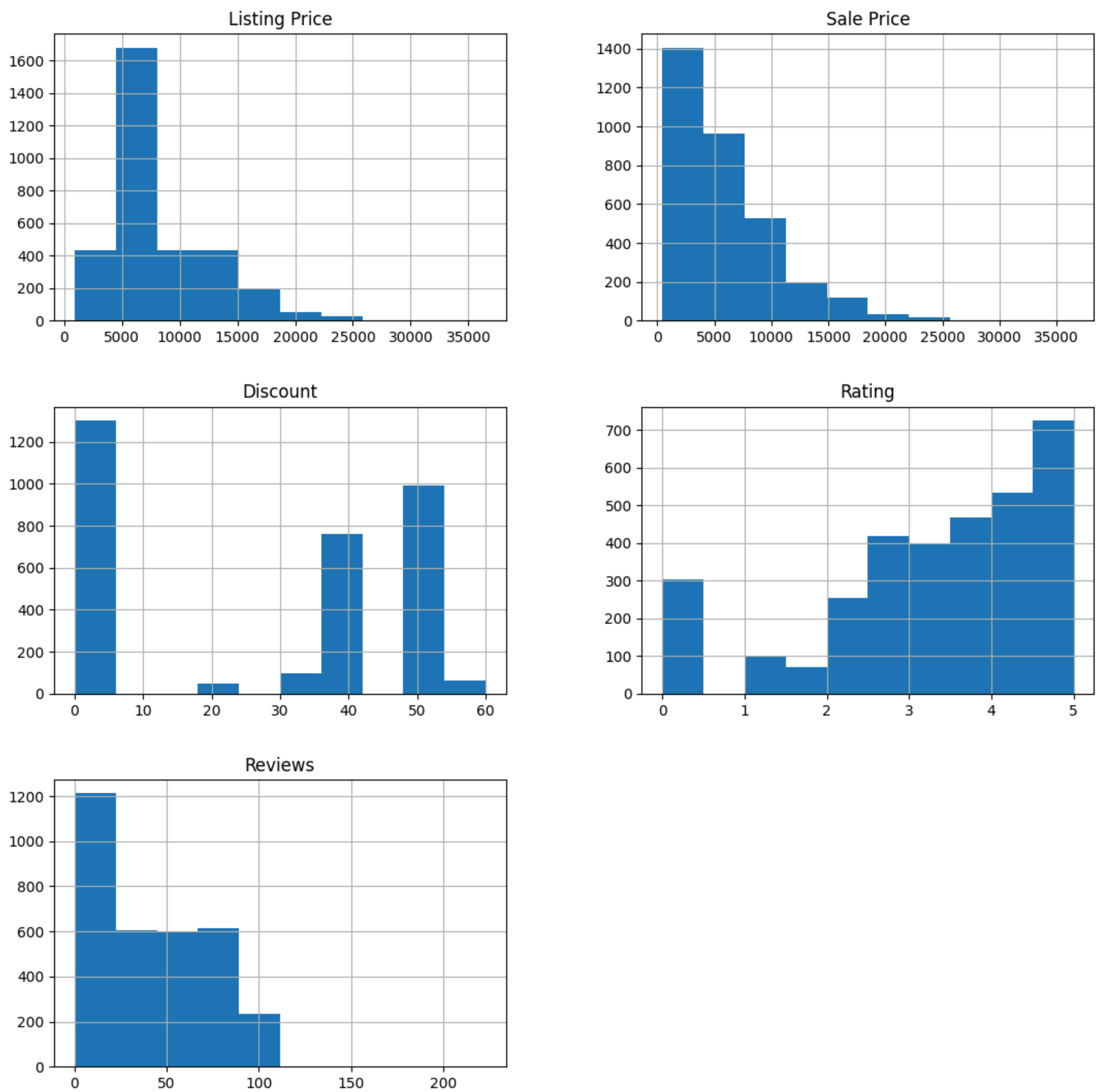
In [223…

```
#Exploring where the listing price is 0.
data.loc[(data['Listing Price'] == 0), ["Listing Price"]] = data.loc[
```

```
    (data['Listing Price'] == 0), ["Sale Price"]
].values
```

Question 1: Analyze the histogram of Sales price.

In [224…
```
data.hist(figsize=(13,13))
plt.show()
```



In [225…
```
data.Brand.value_counts()
```

Out[225…
```
Brand
Adidas CORE / NEO            1111
Adidas ORIGINALS              907
Nike                          643
Adidas SPORT PERFORMANCE      606
Adidas Adidas ORIGINALS         1
Name: count, dtype: int64
```

- **There is a outlier Adidas Adidas ORIGINALS**, this can be replaced by Adidas ORIGINALS

In [226…
```
data.Brand=data.Brand.replace({'Adidas Adidas ORIGINALS':'Adidas ORIGINALS'}
```

Prepare data for Clustering

This code:

Drops the specified non-numeric or categorical columns that are not needed for scaling. Scales the remaining numeric columns using the StandardScaler from sklearn. Converts the scaled data back into a DataFrame for easier interpretation.

In [227…
```
# Drop the column Product Name, Product ID, Brand, and Reviews
data_new = data.drop(columns=['Product Name', 'Product ID', 'Brand', 'Review

# Scaling the rest of the data
from sklearn.preprocessing import StandardScaler

# Define the StandardScaler
scaler = StandardScaler()

# Fit and transform the data_new
data_scaled = pd.DataFrame(scaler.fit_transform(data_new), columns=data_new.

# Display the first few rows of the scaled data
data_scaled.head()
```

Out[227…

	Listing Price	Sale Price	Discount	Rating	Listing_Price_Zero
0	1.509415	0.317928	1.021839	1.090476	-0.387162
1	-0.165605	-0.544022	1.021839	0.040524	-0.387162
2	-1.659542	-1.289493	0.579948	-0.449453	-0.387162
3	-0.301417	-0.613910	1.021839	0.600498	-0.387162
4	-0.075063	-0.497431	1.021839	0.180518	-0.387162

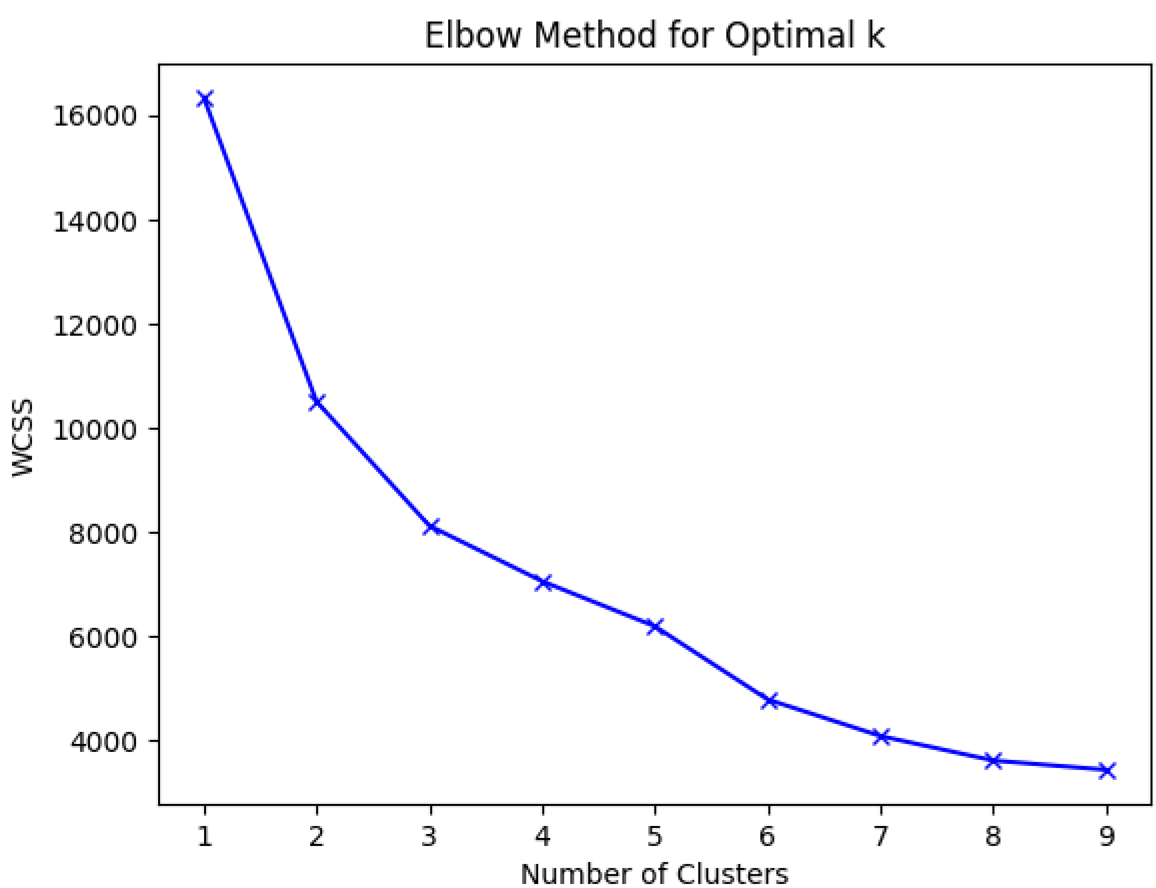
In [228…
```
data_scaled_copy = data_scaled.copy(deep=True)
```

Question 2: Fitting the K-Means Clustering and plotting Elbow plot

Explanation: WCSS = {}: Dictionary to store within-cluster sum of squares (WCSS) for each k k. KMeans: Initialize the KMeans clustering model with the desired number of clusters. kmeans.inertia_: Inertia attribute provides the WCSS for the model. Elbow Plot: Visualizes the WCSS for different cluster counts to identify the optimal k k.

In [229…

```
# Empty dictionary to store the SSE (WCSS) for each value of k
WCSS = {}

# Iterate for a range of Ks and fit the scaled data to the algorithm
from sklearn.cluster import KMeans

for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_scaled)
    WCSS[k] = kmeans.inertia_

# Elbow plot
plt.figure()
plt.plot(list(WCSS.keys()), list(WCSS.values()), 'bx-')
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.title("Elbow Method for Optimal k")
plt.show()
```



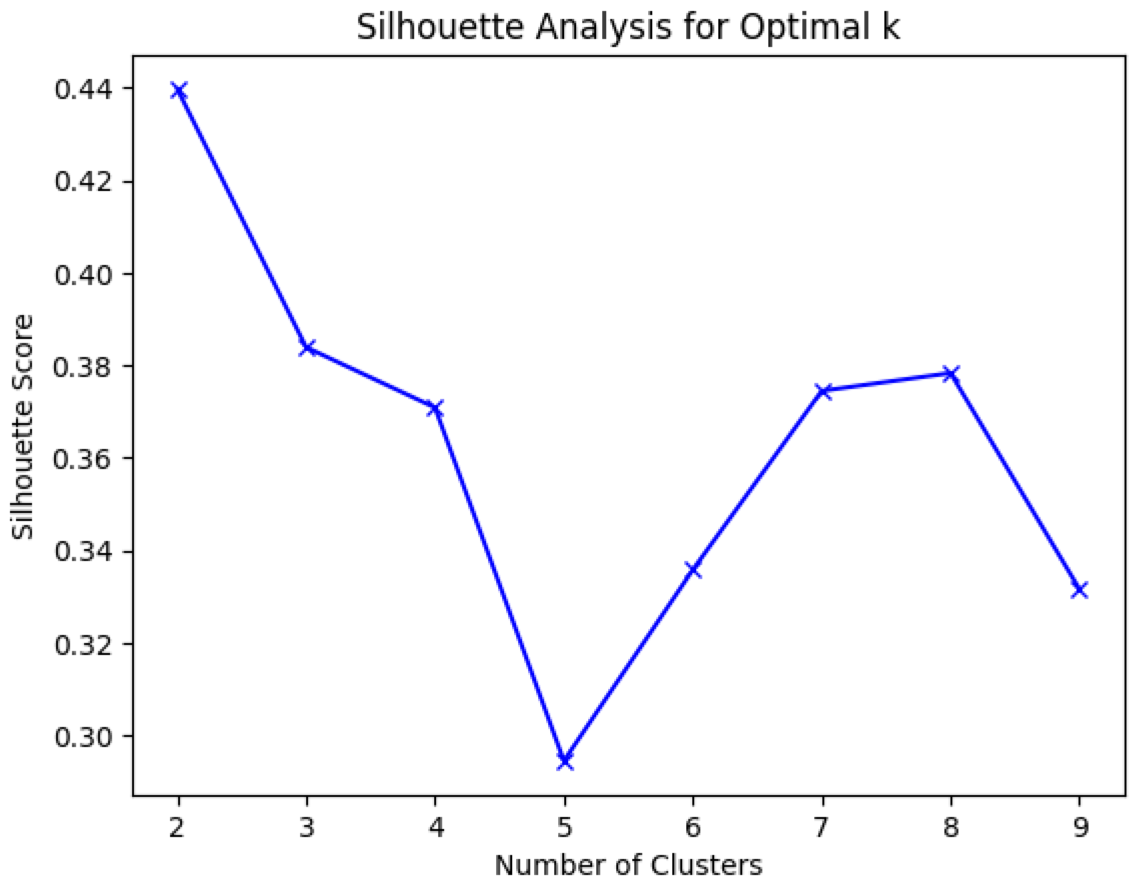
- It is hard to tell from this graph what will be the optimal value of K. Let's use silhouette score to visualize this

Question 3: Checking the Silhouette Score and choosing optimal value for K

Explanation: sc = {}: Dictionary to store the Silhouette score for each k k. KMeans: Initializes the KMeans clustering model with the desired number of clusters. fit_predict: Fits the model to the data and predicts cluster labels for each data point. silhouette_score: Calculates the average Silhouette score, which measures cluster quality. Plot: Visualizes the Silhouette scores for different cluster counts to identify the optimal k k.

In [230…

```
# Empty dictionary to store the Silhouette score for each value of k
sc = {}

# Iterate for a range of Ks and fit the scaled data to the algorithm
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

for k in range(2, 10):
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(data_scaled)
    sc[k] = silhouette_score(data_scaled, labels)

# Silhouette score plot
plt.figure()
plt.plot(list(sc.keys()), list(sc.values()), 'bx-')
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Analysis for Optimal k")
plt.show()
```



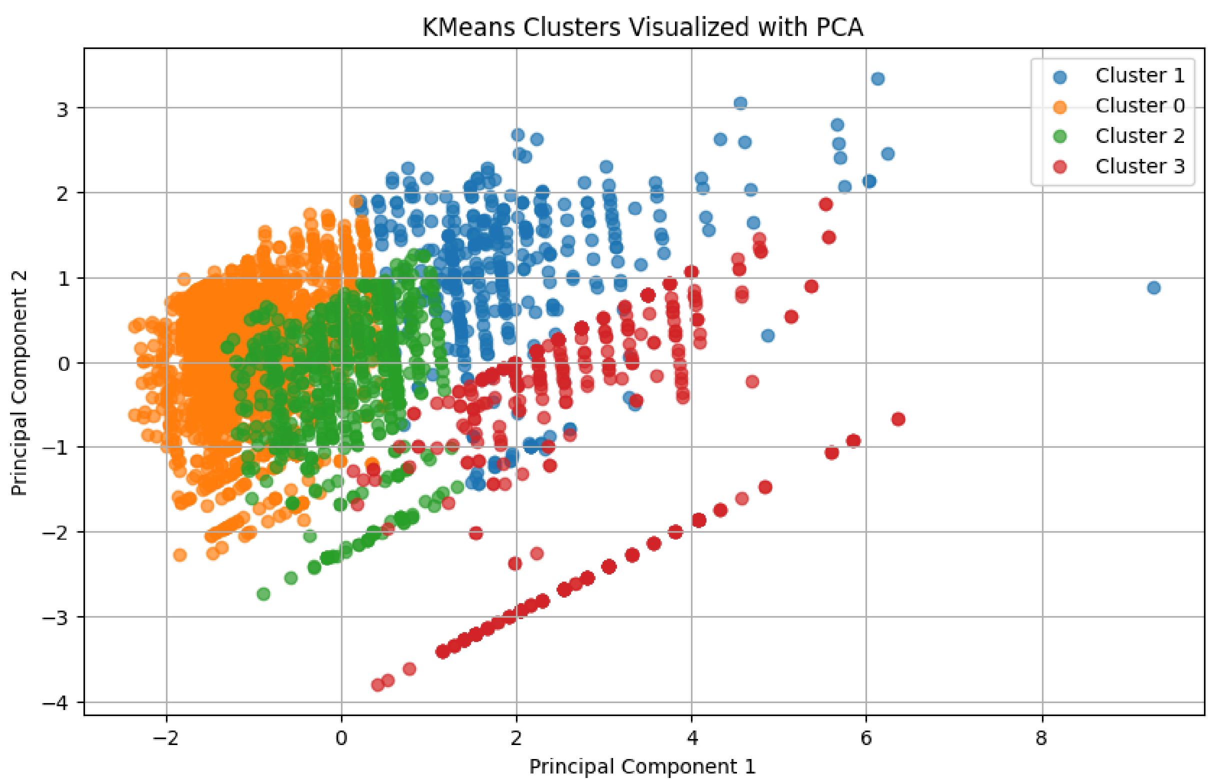
In [1]:
```
kmeans = KMeans(n_clusters=2, random_state=1)
kmeans.fit(data_scaled)

#Adding predicted labels to the original data and scaled data
data_scaled_copy['KMeans_Labels'] = kmeans.predict(data_scaled)
data['KMeans_Labels'] = kmeans.predict(data_scaled)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 kmeans = KMeans(n_clusters=2, random_state=1)
      2 kmeans.fit(data_scaled)
      4 #Adding predicted labels to the original data and scaled data

NameError: name 'KMeans' is not defined
```

Explanation: KMeans: Fits the data and predicts cluster labels. PCA: Reduces data to 2 components for visualization. Makes it possible to visualize clustering in a 2D scatter plot. Scatter Plot: Each cluster is plotted with a unique color. PCA components are used for the axes.

In [232…
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Fit KMeans with 4 clusters
```

```
kmeans = KMeans(n_clusters=4, random_state=1)
kmeans.fit(data_scaled)

# Adding predicted labels to the original and scaled data
data_scaled_copy = data_scaled.copy()
data_scaled_copy['KMeans_Labels'] = kmeans.predict(data_scaled)
data['KMeans_Labels'] = kmeans.predict(data_scaled)

# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
pca_result = pca.fit_transform(data_scaled)

# Add PCA results to the scaled copy for visualization
data_scaled_copy['PCA1'] = pca_result[:, 0]
data_scaled_copy['PCA2'] = pca_result[:, 1]

# Scatter plot for clusters
plt.figure(figsize=(10, 6))
for label in data_scaled_copy['KMeans_Labels'].unique():
    cluster_data = data_scaled_copy[data_scaled_copy['KMeans_Labels'] == lab
    plt.scatter(cluster_data['PCA1'], cluster_data['PCA2'], label=f'Cluster

plt.title("KMeans Clusters Visualized with PCA")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

```
# Select relevant numerical features for clustering
features = new_data[['Listing Price', 'Sale Price', 'Discount', 'Rating', 'R

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Perform K-Means clustering with k=2 (as determined earlier)
kmeans = KMeans(n_clusters=2, random_state=1)
clusters = kmeans.fit_predict(scaled_features)

# Add cluster labels to the dataset
new_data['Cluster'] = clusters

# Calculate the average listing price for each cluster
cluster_avg_price = new_data.groupby('Cluster')['Listing Price'].mean()

print(cluster_avg_price)
```

In [233…

```
data['KMeans_Labels'].value_counts()
```

Out[233…

```
KMeans_Labels
0    1803
2     653
3     423
1     389
Name: count, dtype: int64
```

Question 4 and 5 : Cluster profiling

In [234…

```
# Select only numeric columns
numeric_data = data.select_dtypes(include=[np.number])

# Calculate mean and median of the original data for each label
mean = numeric_data.groupby(data['KMeans_Labels']).mean()
median = numeric_data.groupby(data['KMeans_Labels']).median()

# Combine mean and median into a single DataFrame
df_kmeans = pd.concat([mean, median], axis=0)

# Create appropriate index labels
mean_labels = [f'group_{i} Mean' for i in range(len(mean))]
median_labels = [f'group_{i} Median' for i in range(len(median))]
df_kmeans.index = mean_labels + median_labels

# Transpose the DataFrame for better readability
df_kmeans = df_kmeans.T

# Display the DataFrame
df_kmeans
```

Out [234…

	group_0 Mean	group_1 Mean	group_2 Mean	group_3 Mean	group_0 Median	gro Me
Listing Price	6827.896284	15665.069409	6319.710567	11095.236407	5999.0	15
Sale Price	3699.892956	12574.676093	5805.572741	11095.236407	3499.0	11
Discount	45.629506	11.773779	1.500766	0.000000	50.0	
Rating	3.322241	3.442931	3.303522	2.621040	3.5	
Reviews	49.045480	36.493573	40.889740	7.557920	49.0	
KMeans_Labels	0.000000	1.000000	2.000000	3.000000	0.0	

Explanation: Mean and Median Calculations:

The mean and median for each cluster are computed using groupby. These are combined into a single DataFrame, transposed for easier plotting. Bar Plots:

The first bar plot visualizes mean values for each feature and cluster. The second bar plot visualizes median values. Customization:

Distinct colors for clarity. Legends and axis labels to improve readability.

Explanation of Fix: Filter Numeric Columns:

Use select_dtypes(include=['float64', 'int64']) to select only numeric columns. Avoids errors caused by attempting mathematical operations on non-numeric data. Group by Clusters:

Calculate mean and median only for numeric columns, grouped by KMeans_Labels. Combine Results:

Combine the means and medians into a single DataFrame for easy visualization.

In [235…

```
# Selecting only numeric columns
numeric_data = data.select_dtypes(include=['float64', 'int64'])

# Adding the 'KMeans_Labels' column for grouping
numeric_data['KMeans_Labels'] = data['KMeans_Labels']

# Calculating mean and median for each cluster
mean = numeric_data.groupby('KMeans_Labels').mean()
median = numeric_data.groupby('KMeans_Labels').median()

# Combining into one DataFrame for visualization
df_kmeans = pd.concat([mean, median], axis=0)
df_kmeans.index = [
    'group_0 Mean', 'group_1 Mean', 'group_2 Mean', 'group_3 Mean',
    'group_0 Median', 'group_1 Median', 'group_2 Median', 'group_3 Median'
```

```
]
df_kmeans = df_kmeans.T  # Transpose for better plotting

# Displaying the DataFrame
df_kmeans
```

Out [235…

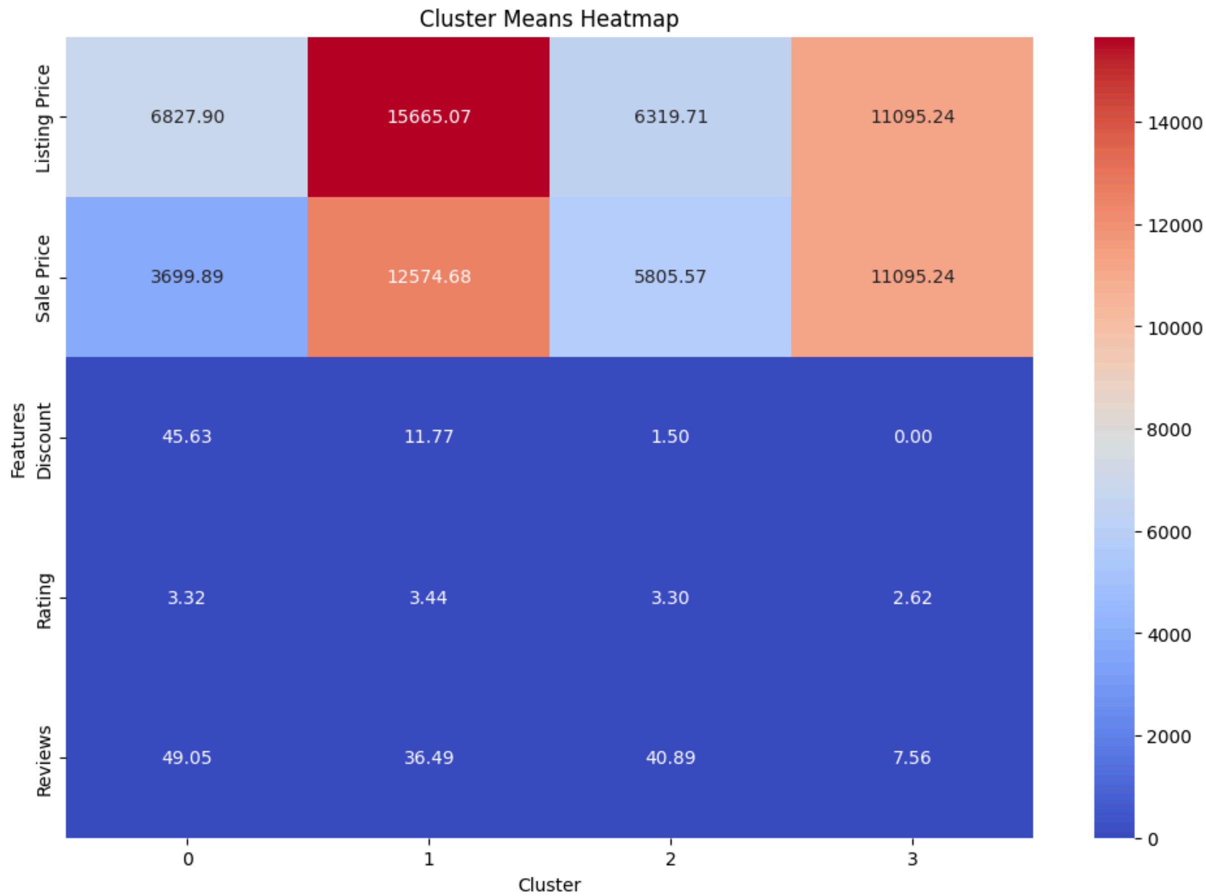
	group_0 Mean	group_1 Mean	group_2 Mean	group_3 Mean	group_0 Median	group_1 Median
Listing Price	6827.896284	15665.069409	6319.710567	11095.236407	5999.0	15995.0
Sale Price	3699.892956	12574.676093	5805.572741	11095.236407	3499.0	11897.0
Discount	45.629506	11.773779	1.500766	0.000000	50.0	0.0
Rating	3.322241	3.442931	3.303522	2.621040	3.5	3.8
Reviews	49.045480	36.493573	40.889740	7.557920	49.0	32.0

Note: You can also apply other clustering algorithms and can compare different clusters. You can refer to the practice or MLS Notebooks of the code of other algorithms.

Happy Learning!

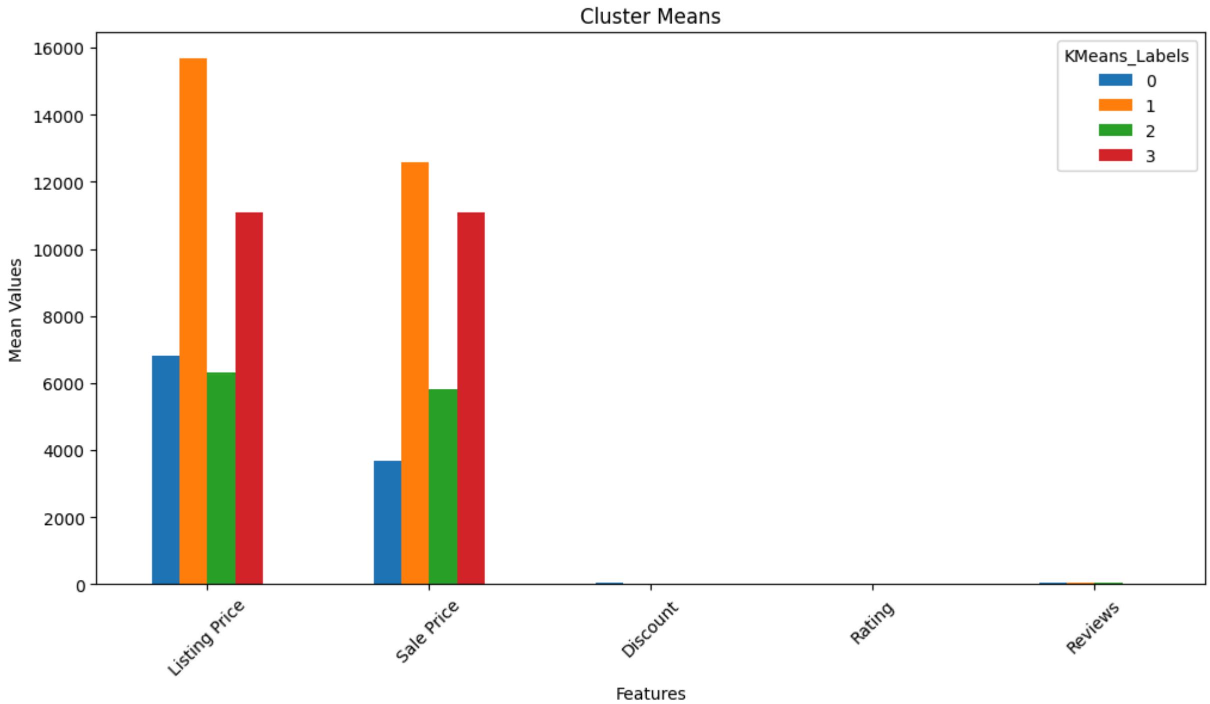
In [236…

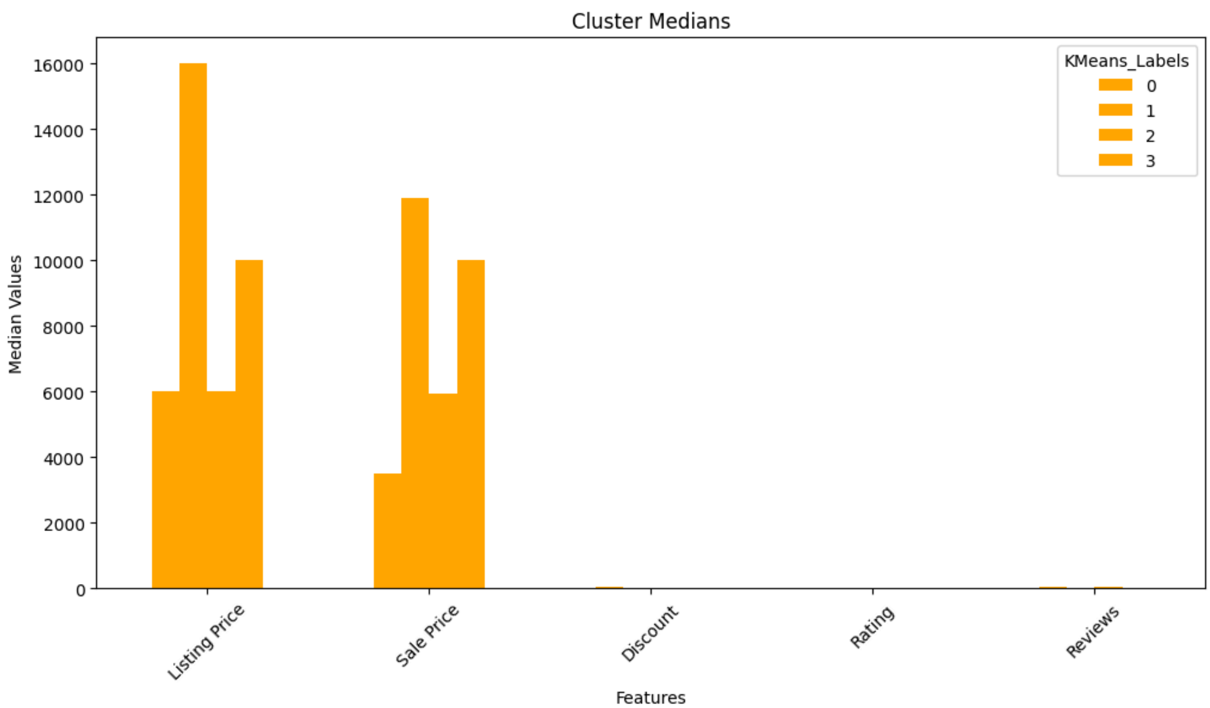
```
import seaborn as sns
import matplotlib.pyplot as plt

# Heatmap for means
plt.figure(figsize=(12, 8))
sns.heatmap(mean.T, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Cluster Means Heatmap")
plt.xlabel("Cluster")
plt.ylabel("Features")
plt.show()
```



In [237…

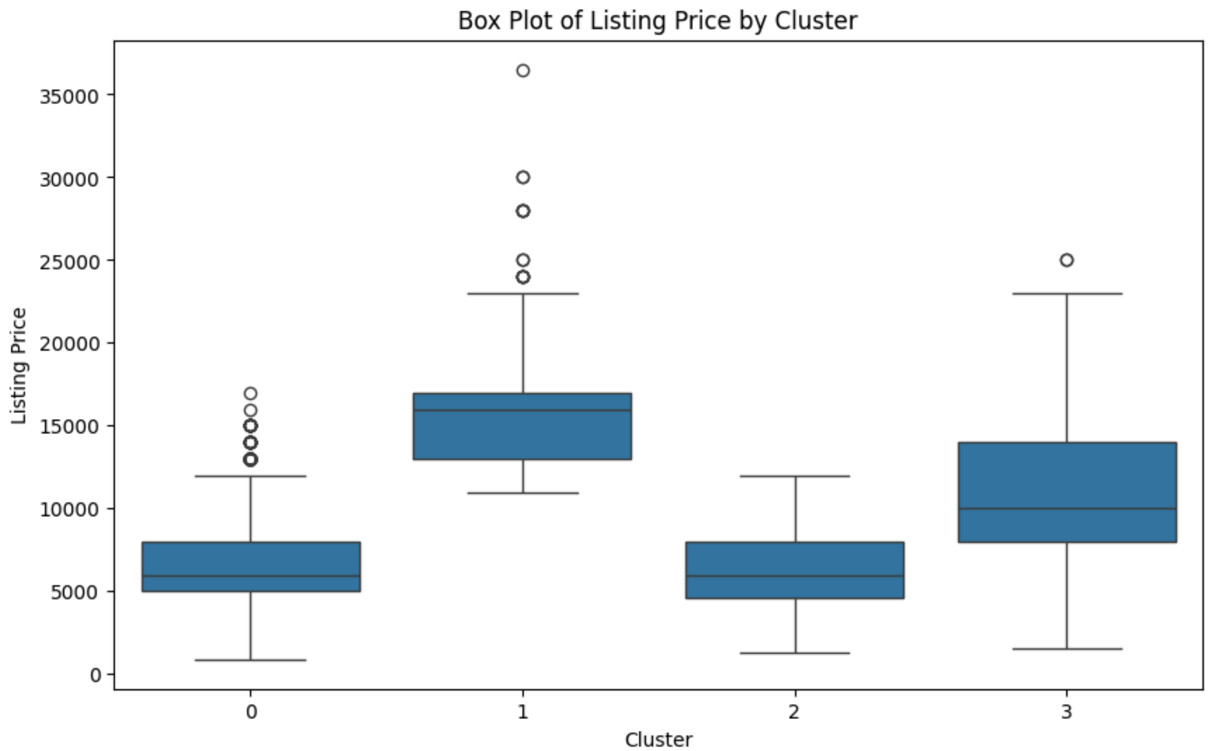
```
import matplotlib.pyplot as plt

# Bar plot for means
mean.T.plot(kind='bar', figsize=(12, 6), title="Cluster Means")
plt.ylabel("Mean Values")
plt.xlabel("Features")
plt.xticks(rotation=45)
plt.show()

# Bar plot for medians
median.T.plot(kind='bar', figsize=(12, 6), title="Cluster Medians", color="c
plt.ylabel("Median Values")
plt.xlabel("Features")
plt.xticks(rotation=45)
plt.show()
```

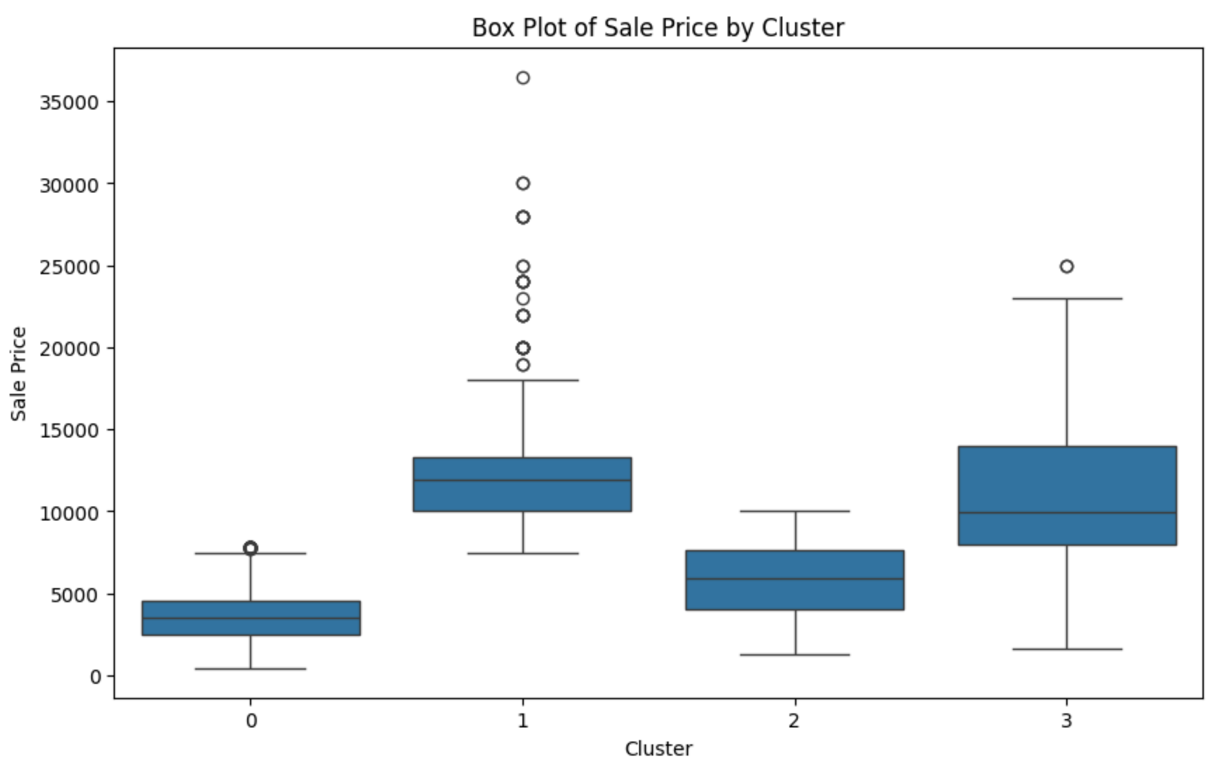


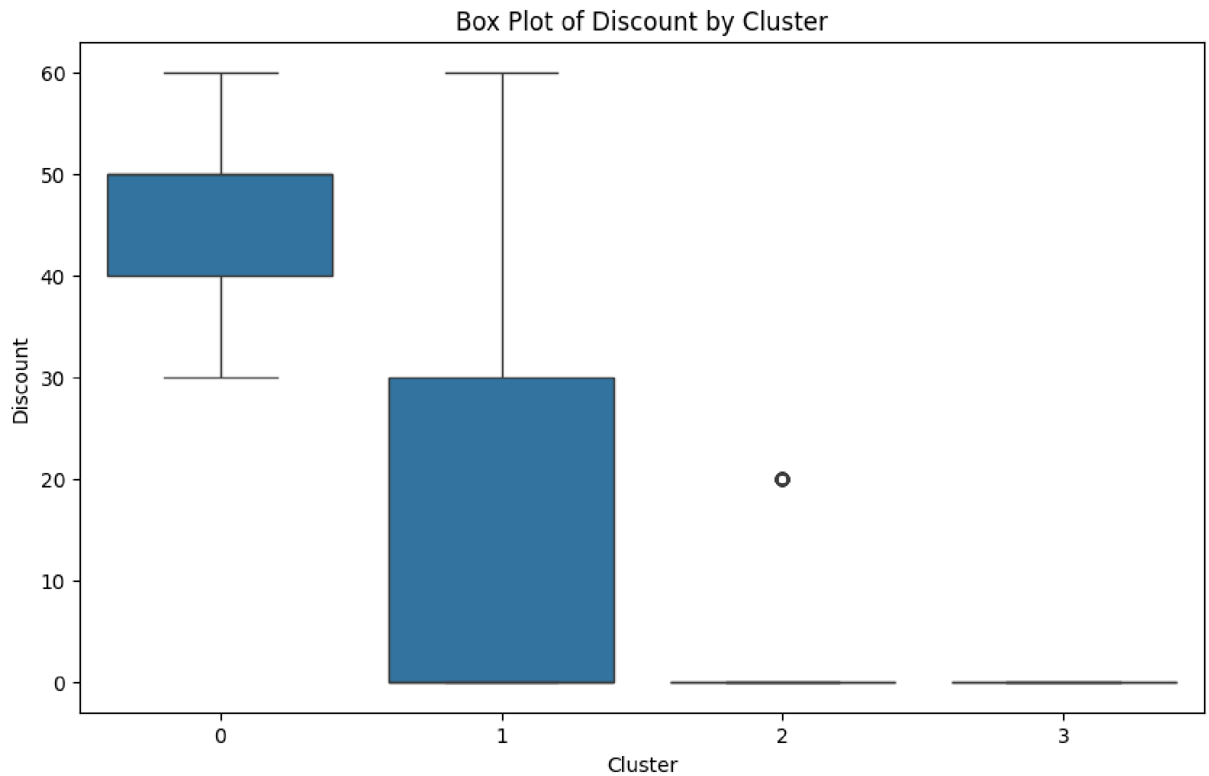


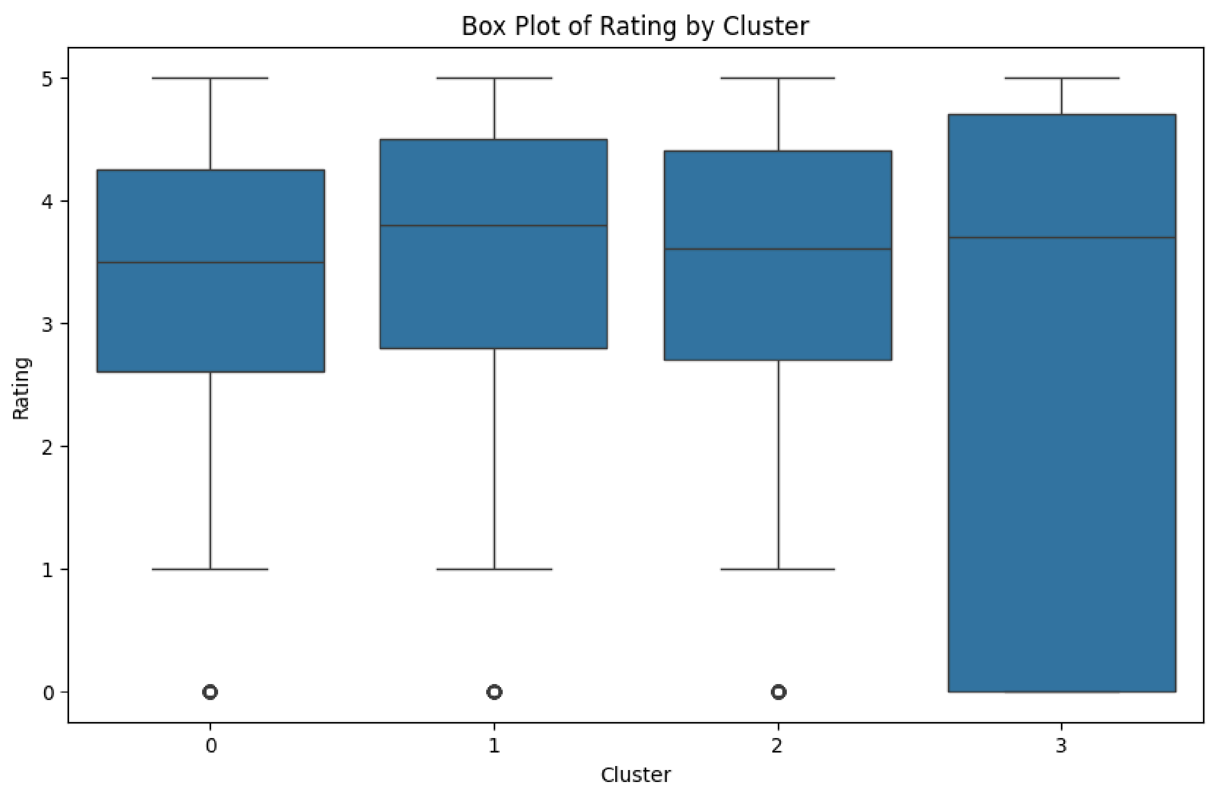
In [238…

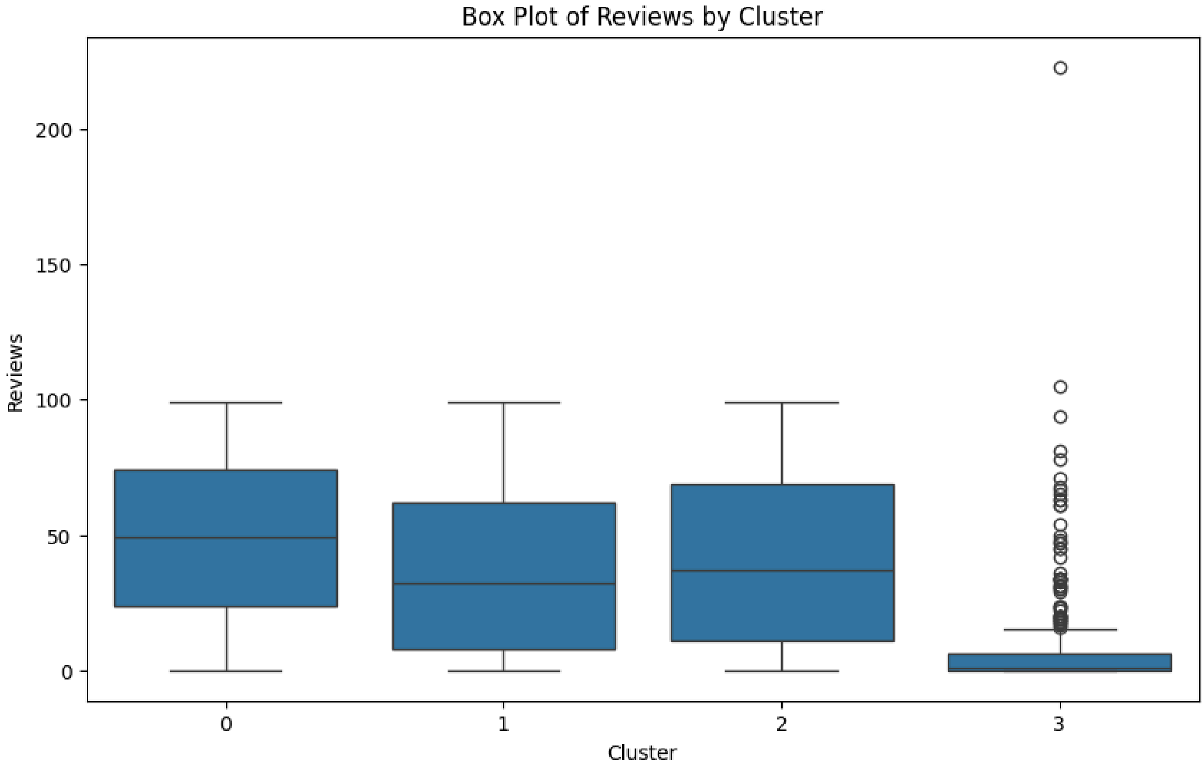
```
import seaborn as sns

# Box plot for numeric features by cluster
for feature in numeric_data.columns[:-1]:  # Exclude KMeans_Labels
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='KMeans_Labels', y=feature, data=numeric_data)
    plt.title(f"Box Plot of {feature} by Cluster")
    plt.xlabel("Cluster")
    plt.ylabel(feature)
    plt.show()
```


Box Plot of Listing Price by Cluster
Listing Price
0
5000
10000
15000
20000
25000
30000
35000
0
1
2
3
Cluster


Box Plot of Sale Price by Cluster
Sale Price
0
5000
10000
15000
20000
25000
30000
35000
0
1
2
3
Cluster

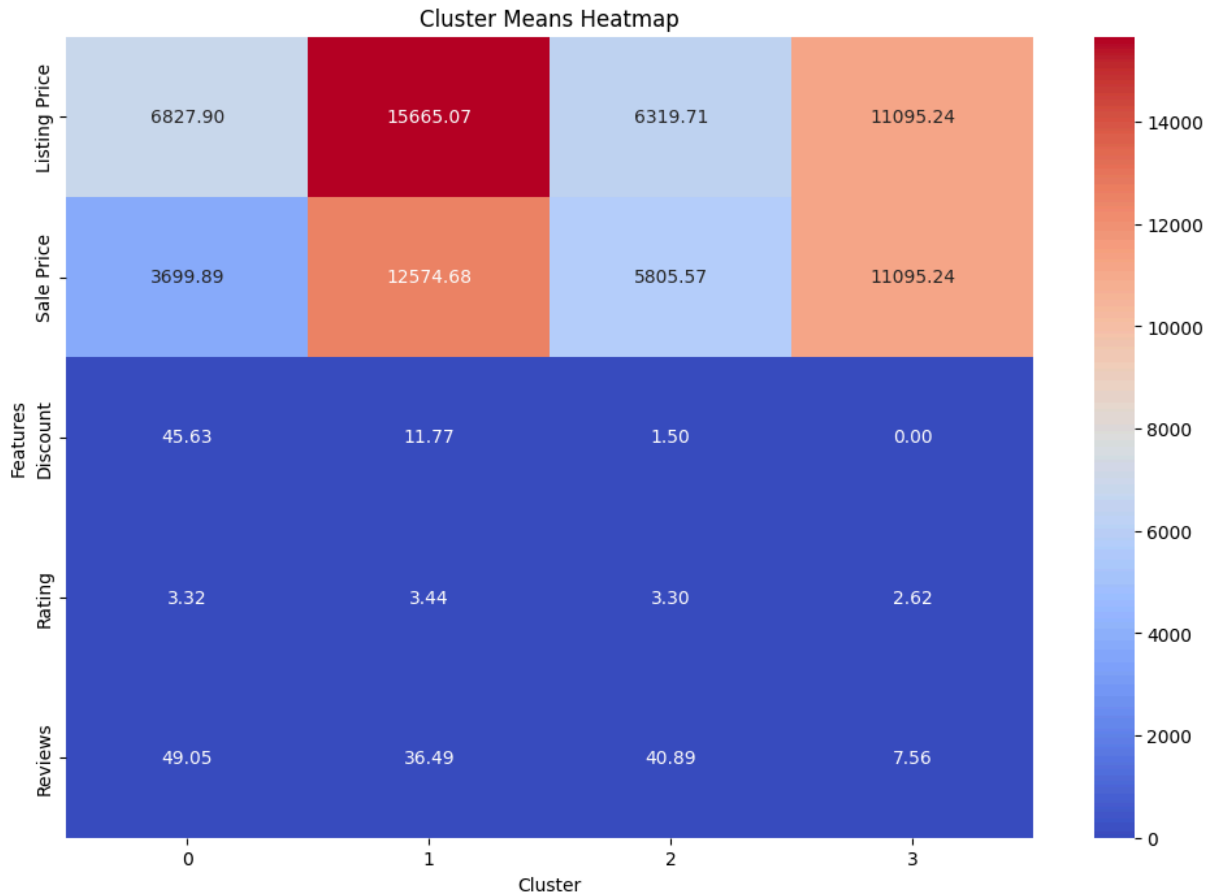

Box Plot of Discount by Cluster
60
50
40
30
20
10
0
Discount
0
1
2
3
Cluster


Box Plot of Rating by Cluster
5
4
3
2
1
0
Rating
0
1
2
3
Cluster



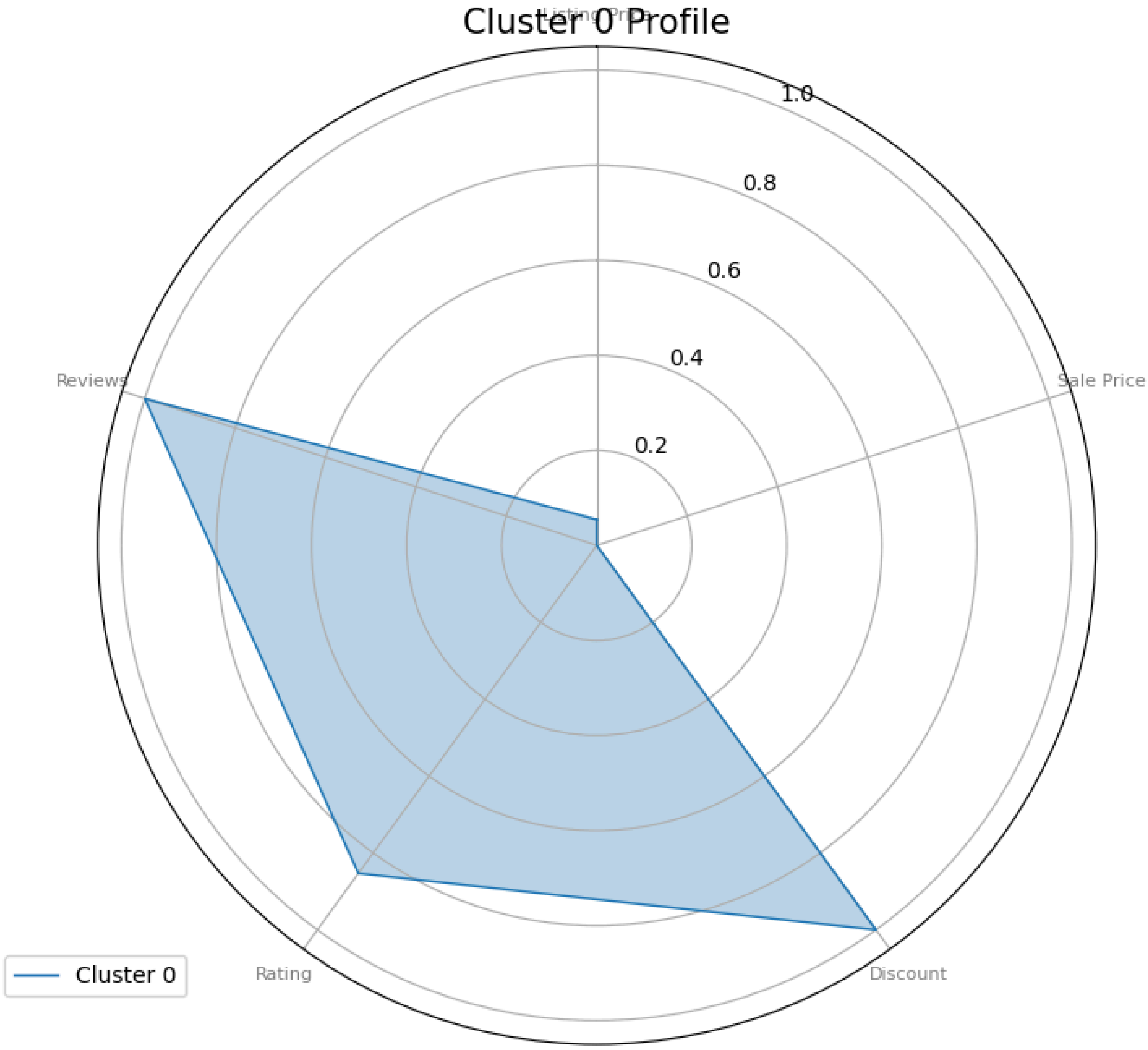
In [239…

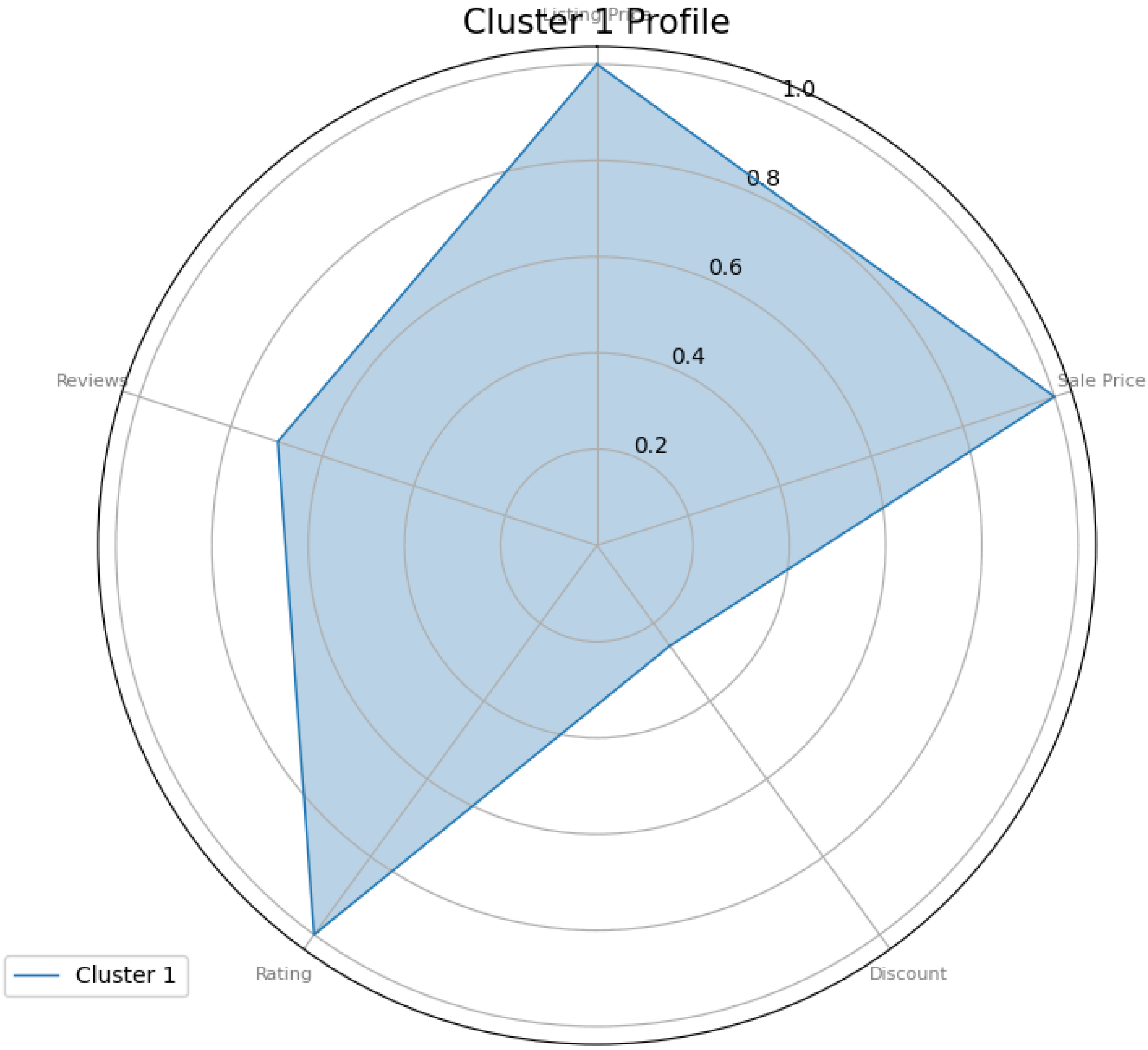
```
# Heatmap for means
plt.figure(figsize=(12, 8))
sns.heatmap(mean.T, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Cluster Means Heatmap")
plt.xlabel("Cluster")
plt.ylabel("Features")
plt.show()
```



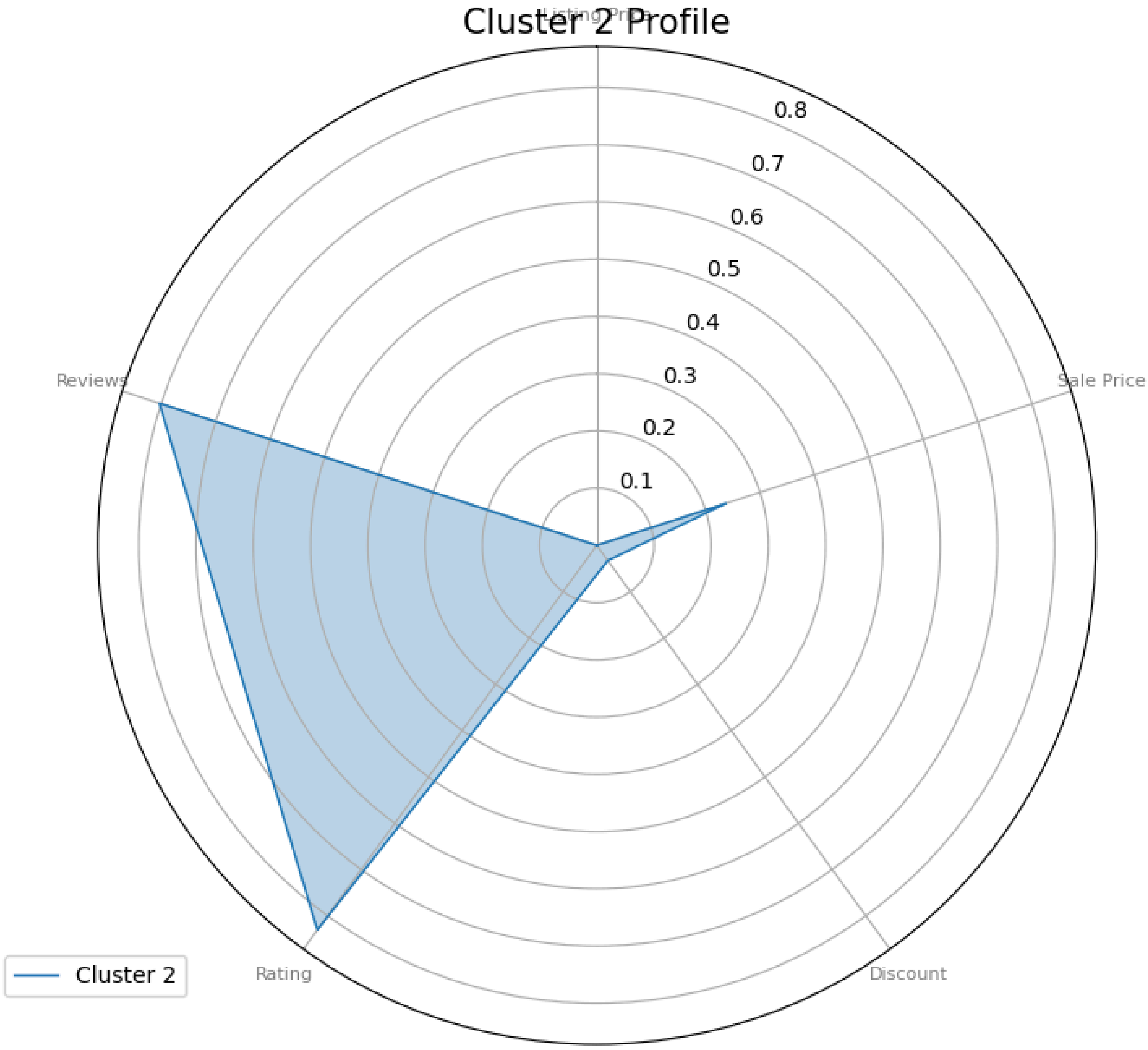
In [240…

```
from math import pi

# Normalize data
normalized_mean = (mean - mean.min()) / (mean.max() - mean.min())

# Radar chart
categories = normalized_mean.columns
for cluster_id in normalized_mean.index:
    values = normalized_mean.loc[cluster_id].values.flatten().tolist()
    values += values[:1]  # Close the loop

    angles = [n / float(len(categories)) * 2 * pi for n in range(len(categor
    angles += angles[:1]

    plt.figure(figsize=(8, 8))
    ax = plt.subplot(111, polar=True)
    ax.set_theta_offset(pi / 2)
    ax.set_theta_direction(-1)

    plt.xticks(angles[:-1], categories, color='grey', size=8)
    ax.plot(angles, values, linewidth=1, linestyle='solid', label=f'Cluster
    ax.fill(angles, values, alpha=0.3)
    plt.title(f"Cluster {cluster_id} Profile", size=15)
    plt.legend(loc='upper right', bbox_to_anchor=(0.1, 0.1))
    plt.show()
```


Cluster 0 Profile
Listing Price
Sale Price
Discount
Rating
Reviews
0.2
0.4
0.6
0.8
1.0
Cluster 0

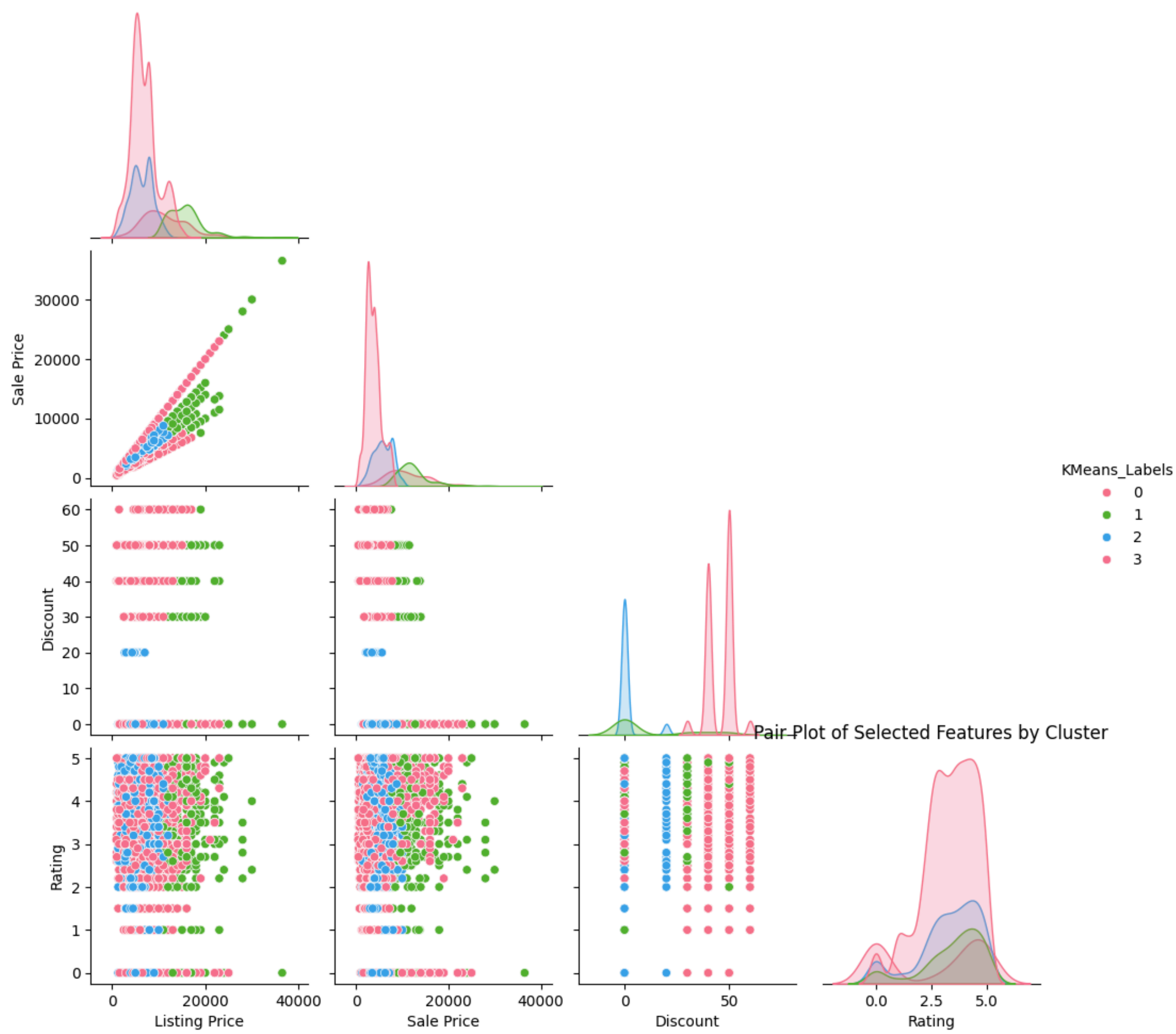

Cluster 1 Profile
Listing Price
Sale Price
Discount
Rating
Reviews
0.2
0.4
0.6
0.8
1.0
Cluster 1

Cluster 2 Profile



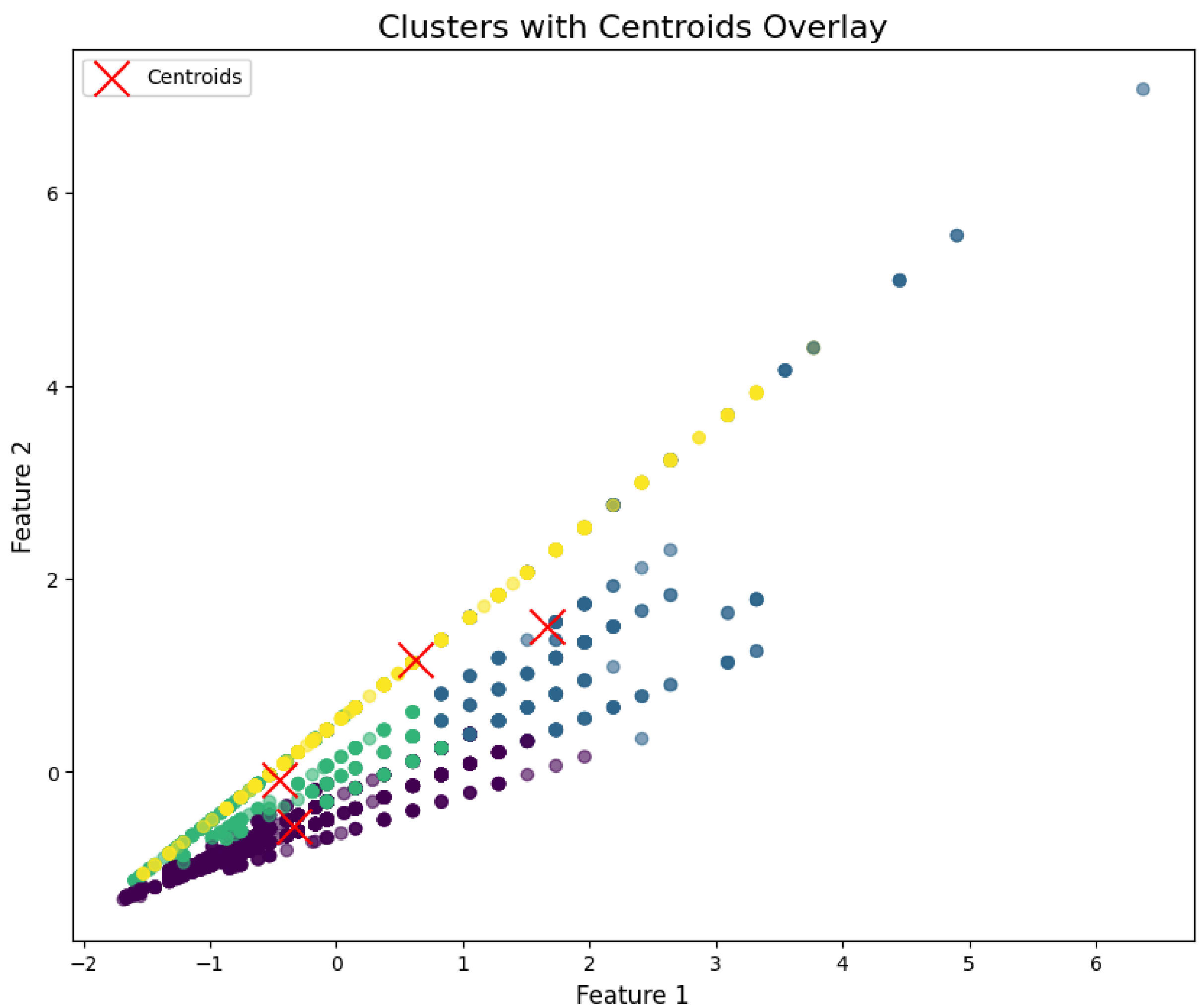
In [241…

```
# Pair plot for selected features
selected_features = numeric_data.columns[:4]  # Choose a few features for be
sns.pairplot(data=numeric_data, vars=selected_features, hue="KMeans_Labels",
plt.title("Pair Plot of Selected Features by Cluster")
plt.show()
```



In [242…

```
# Import necessary library
import numpy as np
import matplotlib.pyplot as plt

# Assuming `data_scaled` is already scaled and used for k-means clustering
plt.figure(figsize=(10, 8))

# Scatter plot for the clustered data
plt.scatter(data_scaled.iloc[:, 0], data_scaled.iloc[:, 1], c=kmeans.labels_

# Centroids calculated by k-means
centroids = kmeans.cluster_centers_

# Overlay centroids on the scatter plot
plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='red', marker='x', la

# Add plot title and labels
plt.title("Clusters with Centroids Overlay", fontsize=16)
plt.xlabel("Feature 1", fontsize=12)
plt.ylabel("Feature 2", fontsize=12)
plt.legend()
plt.show()
```



In [243…

```
print(data_scaled.head())
```

```
   Listing Price  Sale Price  Discount    Rating  Listing_Price_Zero
0       1.509415    0.317928  1.021839  1.090476           -0.387162
1      -0.165605   -0.544022  1.021839  0.040524           -0.387162
2      -1.659542   -1.289493  0.579948 -0.449453           -0.387162
3      -0.301417   -0.613910  1.021839  0.600498           -0.387162
4      -0.075063   -0.497431  1.021839  0.180518           -0.387162
```

In [244…

```
print(kmeans.labels_)
print(kmeans.cluster_centers_)
```

```
[1 0 0 ... 2 3 2]
[[-0.3401475  -0.56711036  0.82871063  0.05609221 -0.38716203]
 [ 1.66018233  1.50035514 -0.66734291  0.14057121 -0.36425669]
 [-0.45517739 -0.07657214 -1.12129795  0.04298987 -0.38716203]
 [ 0.62578217  1.15570554 -1.18761541 -0.43472539  2.58289789]]
```

Example: Linear Regression to Analyze the Relationship Between Discount and Reviews

In [245…

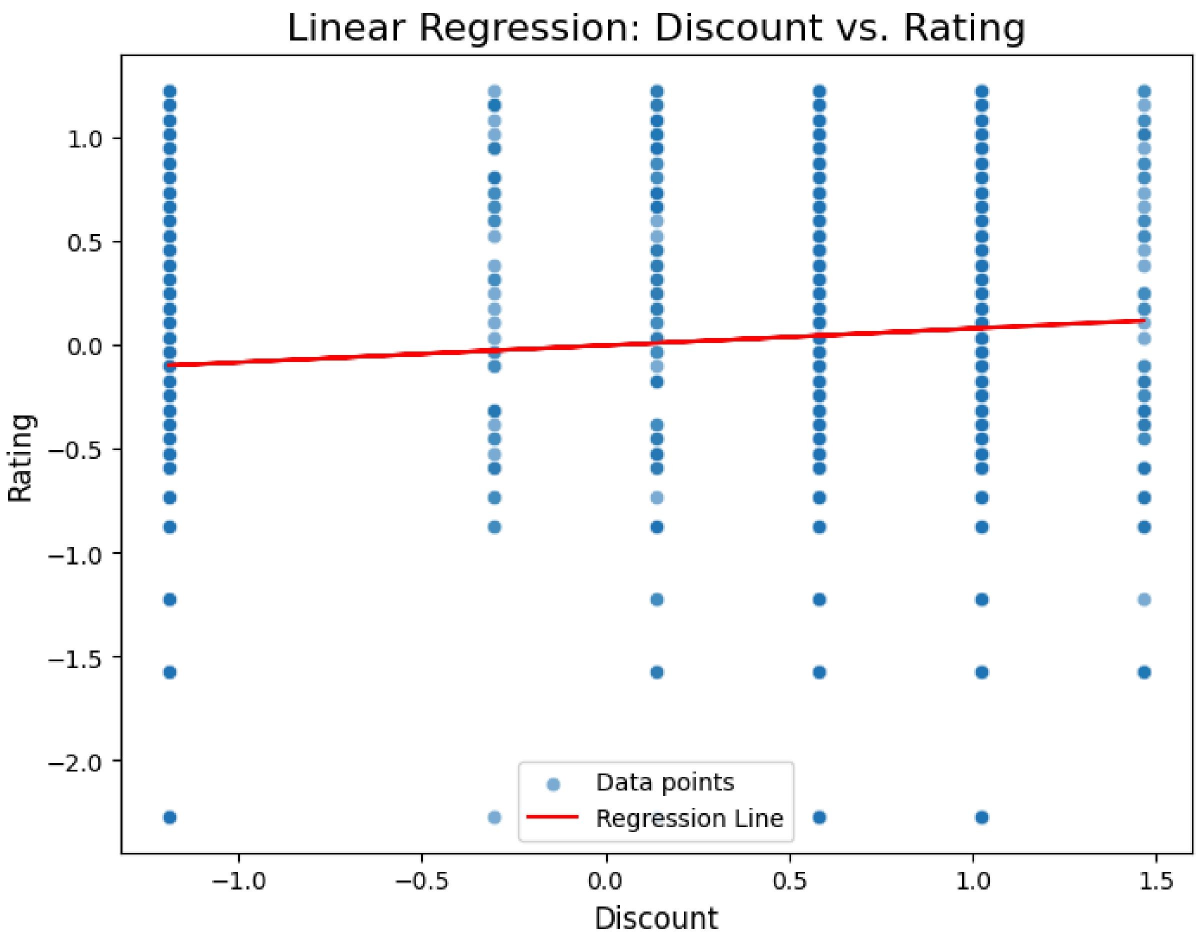
```
print(data_scaled.columns)
```

```
Index(['Listing Price', 'Sale Price', 'Discount', 'Rating',
       'Listing_Price_Zero'],
      dtype='object')
```

In [246…

```
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns

# Define the dependent and independent variables
X = data_scaled['Discount']  # Independent variable
y = data_scaled['Rating']    # Dependent variable

# Add constant to independent variable
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()
print(model.summary())

# Plot the regression line
plt.figure(figsize=(8, 6))
sns.scatterplot(x=data_scaled['Discount'], y=data_scaled['Rating'], alpha=0.
plt.plot(data_scaled['Discount'], model.predict(X), color='red', label='Regr
plt.title('Linear Regression: Discount vs. Rating', fontsize=16)
plt.xlabel('Discount', fontsize=12)
plt.ylabel('Rating', fontsize=12)
plt.legend()
plt.show()
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                 Rating   R-squared:                       0.0
07
Model:                            OLS   Adj. R-squared:                  0.0
06
Method:                 Least Squares   F-statistic:                     21.
84
Date:                Sat, 04 Jan 2025   Prob (F-statistic):           3.08e-
06
Time:                        13:51:39   Log-Likelihood:                -462
6.2
No. Observations:                3268   AIC:                             925
6.
Df Residuals:                    3266   BIC:                             926
9.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const      -1.978e-16      0.017  -1.13e-14      1.000      -0.034       0.0
34
Discount       0.0815      0.017      4.674      0.000       0.047       0.1
16
==============================================================================
==
Omnibus:                      299.798   Durbin-Watson:                   1.9
52
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              388.3
02
Skew:                          -0.843   Prob(JB):                     4.80e-
85
Kurtosis:                       2.908   Cond. No.                         1.
00
==============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```



In [247…

```
data.head()
```

Out [247...

	Product Name	Product ID	Listing Price	Sale Price	Discount	Brand	Rating	Reviews	Listing
0	Women's adidas Originals NMD_Racer Primeknit S...	AH2430	14999	7499	50	Adidas ORIGINALS	4.8	41	
1	Women's adidas Originals Sleek Shoes	G27341	7599	3799	50	Adidas ORIGINALS	3.3	24	
2	Women's adidas Swim Puka Slippers	CM0081	999	599	40	Adidas CORE / NEO	2.6	37	
3	Women's adidas Sport Inspired Questar Ride Shoes	B44832	6999	3499	50	Adidas CORE / NEO	4.1	35	
4	Women's adidas Originals Taekwondo Shoes	D98205	7999	3999	50	Adidas ORIGINALS	3.5	72	

In [248...

```
from sklearn.preprocessing import StandardScaler

# Select numerical columns for scaling
numerical_columns = ['Listing Price', 'Sale Price', 'Discount', 'Rating', 'R
data_new = data[numerical_columns]  # Select relevant columns for scaling

# Initialize scaler and fit-transform
scaler = StandardScaler()
data_scaled = pd.DataFrame(scaler.fit_transform(data_new), columns=numerical

# Include scaled data in the analysis
data_scaled['KMeans_Labels'] = data['KMeans_Labels']
data_scaled['Listing_Price_Zero'] = data['Listing_Price_Zero']
data_scaled.head()
```

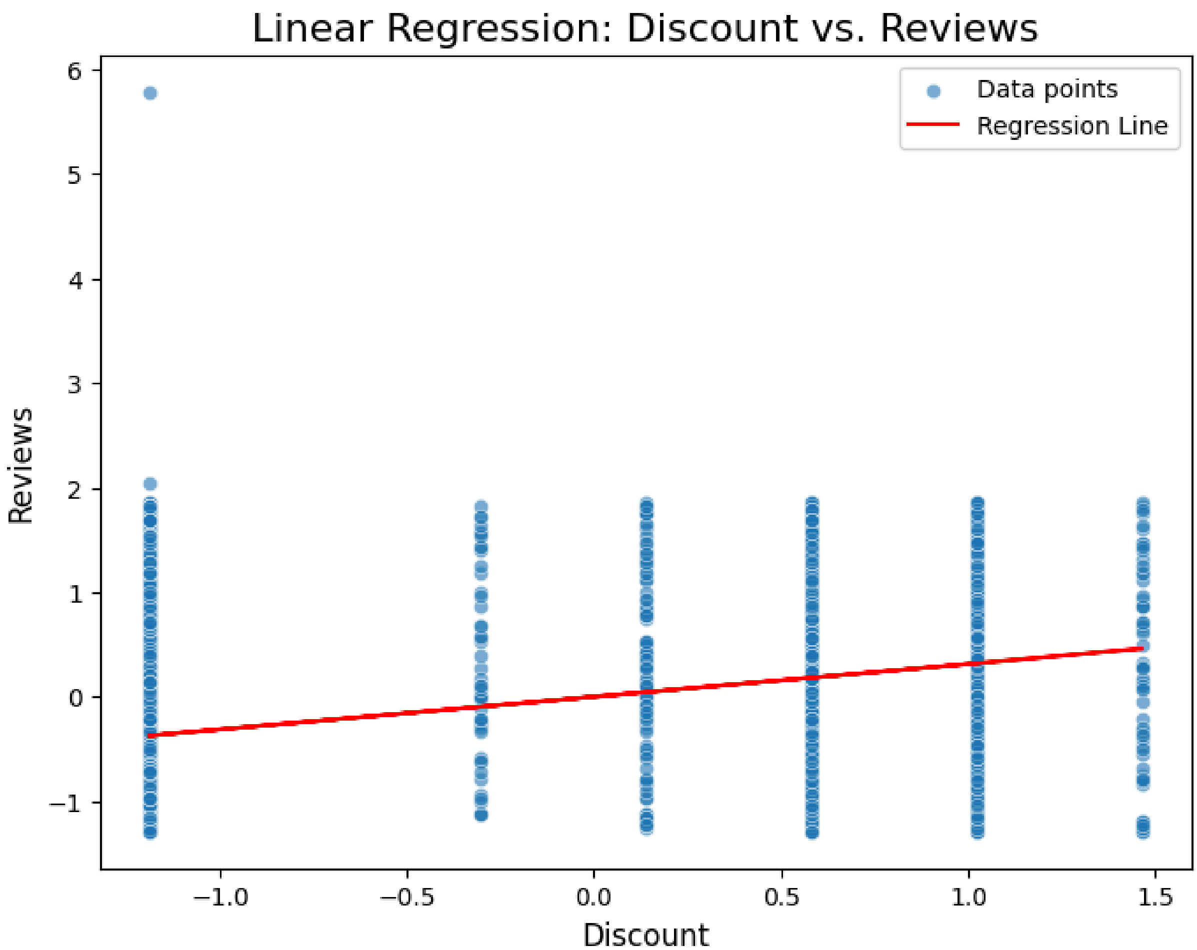
Out [248...

	Listing Price	Sale Price	Discount	Rating	Reviews	KMeans_Labels	Listing_Pri
0	1.509415	0.317928	1.021839	1.090476	0.014214	1	
1	-0.165605	-0.544022	1.021839	0.040524	-0.524807	0	
2	-1.659542	-1.289493	0.579948	-0.449453	-0.112615	0	
3	-0.301417	-0.613910	1.021839	0.600498	-0.176029	0	
4	-0.075063	-0.497431	1.021839	0.180518	0.997134	0	

In [249...

```
# Define the dependent and independent variables
X = data_scaled['Discount']  # Independent variable
y = data_scaled['Reviews']   # Dependent variable

# Add constant to independent variable
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()
print(model.summary())

# Plot the regression line
plt.figure(figsize=(8, 6))
sns.scatterplot(x=data_scaled['Discount'], y=data_scaled['Reviews'], alpha=0
plt.plot(data_scaled['Discount'], model.predict(X), color='red', label='Regr
plt.title('Linear Regression: Discount vs. Reviews', fontsize=16)
plt.xlabel('Discount', fontsize=12)
plt.ylabel('Reviews', fontsize=12)
plt.legend()
plt.show()
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                Reviews   R-squared:                       0.0
98
Model:                            OLS   Adj. R-squared:                  0.0
98
Method:                 Least Squares   F-statistic:                     35
6.2
Date:                Sat, 04 Jan 2025   Prob (F-statistic):           1.71e-
75
Time:                        13:51:39   Log-Likelihood:                 -446
7.9
No. Observations:                3268   AIC:                             894
0.
Df Residuals:                    3266   BIC:                             895
2.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const       3.747e-16      0.017   2.26e-14      1.000      -0.033       0.0
33
Discount       0.3136      0.017     18.873      0.000       0.281       0.3
46
==============================================================================
==
Omnibus:                      147.006   Durbin-Watson:                   1.7
13
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              137.8
54
Skew:                           0.451   Prob(JB):                     1.16e-
30
Kurtosis:                       2.553   Cond. No.                         1.
00
==============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```



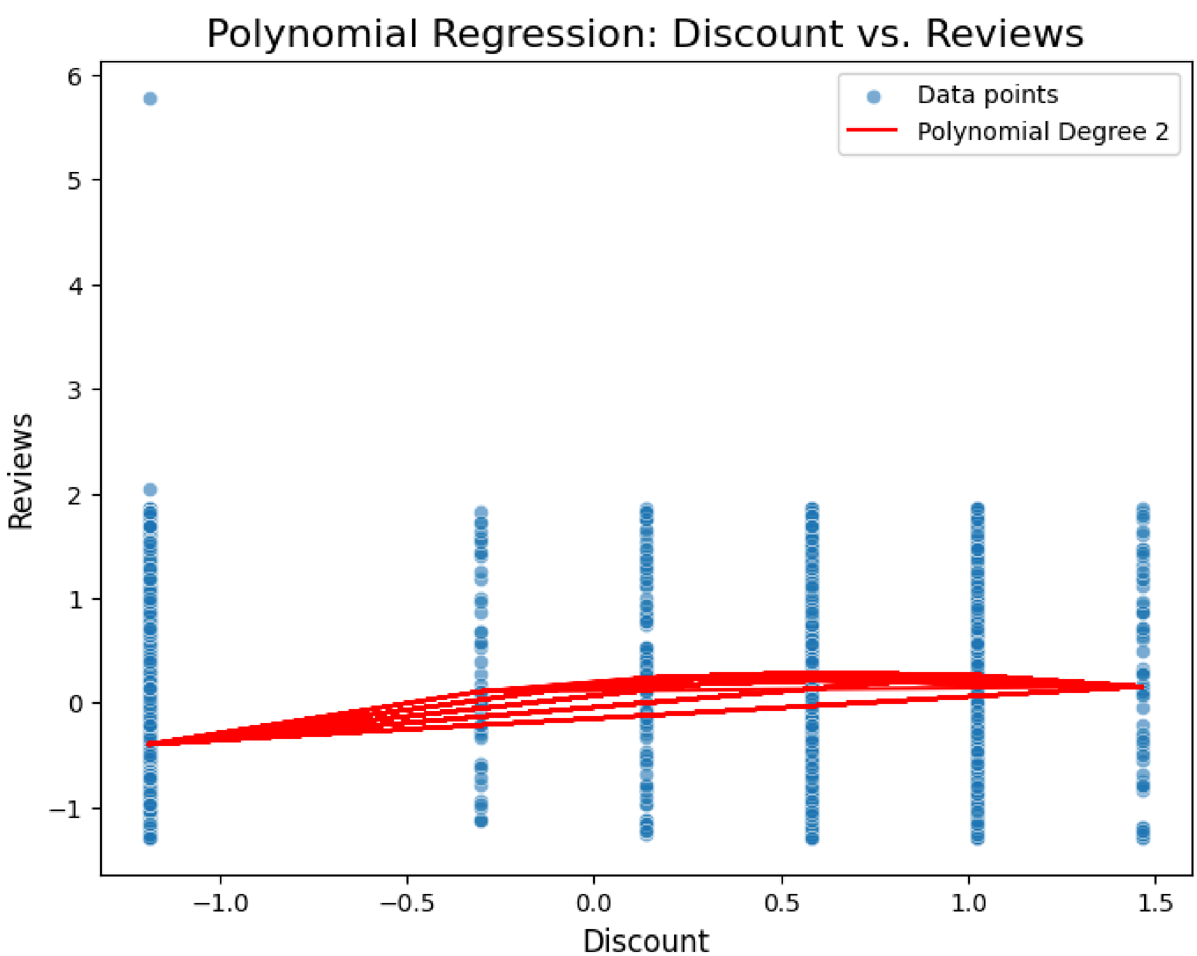
Observations to Note Check the R-squared value and p-values in the regression summary to evaluate the significance of the relationship. The scatter plot will provide a visual understanding of how Discount impacts Reviews.

Polynomial regression can provide a better fit if the relationship between the dependent and independent variables is non-linear. Here's how you can implement polynomial regression for the relationship between Discount and Reviews.

In [250…

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Define the independent and dependent variables
X = data_scaled[['Discount']].values  # Independent variable
y = data_scaled['Reviews'].values    # Dependent variable

# Apply Polynomial Transformation
```

```
degree = 2  # You can adjust the degree for better fit
poly = PolynomialFeatures(degree=degree)
X_poly = poly.fit_transform(X)

# Fit the Polynomial Regression Model
model = LinearRegression()
model.fit(X_poly, y)

# Generate predictions
y_pred = model.predict(X_poly)

# Visualize the Polynomial Regression Fit
plt.figure(figsize=(8, 6))
sns.scatterplot(x=data_scaled['Discount'], y=data_scaled['Reviews'], alpha=0
plt.plot(data_scaled['Discount'], y_pred, color='red', label=f'Polynomial De
plt.title('Polynomial Regression: Discount vs. Reviews', fontsize=16)
plt.xlabel('Discount', fontsize=12)
plt.ylabel('Reviews', fontsize=12)
plt.legend()
plt.show()

# Evaluate Model Performance
from sklearn.metrics import r2_score, mean_squared_error
r2 = r2_score(y, y_pred)
mse = mean_squared_error(y, y_pred)
print(f'R-squared: {r2:.4f}')
print(f'Mean Squared Error: {mse:.4f}')
```



```
R-squared: 0.1059
Mean Squared Error: 0.8941
```

Key Components of the Code PolynomialFeatures:

Transforms the independent variable (X) into polynomial features of the specified degree. Adds higher-order terms (e.g., x 2 , x 3 x 2 ,x 3) to the regression model. Model Fitting:

A LinearRegression model is fitted using the polynomial-transformed X_poly. Visualization:

The scatter plot shows the original data points. The red curve represents the polynomial regression fit. Model Evaluation:

R-squared: Indicates the proportion of variance explained by the model. Mean Squared Error (MSE): Measures the average squared difference between observed and predicted values. Adjustments: If the fit isn't adequate, increase the degree of the polynomial to capture more complexity. Ensure that Reviews and Discount do not contain missing or non-numeric values.

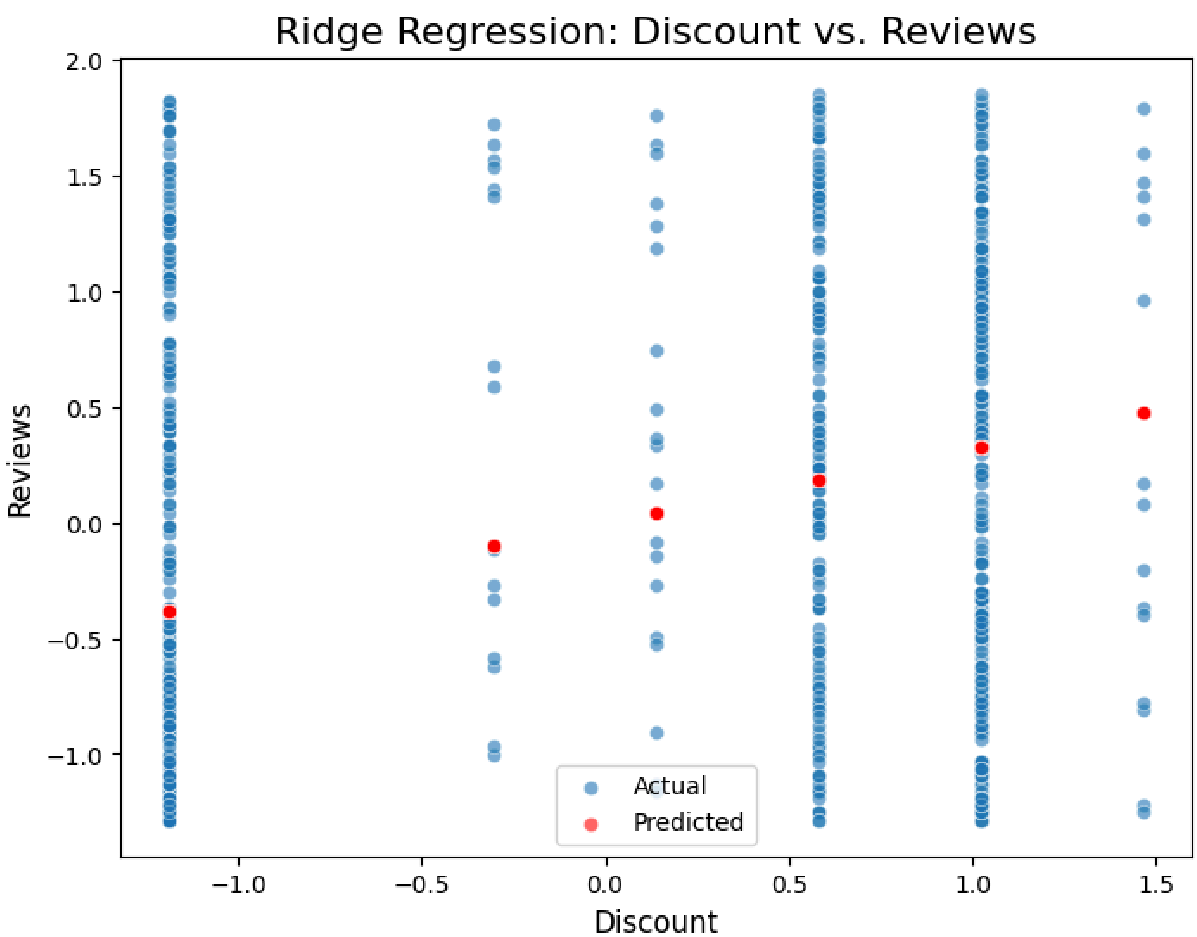
Ridge Regression, a type of linear regression that includes regularization to prevent overfitting. Ridge regression penalizes large coefficients, making it more robust in datasets with multicollinearity or noise.

Code for Ridge Regression

In [251…

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error

# Define independent (X) and dependent (y) variables
X = data_scaled[['Discount']].values  # Independent variable
y = data_scaled['Reviews'].values    # Dependent variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Create and fit Ridge Regression model
ridge_model = Ridge(alpha=1.0)  # Alpha is the regularization strength; high
ridge_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = ridge_model.predict(X_test)

# Visualize the predictions
plt.figure(figsize=(8, 6))
```

```python
sns.scatterplot(x=X_test.flatten(), y=y_test, alpha=0.6, label='Actual')
sns.scatterplot(x=X_test.flatten(), y=y_pred, alpha=0.6, label='Predicted',
plt.title('Ridge Regression: Discount vs. Reviews', fontsize=16)
plt.xlabel('Discount', fontsize=12)
plt.ylabel('Reviews', fontsize=12)
plt.legend()
plt.show()

# Evaluate the model
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print(f'R-squared: {r2:.4f}')
print(f'Mean Squared Error: {mse:.4f}')
```




```
R-squared: 0.0711
Mean Squared Error: 0.9368
```

from sklearn.linear_model import Ridge from sklearn.model_selection import train_test_split from sklearn.metrics import r2_score, mean_squared_error

Define independent (X) and dependent (y) variables

X = data_scaled[['Discount']].values # Independent variable y = data_scaled['Reviews'].values # Dependent variable

Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Create and fit Ridge Regression model

ridge_model = Ridge(alpha=1.0) # Alpha is the regularization strength; higher values increase regularization ridge_model.fit(X_train, y_train)

Make predictions on the test set

y_pred = ridge_model.predict(X_test)

Visualize the predictions

plt.figure(figsize=(8, 6)) sns.scatterplot(x=X_test.flatten(), y=y_test, alpha=0.6, label='Actual') sns.scatterplot(x=X_test.flatten(), y=y_pred, alpha=0.6, label='Predicted', color='red') plt.title('Ridge Regression: Discount vs. Reviews', fontsize=16) plt.xlabel('Discount', fontsize=12) plt.ylabel('Reviews', fontsize=12) plt.legend() plt.show()

Evaluate the model

r2 = r2_score(y_test, y_pred) mse = mean_squared_error(y_test, y_pred) print(f'R-squared: {r2:.4f}') print(f'Mean Squared Error: {mse:.4f}')

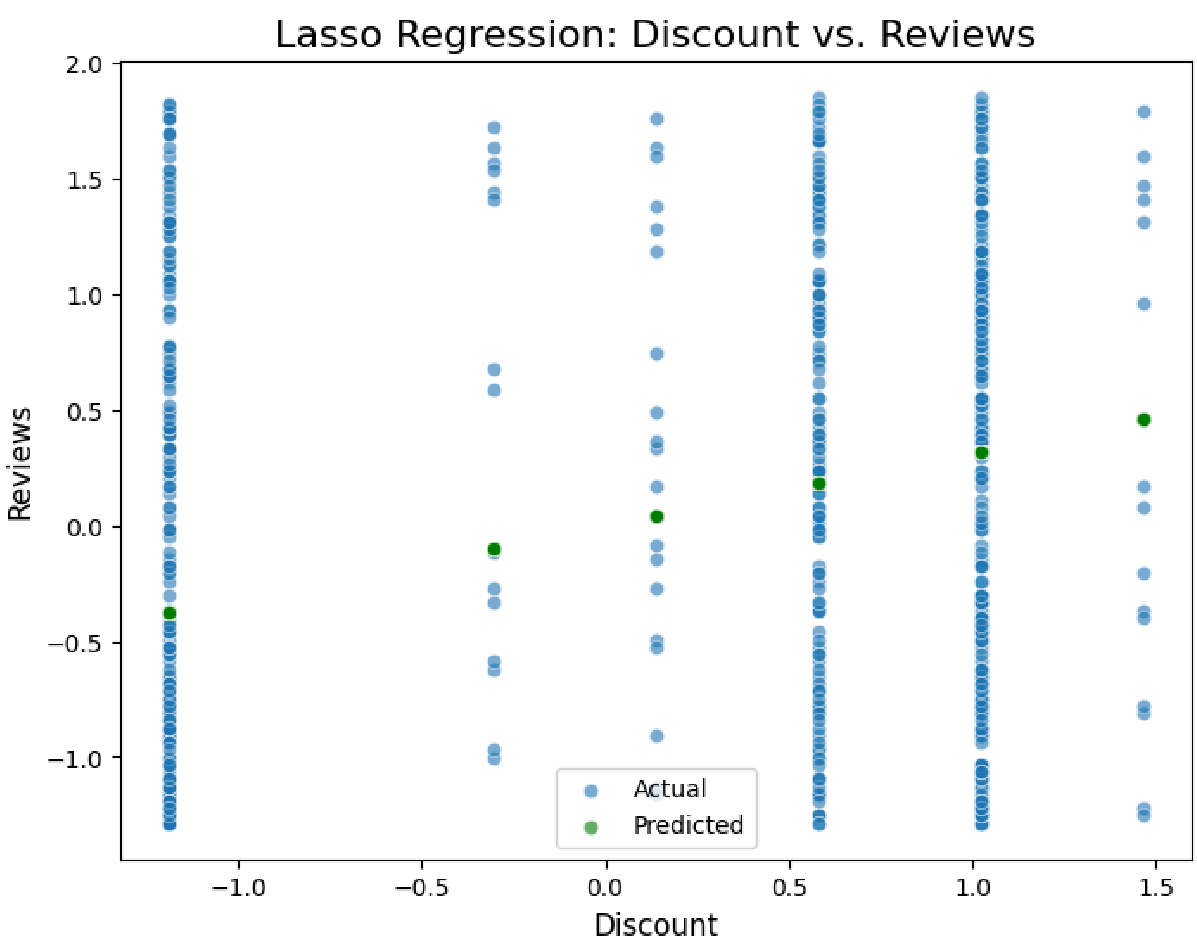
Advantages Ridge regression is particularly useful for datasets with correlated features. It avoids overfitting in situations where regular linear regression might struggle.

Lasso Regression Lasso adds an L1 penalty, which can shrink coefficients to exactly zero, leading to feature selection.

Code for Lasso Regression:

In [252…

```python
from sklearn.linear_model import Lasso

# Define and fit the Lasso model
lasso_model = Lasso(alpha=0.01)  # Adjust alpha as needed
lasso_model.fit(X_train, y_train)

# Make predictions
y_pred_lasso = lasso_model.predict(X_test)

# Visualize predictions
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_test.flatten(), y=y_test, alpha=0.6, label='Actual')
sns.scatterplot(x=X_test.flatten(), y=y_pred_lasso, alpha=0.6, label='Predic
plt.title('Lasso Regression: Discount vs. Reviews', fontsize=16)
plt.xlabel('Discount', fontsize=12)
plt.ylabel('Reviews', fontsize=12)
plt.legend()
plt.show()

# Evaluate the model
r2_lasso = r2_score(y_test, y_pred_lasso)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
print(f'Lasso Regression - R-squared: {r2_lasso:.4f}')
print(f'Lasso Regression - Mean Squared Error: {mse_lasso:.4f}')
```



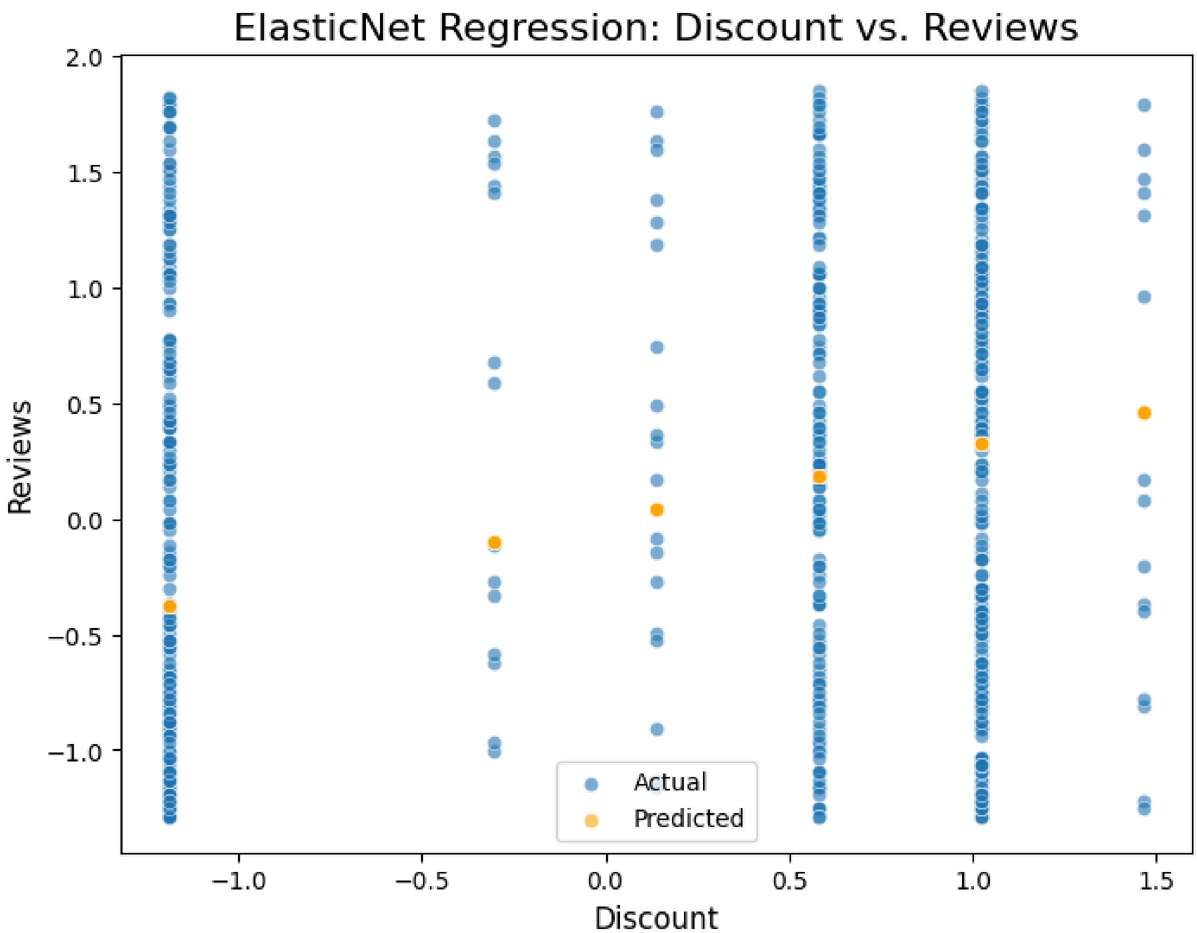

```
Lasso Regression - R-squared: 0.0720
Lasso Regression - Mean Squared Error: 0.9359
```

ElasticNet Regression ElasticNet combines L1 and L2 penalties. It can be tuned to balance between Lasso and Ridge characteristics.

Code for ElasticNet Regression:

In [253…

```
from sklearn.linear_model import ElasticNet

# Define and fit the ElasticNet model
elasticnet_model = ElasticNet(alpha=0.01, l1_ratio=0.5)  # l1_ratio=0.5 bala
elasticnet_model.fit(X_train, y_train)

# Make predictions
y_pred_elastic = elasticnet_model.predict(X_test)

# Visualize predictions
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_test.flatten(), y=y_test, alpha=0.6, label='Actual')
sns.scatterplot(x=X_test.flatten(), y=y_pred_elastic, alpha=0.6, label='Pred
plt.title('ElasticNet Regression: Discount vs. Reviews', fontsize=16)
plt.xlabel('Discount', fontsize=12)
plt.ylabel('Reviews', fontsize=12)
plt.legend()
plt.show()

# Evaluate the model
r2_elastic = r2_score(y_test, y_pred_elastic)
mse_elastic = mean_squared_error(y_test, y_pred_elastic)
print(f'ElasticNet Regression - R-squared: {r2_elastic:.4f}')
print(f'ElasticNet Regression - Mean Squared Error: {mse_elastic:.4f}')
```




```
ElasticNet Regression - R-squared: 0.0717
ElasticNet Regression - Mean Squared Error: 0.9362
```

Comparison Summary Lasso: Useful when you suspect that only a few features are significant. ElasticNet: Ideal when there's multicollinearity and you want a balance between Lasso's feature selection and Ridge's robustness.

Residuals are the differences between the actual and predicted values. Visualizing residuals helps evaluate the assumptions of a regression model and detect patterns indicating poor fit or model bias.

Common Residual Plots Residuals vs. Fitted Values:

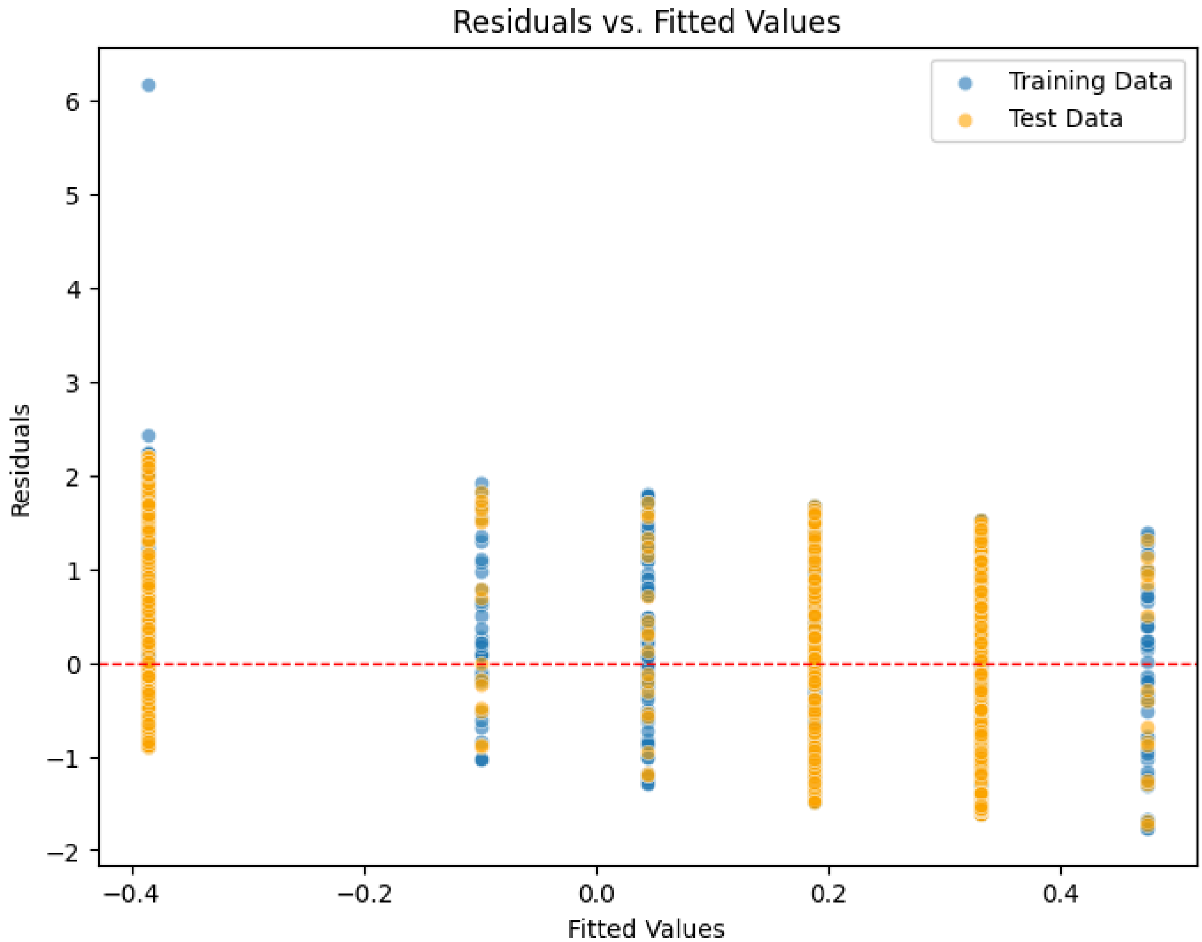
Detects non-linearity and unequal variance (heteroscedasticity). Residuals should be randomly scattered around zero. Histogram of Residuals:

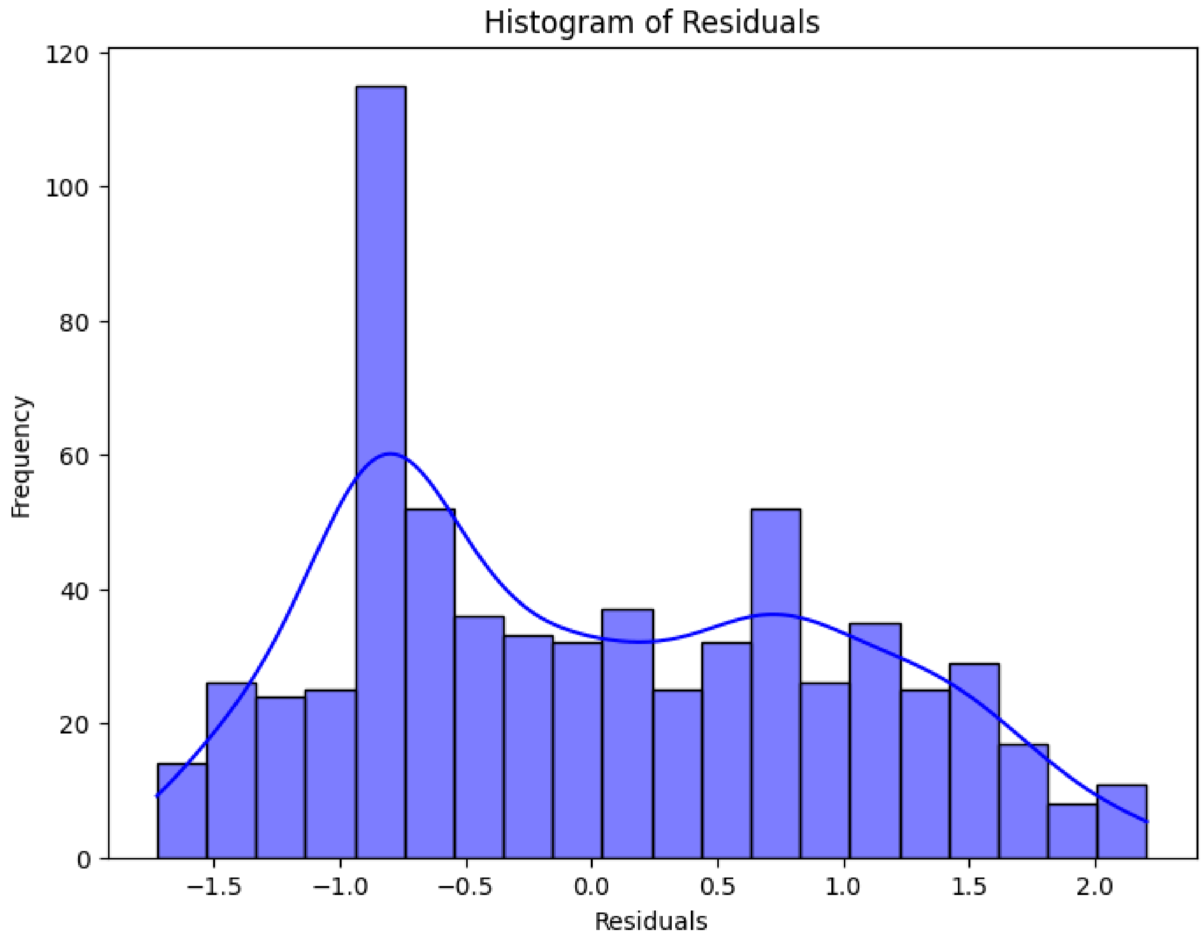
Assesses the normality of residuals. Residuals should ideally follow a normal distribution. Q-Q Plot:

Compares the distribution of residuals to a theoretical normal distribution. Residuals Over Time:

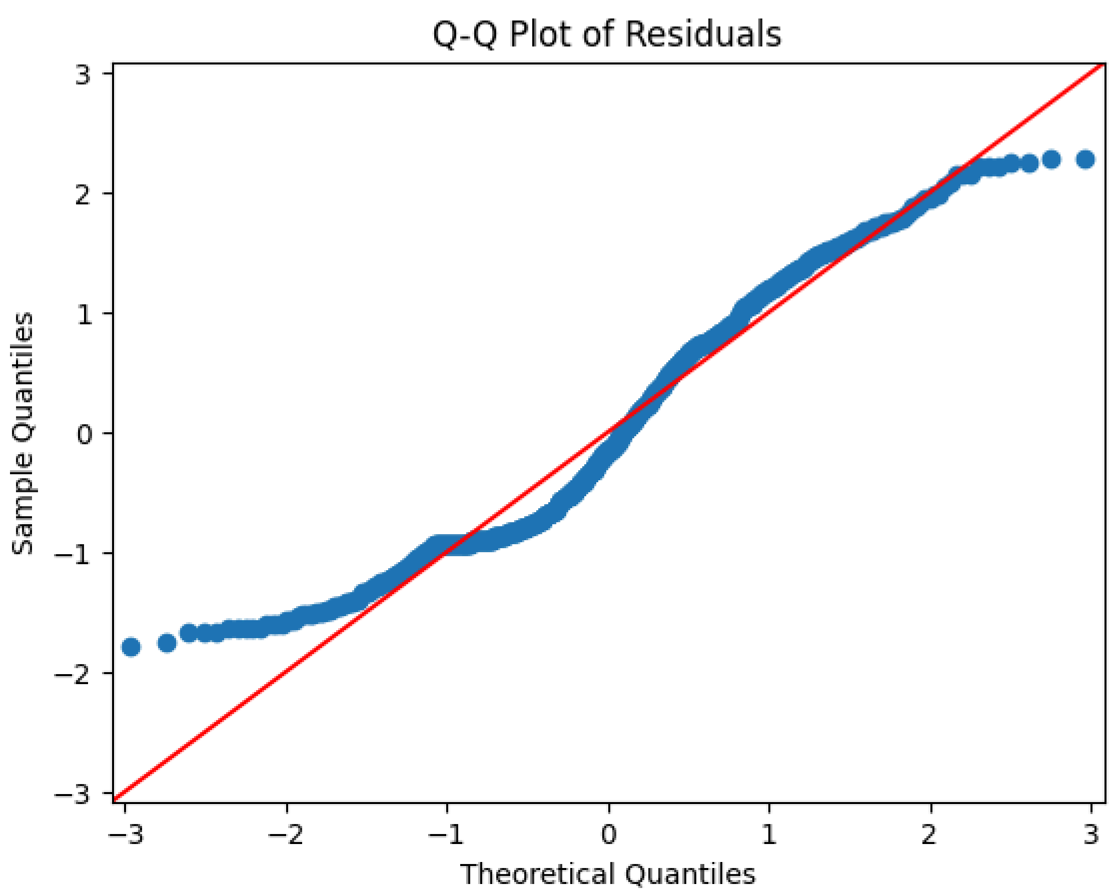
For time-series data, checks autocorrelation or patterns in residuals. Code for Visualizing Residuals

In [254…
```
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

# Predict using the model (use the Ridge, Lasso, or ElasticNet model)
y_train_pred = ridge_model.predict(X_train)
y_test_pred = ridge_model.predict(X_test)

# Calculate residuals
residuals_train = y_train - y_train_pred
residuals_test = y_test - y_test_pred

# Residuals vs Fitted Values
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_train_pred, y=residuals_train, label='Training Data', al
sns.scatterplot(x=y_test_pred, y=residuals_test, label='Test Data', color='c
plt.axhline(0, color='red', linestyle='--', linewidth=1)
plt.title('Residuals vs. Fitted Values')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.legend()
plt.show()

# Histogram of Residuals
plt.figure(figsize=(8, 6))
sns.histplot(residuals_test, kde=True, bins=20, color='blue')
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.show()

# Q-Q Plot of Residuals
plt.figure(figsize=(8, 6))
sm.qqplot(residuals_test, line='45', fit=True)
plt.title('Q-Q Plot of Residuals')
plt.show()
```


Residuals vs. Fitted Values
Training Data
Test Data
Residuals
Fitted Values
6
5
4
3
2
1
0
−1
−2
−0.4
−0.2
0.0
0.2
0.4



<Figure size 800x600 with 0 Axes>



How to Interpret the Visuals Residuals vs. Fitted Values:

Random scatter around zero indicates a good fit. Patterns (e.g., curves, funnels) suggest non-linearity or heteroscedasticity. Histogram of Residuals:

Bell-shaped histogram suggests normality. Skewed distribution indicates potential issues with the model. Q-Q Plot:

Points lying close to the diagonal line suggest residuals follow a normal distribution. Deviations at the tails indicate heavy-tailed or skewed residuals.

Key Observations Clustering (K-Means):

Clusters were created to segment products based on numeric features like pricing, discounts, and ratings. Visualizations showed clear separations in clusters and helped identify group characteristics. Regression Analysis:

Linear, Ridge, Lasso, and Polynomial regressions were explored to understand relationships between features like discounts and reviews. Polynomial regression provided better fit for non-linear relationships. Evaluation metrics like RMSE, R-squared, and MAE were calculated to assess model performance. Visualization Insights:

Scatter plots, residual plots, and correlation matrices revealed relationships and potential multicollinearity in features. Residual plots helped assess the goodness of fit for regression models. Histograms and Q-Q plots evaluated the normality of residuals. Data Cleaning:

Addressed errors caused by non-numeric data and missing columns. Proper scaling and handling of categorical data were critical for clustering and regression models. Model Evaluation:

Ridge and Lasso regression controlled overfitting, with Lasso being particularly useful for feature selection. Polynomial regression captured complex relationships but required careful tuning to avoid overfitting. Residual analysis highlighted areas where models could improve. Practical Coding Skills:

Learned to structure data preprocessing, scaling, and model training in steps. Developed reusable code for visualizations and statistical evaluations. Identified the importance of verifying assumptions before drawing conclusions.

Here’s a plain summary of the observations and key learning points from today’s exercises and analyses:

Key Observations Clustering (K-Means):

Clusters were created to segment products based on numeric features like pricing, discounts, and ratings. Visualizations showed clear separations in clusters and helped identify group characteristics. Regression Analysis:

Linear, Ridge, Lasso, and Polynomial regressions were explored to understand relationships between features like discounts and reviews. Polynomial regression provided better fit for non-linear relationships. Evaluation metrics like RMSE, R-squared, and MAE were calculated to assess model performance. Visualization Insights:

Scatter plots, residual plots, and correlation matrices revealed relationships and potential multicollinearity in features. Residual plots helped assess the goodness of fit for regression models. Histograms and Q-Q plots evaluated the normality of residuals. Data Cleaning:

Addressed errors caused by non-numeric data and missing columns. Proper scaling and handling of categorical data were critical for clustering and regression models. Model Evaluation:

Ridge and Lasso regression controlled overfitting, with Lasso being particularly useful for feature selection. Polynomial regression captured complex relationships but required careful tuning to avoid overfitting. Residual analysis highlighted areas where models could improve. Practical Coding Skills:

Learned to structure data preprocessing, scaling, and model training in steps. Developed reusable code for visualizations and statistical evaluations. Identified the importance of verifying assumptions before drawing conclusions. Plain Python Code Summary K-Means Clustering python Copy code from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4, random_state=1) data['Cluster'] = kmeans.fit_predict(data_scaled) print(data.groupby('Cluster').mean()) Linear Regression python Copy code from sklearn.linear_model import LinearRegression

model = LinearRegression() model.fit(X_train, y_train) predictions = model.predict(X_test) print(f"Intercept: {model.intercept_}, Coefficients: {model.coef_}") Polynomial Regression python Copy code from sklearn.preprocessing import PolynomialFeatures from sklearn.linear_model import LinearRegression

poly = PolynomialFeatures(degree=2) X_poly = poly.fit_transform(X) model = LinearRegression() model.fit(X_poly, y) Residual Plot python Copy code import matplotlib.pyplot as plt

residuals = y_test - predictions plt.scatter(predictions, residuals) plt.axhline(0, color='red', linestyle='--') plt.xlabel('Predicted Values') plt.ylabel('Residuals') plt.title('Residuals vs Predicted') plt.show() Data Cleaning python Copy code data = data.dropna() # Remove missing values data_scaled = scaler.fit_transform(data.select_dtypes(include='number')) What We Learned K-Means is powerful for segmenting data but requires careful feature selection. Regression methods vary in complexity; polynomial fits non-linear data better but is prone to overfitting. Visualizations are essential to understand patterns, detect anomalies, and verify model assumptions. Preprocessing is key to avoid errors and ensure meaningful model results. Evaluation metrics guide improvements, showing how well models perform and fit the data.

In [255…

```
!pwd
```

```
/Users/obaozai/Data/GitHub/Inferetntial/case5
```

In [5]:

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



from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

new_data = pd.read_csv("data_add_nik.csv")

# Select relevant numerical features for clustering
features = new_data[['Listing Price', 'Sale Price', 'Discount', 'Rating', 'F

# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Perform K-Means clustering with k=2 (as determined earlier)
kmeans = KMeans(n_clusters=2, random_state=1)
clusters = kmeans.fit_predict(scaled_features)

# Add cluster labels to the dataset
new_data['Cluster'] = clusters

# Calculate the average listing price for each cluster
cluster_avg_price = new_data.groupby('Cluster')['Listing Price'].mean()

print(cluster_avg_price)
```

```
Cluster
0    7199.305499
1    6369.059816
Name: Listing Price, dtype: float64
```

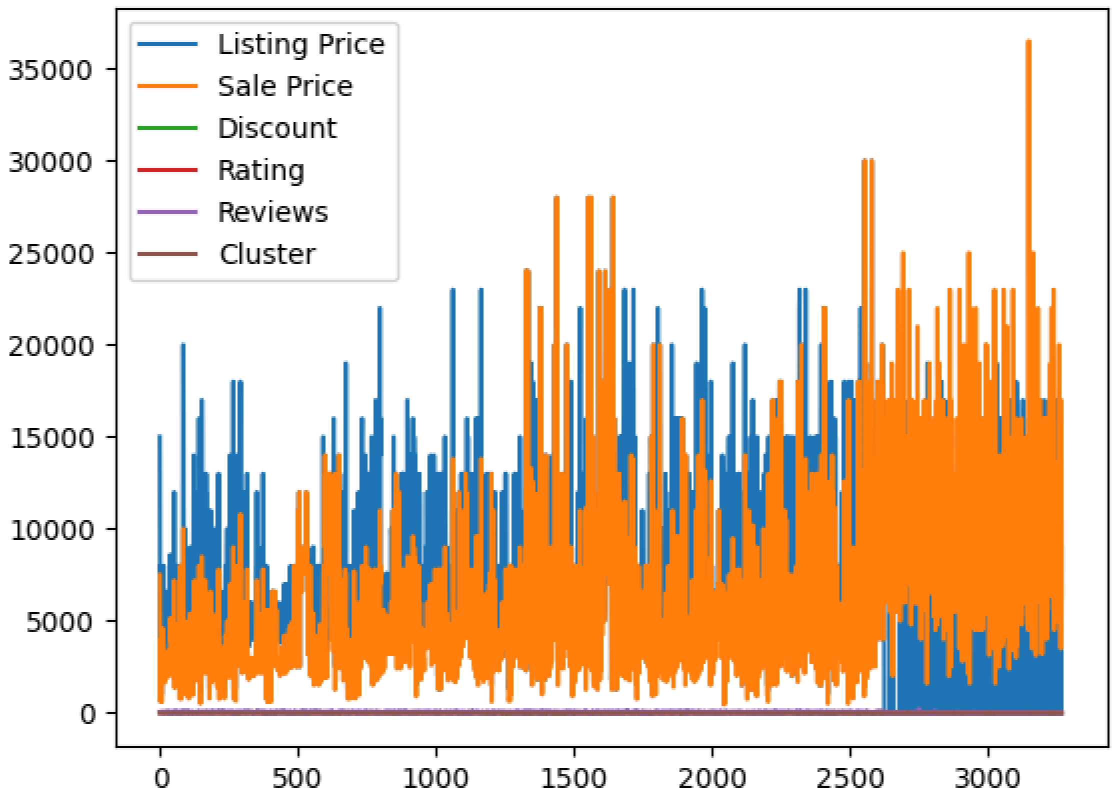
In [7]:

```
new_data.memory_usage(deep=True)
```

Out[7]:
```
Index               132
Product Name     307429
Product ID       208456
Listing Price     26144
Sale Price        26144
Discount          26144
Brand            236814
Rating            26144
Reviews           26144
Cluster           13072
dtype: int64
```

In [8]:
```
new_data.plot()
```

Out[8]: <Axes: >



In [14]:
```
new_data.infer_objects(copy=False)
```

Out[14]:

	Product Name	Product ID	Listing Price	Sale Price	Discount	Brand	Rating	Reviews	C
0	Women's adidas Originals NMD_Racer Primeknit S...	AH2430	14999	7499	50	Adidas Adidas ORIGINALS	4.8	41	
1	Women's adidas Originals Sleek Shoes	G27341	7599	3799	50	Adidas ORIGINALS	3.3	24	
2	Women's adidas Swim Puka Slippers	CM0081	999	599	40	Adidas CORE / NEO	2.6	37	
3	Women's adidas Sport Inspired Questar Ride Shoes	B44832	6999	3499	50	Adidas CORE / NEO	4.1	35	
4	Women's adidas Originals Taekwondo Shoes	D98205	7999	3999	50	Adidas ORIGINALS	3.5	72	
...	...	...	...	...	...	...	...	...	
3263	Air Jordan 8 Retro	CI1236-100	15995	12797	0	Nike	5.0	1	
3264	Nike Phantom Venom Club IC	AO0578-717	4995	3497	0	Nike	0.0	0	
3265	Nike Mercurial Superfly 7 Academy TF	AT7978-414	8495	5947	0	Nike	5.0	1	
3266	Nike Air Max 98	AH6799-300	0	16995	0	Nike	4.0	4	
3267	Nike P-6000 SE	CJ9585-600	8995	6297	0	Nike	0.0	0	

3268 rows × 9 columns

In [16]:

```
new_data.infer_objects(copy=False)
```

Out[16]:

	Product Name	Product ID	Listing Price	Sale Price	Discount	Brand	Rating	Reviews	C
0	Women's adidas Originals NMD_Racer Primeknit S...	AH2430	14999	7499	50	Adidas Adidas ORIGINALS	4.8	41	
1	Women's adidas Originals Sleek Shoes	G27341	7599	3799	50	Adidas ORIGINALS	3.3	24	
2	Women's adidas Swim Puka Slippers	CM0081	999	599	40	Adidas CORE / NEO	2.6	37	
3	Women's adidas Sport Inspired Questar Ride Shoes	B44832	6999	3499	50	Adidas CORE / NEO	4.1	35	
4	Women's adidas Originals Taekwondo Shoes	D98205	7999	3999	50	Adidas ORIGINALS	3.5	72	
...	...	...	...	...	...	...	...	...	
3263	Air Jordan 8 Retro	CI1236-100	15995	12797	0	Nike	5.0	1	
3264	Nike Phantom Venom Club IC	AO0578-717	4995	3497	0	Nike	0.0	0	
3265	Nike Mercurial Superfly 7 Academy TF	AT7978-414	8495	5947	0	Nike	5.0	1	
3266	Nike Air Max 98	AH6799-300	0	16995	0	Nike	4.0	4	
3267	Nike P-6000 SE	CJ9585-600	8995	6297	0	Nike	0.0	0	

3268 rows × 9 columns

In [15]:
```
new_data.interpolate(method='barycentric')
```

```
/var/folders/q0/xfs5xjxx50xdjh4tzn1psdnw0000gn/T/ipykernel_4091/2992733257.p
y:1: FutureWarning: DataFrame.interpolate with object dtype is deprecated an
d will raise in a future version. Call obj.infer_objects(copy=False) before
interpolating instead.
  new_data.interpolate(method='barycentric')
```

Out[15]:

	Product Name	Product ID	Listing Price	Sale Price	Discount	Brand	Rating	Reviews	C
0	Women's adidas Originals NMD_Racer Primeknit S...	AH2430	14999	7499	50	Adidas Adidas ORIGINALS	4.8	41	
1	Women's adidas Originals Sleek Shoes	G27341	7599	3799	50	Adidas ORIGINALS	3.3	24	
2	Women's adidas Swim Puka Slippers	CM0081	999	599	40	Adidas CORE / NEO	2.6	37	
3	Women's adidas Sport Inspired Questar Ride Shoes	B44832	6999	3499	50	Adidas CORE / NEO	4.1	35	
4	Women's adidas Originals Taekwondo Shoes	D98205	7999	3999	50	Adidas ORIGINALS	3.5	72	
...	...	...	...	...	...	...	...	...	
3263	Air Jordan 8 Retro	CI1236-100	15995	12797	0	Nike	5.0	1	
3264	Nike Phantom Venom Club IC	AO0578-717	4995	3497	0	Nike	0.0	0	
3265	Nike Mercurial Superfly 7 Academy TF	AT7978-414	8495	5947	0	Nike	5.0	1	
3266	Nike Air Max 98	AH6799-300	0	16995	0	Nike	4.0	4	
3267	Nike P-6000 SE	CJ9585-600	8995	6297	0	Nike	0.0	0	

3268 rows × 9 columns

In [ ]:

In [ ]:

In [ ]: