eda-project

November 17, 2023

EDA Project(2023)

# AMAZON PRODUCT SALES ANALYSIS

---

## 1 TABLE OF CONTENT

    * Multi-variate analysis
    * Summary statisctis
    * Visualizations
    * Insighs

- Distribution
- Hypothesis Testing
- Conclusion

---

### 1.0.1 REQUIRED LIBRARIES

[203]:
```
import pandas as pd # used for Data Manuplation and Handling
import numpy as np # used for Numerical operations
import statistics as stat # used for Statistical Calculations
import matplotlib.pyplot as plt # used for plotting graphs(python plotting␣
 ↪package)
import seaborn as sb # used for visualization
```

.

---

## 2 DATA COLLECTION

- Data collected from Kaggle

### 2.0.1 IMPORTING DATA SET USING ITS PATH

[204]:
```
path = "C:\\Users\\uppada satwik\\Downloads\\archive\\Amazon-Products.csv"
pd.set_option('display.expand_frame_repr',  False)
data = pd.read_csv(path, low_memory = False)
```

## 3 DATASET WALKTHROUGH

[205]:
```
data.head(5)
```

[205] :
```
   Unnamed: 0                                               name  main_category
sub_category                                               image
link  ratings  no_of_ratings  discount_price  actual_price
0           0  Lloyd 1.5 Ton 3 Star Inverter Split Ac (5 In 1…     appliances
Air Conditioners   https://m.media-amazon.com/images/I/31UISB90sY…
https://www.amazon.in/Lloyd-Inverter-Convertib…     4.2          2,255
 32,999       58,990
1           1  LG 1.5 Ton 5 Star AI DUAL Inverter Split AC (C…     appliances
Air Conditioners   https://m.media-amazon.com/images/I/51JFb7FctD…
```

```
https://www.amazon.in/LG-Convertible-Anti-Viru…        4.2          2,948
 46,490       75,990
2              2  LG 1 Ton 4 Star Ai Dual Inverter Split Ac (Cop…     appliances
Air Conditioners    https://m.media-amazon.com/images/I/51JFb7FctD…
https://www.amazon.in/LG-Inverter-Convertible-…        4.2          1,206
 34,490       61,990
3              3  LG 1.5 Ton 3 Star AI DUAL Inverter Split AC (C…      appliances
Air Conditioners    https://m.media-amazon.com/images/I/51JFb7FctD…
https://www.amazon.in/LG-Convertible-Anti-Viru…        4.0            69
 37,990       68,990
4              4  Carrier 1.5 Ton 3 Star Inverter Split AC (Copp…      appliances
Air Conditioners    https://m.media-amazon.com/images/I/41lrtqXPiW…
https://www.amazon.in/Carrier-Inverter-Split-C…        4.1           630
 34,490       67,790
```

# 4 DATA INSPECTION

### 4.0.1 To find number of rows in the data set

[206] :
```
data.shape[0]
```

[206]:
```
551585
```

### 4.0.2 To find number of columns in the dataset

[207] :
```
data.shape[1]
```

[207]:
```
10
```

1. What is the size if the data set? A. size of the data set is 551585 * 10

### 4.0.3 What is the structure and integrity of the dataset?

[208] :
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 551585 entries, 0 to 551584
Data columns (total 10 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Unnamed: 0      551585 non-null  int64
 1   name            551585 non-null  object
 2   main_category   551585 non-null  object
 3   sub_category    551585 non-null  object
 4   image           551585 non-null  object
 5    link               551585 non-null   object
 6    ratings            375791 non-null   object
 7    no_of_ratings      375791 non-null   object
 8    discount_price     490422 non-null   object
 9    actual_price       533772  non-null object
dtypes:  int64(1),  object(9)
memory usage: 42.1+ MB
```

- we can see some missing values in discount_price, actual_price, ratings, no_of_ratings

### 4.0.4 To display column names of the dataset

[209] :
```
# Showing columns
data.columns
```

[209] :
```
Index(['Unnamed: 0', 'name', 'main_category', 'sub_category', 'image', 'link',
       'ratings',  'no_of_ratings',  'discount_price',  'actual_price'],
       dtype='object')
```

## 5 UNDERSTANDING DATASET

### 5.0.1 To display statistical values of the dataset

[210] :
```
data.describe()
```

[210]:
```
          Unnamed: 0
count  551585.000000
mean     7006.200471
std      5740.835523
min         0.000000
25%      1550.000000
50%      5933.000000
75%     11482.000000
max     19199.000000
```

- showing only for unnamed column because the data set contains only one numerical column rest are set to object type. we need to change them in data tranformation part

### 5.0.2 To display first 2 rows of the data set

[211] :
```
# First 2 rows in the data
data.head(2)
```

[211] :
```
    Unnamed: 0                                              name  main_category
sub_category                                                image
link  ratings  no_of_ratings  discount_price  actual_price
0              0  Lloyd 1.5 Ton 3 Star Inverter Split Ac (5 In 1…      appliances
Air Conditioners   https://m.media-amazon.com/images/I/31UISB90sY…
```

```
https://www.amazon.in/Lloyd-Inverter-Convertib…        4.2            2,255
 32,999        58,990
1              1   LG 1.5 Ton 5 Star AI DUAL Inverter Split AC (C…    appliances
Air Conditioners    https://m.media-amazon.com/images/I/51JFb7FctD…
https://www.amazon.in/LG-Convertible-Anti-Viru…      4.2            2,948
 46,490        75,990
```

---

## 6 DATA CLEANING

#### 6.0.1 Dropping unneccesary columns

- we don't want image and link columns for analysis so we can remove them

[212] :
```
columns_to_drop    =    ['image','link']
data  =data.drop(columns=columns_to_drop)
```

- we can see that Unamed: 0 columns just shows index, so it is useless, we can delete this one also

[213] :
```
del data['Unnamed: 0'] # data.drop(columns='Unnamed: 0',inplace= True)
```

### 6.1 Now required data is there , lets check the dtypes

---

[214] :
```
data.info() # info about column names,non null values,DTYPES
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 551585 entries, 0 to 551584
Data columns (total 7 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   name            551585 non-null  object
 1   main_category   551585 non-null  object
 2   sub_category    551585 non-null  object
 3   ratings         375791 non-null  object
 4   no_of_ratings   375791 non-null  object
 5   discount_price  490422 non-null  object
 6   actual_price    533772 non-null  object
dtypes: object(7)
memory usage: 29.5+ MB
```

[215] :
```
data.isnull().sum()
```

[215] :
```
name               0
main_category      0
sub_category       0
```

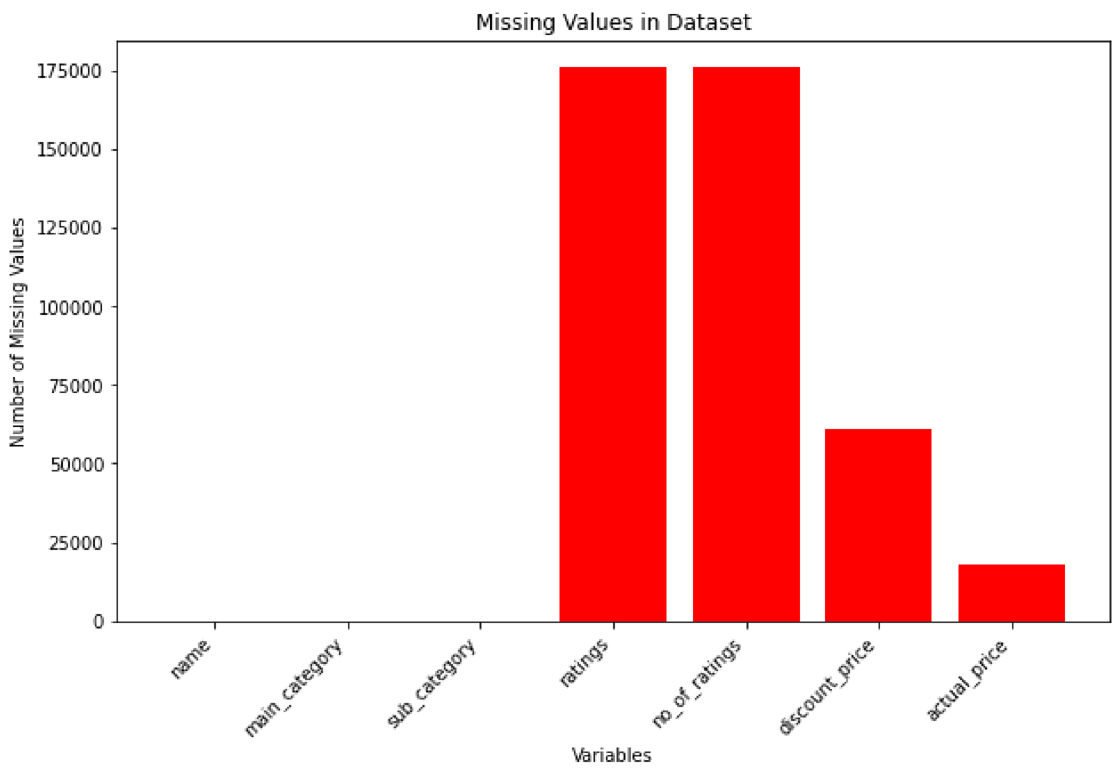
```
ratings           175794
no_of_ratings     175794
discount_price     61163
actual_price       17813
dtype: int64
```

- We can see that all the columns are in string format, it is an error that ratings, no.of ratings, discount_price, actual_price must be in numerical values
- And one more thing is difference in non null values in column
- Lets change them

### 6.2 Displaying Missing values

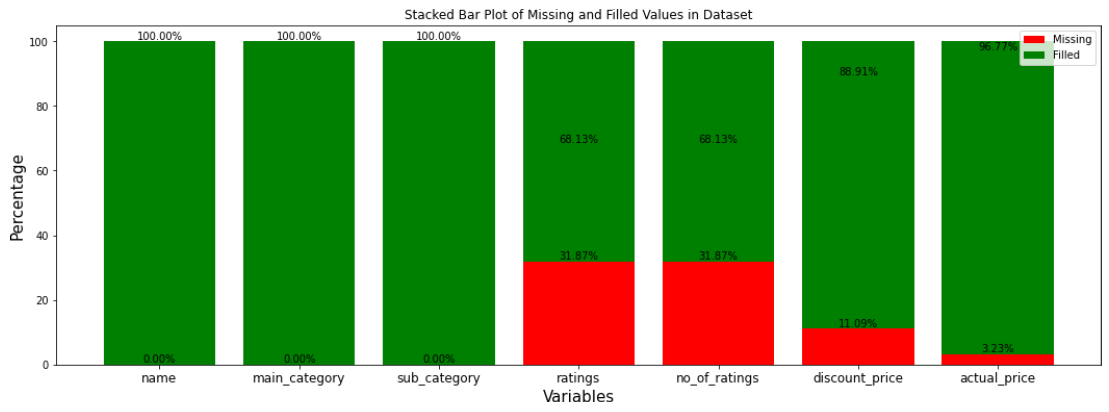
[216] :
```
missing_values= data.isnull().sum()
```

[217] :
```
plt.figure(figsize=(10, 6))
plt.bar(missing_values.index, missing_values.values, color='red')
plt.xlabel('Variables')
plt.ylabel('Number of Missing Values')
plt.title('Missing Values in Dataset')
plt.xticks(rotation=45, ha='right')
plt.show()
```



## 7 Handling Null/Missing values

[218]:
```
missing_percentage = (data.isnull().sum() / len(data)) * 100
filled_percentage = 100 - missing_percentage


plt.figure(figsize=(18, 6))

bars1= plt.bar(missing_percentage.index, missing_percentage, label='Missing',␣
 ↪color='red')
 bars2= plt.bar(filled_percentage.index, filled_percentage,␣
   ↪bottom=missing_percentage,  label='Filled',  color='green')

plt.xlabel('Variables',fontsize=15)
plt.ylabel('Percentage',fontsize=15)
plt.title('Stacked Bar Plot of Missing and Filled Values in Dataset')
plt.xticks(fontsize=12)
plt.legend()

for bar1, bar2 in zip(bars1, bars2):
     yval = bar1.get_height()
     plt.text(bar1.get_x() + bar1.get_width() / 2, yval, f'{yval:.2f}%',␣
 ↪ha='center', va='bottom')
#
for bar1, bar2 in zip(bars1, bars2):
     yval = bar2.get_height()
     plt.text(bar2.get_x() + bar2.get_width() / 2, yval, f'{yval:.2f}%',␣
 ↪ha='center', va='bottom')

plt.show()
```



- now we conclude that no column is having 70% missing values

## 8 RATINGS

[219] :
```
data['ratings'].unique()
```

[219] :
```
array(['4.2', '4.0', '4.1', '4.3', '3.9', '3.8', '3.5', nan, '4.6', '3.3',
       '3.4', '3.7', '2.9', '5.0', '4.4', '3.6', '2.7', '4.5', '3.0',
       '3.1', '3.2', '4.8', '4.7', '2.5', '1.0', '2.6', '2.8', '2.3',
       '1.7', 'Get', '1.8', '2.4', '4.9', '2.2', '1.6', '1.9', '2.0',
       '1.4', '2.1', 'FREE', '1.2', '1.3', '1.5', ' 68.99', ' 65', '1.1',
       ' 70', ' 100', ' 99', ' 2.99'], dtype=object)
```

[220] :
```
replace_dict = {
    np.nan:'0',
    'Get':'0',
    'FREE':'0',
    ' 68.99':'0',
    ' 65':'0',
    ' 70':'0',
    ' 100':'0',
    ' 99':'0',
    ' 2.99':'0'

}
data['ratings']    =    data['ratings'].replace(replace_dict)
```

- we successfully replace the abnormal values and NAN values to 0

[221] :
```
data['ratings'].unique()
```

[221] :
```
array(['4.2', '4.0', '4.1', '4.3', '3.9', '3.8', '3.5', '0', '4.6', '3.3',
       '3.4', '3.7', '2.9', '5.0', '4.4', '3.6', '2.7', '4.5', '3.0',
       '3.1', '3.2', '4.8', '4.7', '2.5', '1.0', '2.6', '2.8', '2.3',
       '1.7', '1.8', '2.4', '4.9', '2.2', '1.6', '1.9', '2.0', '1.4',
       '2.1', '1.2', '1.3', '1.5', '1.1'], dtype=object)
```

## 9 NO_OF_RATINGS

[222] :
```
data['no_of_ratings'].unique()
```

[222] :
```
array(['2,255', '2,948', '1,206', ..., '3,329', '7,141', '4,406'],
      dtype=object)
```

[223]:
```
data['no_of_ratings'].isna().value_counts()
```

[223]:
```
no_of_ratings
False    375791
True     175794
Name: count, dtype: int64
```

- Replacing NAN with 0

[224]:
```
replace_dict = {
    np.nan :"0"
}

data['no_of_ratings']=data['no_of_ratings'].replace(replace_dict)
```

- we have successfully replace the NAN values with 0

[225]:
```
data['no_of_ratings'].isna().value_counts()
```

[225]:
```
no_of_ratings
False    551585
Name: count, dtype: int64
```

- Data has been cleaned with out null and abnormal values (replaced with 0 )

## 10 DISCOUNT PRICE

[226]:
```
data['discount_price'].isna().value_counts()
```

[226]:
```
discount_price
False    490422
True      61163
Name: count, dtype: int64
```

- Here we can see some null values are there, let us replace with 0 for now

[227]:
```
data['discount_price'].unique()
```

[227]:
```
array([' 32,999', ' 46,490', ' 34,490', ..., ' 3,712.10', ' 1,429.60',
       ' 651.01'], dtype=object)
```

- Removing rupees symbol

[228]:
```
data['discount_price']=data['discount_price'].str.replace('[^0-9]', '',␣
↪regex=True)
```

- Replacing NAN values with 0.0

```
[229]: data['discount_price']=data['discount_price'].replace(np.nan,0.0)
```

```
[230]: data['discount_price'].isna().value_counts()
```

```
[230]: discount_price
       False    551585
       Name: count, dtype: int64
```

- Null values are replaced with 0 , now there is no null values

## 11 ACTUAL PRICE

```
[231]: data['actual_price'].isna().value_counts()
```

```
[231]: actual_price
       False    533772
       True      17813
       Name: count, dtype: int64
```

- Here, We can observe some null values

```
[232]: data['actual_price'].unique()
```

```
[232]: array([' 58,990', ' 75,990', ' 61,990', ..., ' 608.97', ' 4,792',
              ' 8,023.60'], dtype=object)
```

- Removing rupees symbol

```
[233]: data['actual_price']=data['actual_price'].str.replace('[^0-9]', '', regex=True)
```

```
[234]: data['actual_price'].unique()
```

```
[234]: array(['58990', '75990', '61990', ..., '60897', '4792', '802360'],
             dtype=object)
```

- Replace NAN with 0.0

```
[235]: data['actual_price']=data['actual_price'].replace(np.nan,0.0)
```

- We successfully replace the null values with 0

```
[236]: data['actual_price'].isna().value_counts()
```

```
[236]: actual_price
       False    551585
       Name: count, dtype: int64
```

- Null values are replaced with 0 , now there is no null values

[237] :
```
percentage_missing = (data.isnull().sum() / len(data)) * 100
percentage_missing
```

[237] :
```
name              0.0
main_category     0.0
sub_category      0.0
ratings           0.0
no_of_ratings     0.0
discount_price    0.0
actual_price      0.0
dtype: float64
```

[238] :
```
data.isnull().sum()
```

[238] :
```
name              0
main_category     0
sub_category      0
ratings           0
no_of_ratings     0
discount_price    0
actual_price      0
dtype: int64
```

---

## 12 DATA TRANFORMATION

- Changing dtypes according to the data

### 12.1 Ratings column

[239] :
```
data['ratings']=data['ratings'].astype(float)
```

### 12.2 No_of_Ratings Column

[240] :
```
def convert_to_int(value):
    try:
        return int(value.replace(',', '').replace('Only left in stock.', '').
 ↪strip())
    except (ValueError, AttributeError):
        return 0  # Return 0 for non-convertible values

# Apply the custom function to the column
data['no_of_ratings'] = data['no_of_ratings'].apply(convert_to_int)
```

## 12.3 Discount_price column

```
[241]: data['discount_price'] = data['discount_price'].astype(float)
```

## 12.4 Actual_price column

```
[242]: data['actual_price'] = data['actual_price'].astype(float)
```

---

## 12.5 CHECKING DUPLICATE ROWS

```
[243]: data.drop_duplicates()
```

```
[243]:                                                          name     main_category  
sub_category  ratings  no_of_ratings  discount_price  actual_price  
0       Lloyd 1.5 Ton 3 Star Inverter Split Ac (5 In 1...        appliances   Air
Conditioners        4.2           2255         32999.0       58990.0  
1         LG 1.5 Ton 5 Star AI DUAL Inverter Split AC (C...        appliances   Air
Conditioners        4.2           2948         46490.0       75990.0  
2         LG 1 Ton 4 Star Ai Dual Inverter Split Ac (Cop...        appliances   Air
Conditioners        4.2           1206         34490.0       61990.0  
3         LG 1.5 Ton 3 Star AI DUAL Inverter Split AC (C...        appliances   Air
Conditioners        4.0             69         37990.0       68990.0  
4         Carrier 1.5 Ton 3 Star Inverter Split AC (Copp...        appliances   Air
Conditioners        4.1            630         34490.0       67790.0  
...                                                    ...                 ...
...        ...            ...             ...           ...  
551580                   Adidas Regular Fit Men's Track Tops  sports & fitness
Yoga       3.2              9          3449.0        4599.0  
551581             Redwolf Noice Toit Smort - Hoodie (Black)  sports & fitness
Yoga       2.0              2          1199.0        1999.0  
551582      Redwolf Schrute Farms B&B - Hoodie (Navy Blue) sports & fitness
Yoga       4.0              1          1199.0        1999.0  
551583                                      Puma Men Shorts    sports & fitness
Yoga       4.4             37             0.0           0.0  
551584 Mothercare Printed Cotton Elastane Girls Infan...   sports & fitness
Yoga       4.6              5          1039.0        1299.0  

[515534 rows x 7 columns]
```

- No duplicate rows found. (becuase no change in rows)

---

#

DATA CLEANING WAS COMPLETED

---

[244] :
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 551585 entries, 0 to 551584
Data columns (total 7 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   name            551585 non-null  object
 1   main_category   551585 non-null  object
 2   sub_category    551585 non-null  object
 3   ratings         551585 non-null  float64
 4   no_of_ratings   551585 non-null  int64
 5   discount_price  551585 non-null  float64
 6   actual_price    551585 non-null  float64
dtypes: float64(3), int64(1), object(3)
memory usage: 29.5+ MB
```

- We change the dtypes and equalize the content

[245] :
```
data.describe()
```

[245]:
```
             ratings  no_of_ratings  discount_price  actual_price
count  551585.000000  551585.000000    5.515850e+05  5.515850e+05
mean        2.567621     563.311245    6.258484e+03  4.669279e+04
std         1.905329    7092.573805    6.774411e+04  1.333252e+07
min         0.000000       0.000000    0.000000e+00  0.000000e+00
25%         0.000000       0.000000    2.990000e+02  8.990000e+02
50%         3.500000       4.000000    5.990000e+02  1.499000e+03
75%         4.100000      49.000000    1.340000e+03  2.999000e+03
max         5.000000  589547.000000    1.065632e+07  9.900000e+09
```

---

[246] :
```
missing_values= data.isnull().sum()
```

[247] :
```
print(missing_values)
```

```
name              0
main_category     0
sub_category      0
ratings           0
no_of_ratings     0
discount_price    0
actual_price      0
dtype: int64
```

---

## 13 NORMALIZATION

#### 13.0.1 RATING COLUMN

```
[248]: data['ratings'].value_counts(normalize=True)*100
```

```
[248]: ratings
       0.0    33.000716
       4.0     6.637055
       5.0     5.960097
       3.9     4.912208
       3.8     4.587688
       4.1     4.497947
       3.7     4.022952
       4.2     3.936655
       3.6     3.532003
       4.3     3.190986
       3.5     3.189898
       4.4     2.503513
       3.4     2.357932
       4.5     2.287589
       3.0     2.138746
       3.3     1.977030
       4.6     1.513457
       3.2     1.481005
       1.0     1.472121
       3.1     1.181142
       4.7     0.972652
       2.9     0.750383
       2.0     0.567999
       2.8     0.551864
       2.7     0.457409
       2.5     0.439280
       4.8     0.424776
       2.6     0.362591
       2.4     0.227164
       2.3     0.174406
       2.2     0.137966
       2.1     0.115848
       4.9     0.094455
       1.5     0.087747
       1.9     0.064904
       1.8     0.058015
       1.7     0.040792
       1.4     0.038253
       1.6     0.031002
       1.3     0.016679
       1.2     0.004895
```

```
1.1      0.000181
Name: proportion, dtype: float64
```

- Above we convert abnormal values to 0.
- It has the highest percentage so that we need to change them into 4.0 which is median of all

[249] :
```
data['ratings']=np.where(data['ratings']==0.0,data['ratings'].
    median(),data['ratings'])
```

[250] :
```
data['ratings'].value_counts(normalize=True)*100
```

[250]:
```
ratings
3.5    36.190614
4.0     6.637055
5.0     5.960097
3.9     4.912208
3.8     4.587688
4.1     4.497947
3.7     4.022952
4.2     3.936655
3.6     3.532003
4.3     3.190986
4.4     2.503513
3.4     2.357932
4.5     2.287589
3.0     2.138746
3.3     1.977030
4.6     1.513457
3.2     1.481005
1.0     1.472121
3.1     1.181142
4.7     0.972652
2.9     0.750383
2.0     0.567999
2.8     0.551864
2.7     0.457409
2.5     0.439280
4.8     0.424776
2.6     0.362591
2.4     0.227164
2.3     0.174406
2.2     0.137966
2.1     0.115848
4.9     0.094455
1.5     0.087747
1.9     0.064904
1.8     0.058015
```

```
1.7       0.040792
1.4       0.038253
1.6       0.031002
1.3       0.016679
1.2       0.004895
1.1       0.000181
Name: proportion, dtype: float64
```

---

### 13.0.2 NO_OF_RATINGS

[251]:
```
data['no_of_ratings'].value_counts(normalize=True)*100
```

[251]:
```
no_of_ratings
0        33.012138
1         7.218470
2         4.484712
3         3.282359
4         2.606307
           ...
6804      0.000181
9538      0.000181
5961      0.000181
12918     0.000181
4406      0.000181
Name: proportion, Length: 8285, dtype: float64
```

[252]:
```
data['no_of_ratings']=np.where(data['no_of_ratings']==0.0,data['no_of_ratings'].
 ↪median(),data['no_of_ratings'])
```

[253]:
```
data['no_of_ratings'].value_counts(normalize=True)*100
```

[253]:
```
no_of_ratings
4.0        35.618445
1.0         7.218470
2.0         4.484712
3.0         3.282359
5.0         2.111189
             ...
55589.0     0.000181
13188.0     0.000181
4724.0      0.000181
7735.0      0.000181
4406.0      0.000181
Name: proportion, Length: 8284, dtype: float64
```

---

### 13.0.3 DISCOUNT PRICE

[254]:
```
data['discount_price'].value_counts(normalize=True)*100
```

[254] :
```
discount_price
0.0        11.088590
499.0       3.308284
299.0       2.780532
399.0       2.630601
999.0       2.356844
              ...
6631.0      0.000181
15540.0     0.000181
13860.0     0.000181
8826.0      0.000181
65101.0     0.000181
Name: proportion, Length: 26677, dtype: float64
```

[255] :
```
data['discount_price']=np.where(data['discount_price']==0.
↪0,data['discount_price'].median(),data['discount_price'])
```

[256] :
```
data['discount_price'].value_counts(normalize=True)*100
```

[256]:
```
discount_price
599.0      13.220628
499.0       3.308284
299.0       2.780532
399.0       2.630601
999.0       2.356844
              ...
6631.0      0.000181
15540.0     0.000181
13860.0     0.000181
8826.0      0.000181
65101.0     0.000181
Name: proportion, Length: 26676, dtype: float64
```

---

### 13.0.4 ACTUAL PRICE

[257]:
```
data['actual_price'].value_counts(normalize=True)*100
```

[257] :
```
actual_price
999.0       8.842517
1999.0      4.676523
1499.0      3.409991
0.0         3.229965
```

```
499.0         2.611746
                 ...
54303.0      0.000181
41697.0      0.000181
59768.0      0.000181
34210.0      0.000181
802360.0     0.000181
Name: proportion, Length: 23124, dtype: float64
```

[258] :
```
data['actual_price']=np.where(data['actual_price']==0.0,data['actual_price'].
  ↪median(),data['actual_price'])
```

[259] :
```
data['actual_price'].value_counts(normalize=True)*100
```

[259]:
```
actual_price
999.0         8.842517
1499.0        6.639956
1999.0        4.676523
499.0         2.611746
1299.0        2.550468
                 ...
54303.0       0.000181
41697.0       0.000181
59768.0       0.000181
34210.0       0.000181
802360.0      0.000181
Name: proportion, Length: 23123, dtype: float64
```

---

## 14 ADDING EXTRA COLUMNS FOR BETTER INSIGHTS

[260] :
```
data.columns
```

[260] :
```
Index(['name', 'main_category', 'sub_category', 'ratings', 'no_of_ratings',
       'discount_price', 'actual_price'],
      dtype='object')
```

- Sales percentage
- Discount Amount
- Rating level
- Discount percentage

### 14.1 Sales Percentage

[261] :
```
data['sales_per'] = ((data['discount_price']/data['actual_price'])*100).round(2)
```

### 14.2 Discount Offered

[262] :
```
data['Discount_Offered'] = data['actual_price']-data['discount_price']
```

### 14.3 Rating level

[263] :
```
conditions = [
    (data['ratings'] > 4.0),
    (data['ratings'] > 3.2),
]

choices = ['Top rated', 'Average']

data['Rating_level'] = np.select(conditions, choices, default='Low')
```

### 14.4 Discount percentage

[264] :
```
data['Discount_per'] = 1- data['discount_price']/data['actual_price']
```

[265] :
```
data['discount_price'].max()
```

[265]:
```
10656317.0
```

[266] :
```
data['discount_price'].min()
```

[266]:
```
8.0
```

---

### 14.5 Brand Name

[267] :
```
data['Brand_Name'] = data['name'].str.split(" ").str[0]
```

[268] :
```
  columns = ['name','Brand_Name', 'main_category',␣
↪'sub_category','actual_price','discount_price',
             'Discount_Offered', 'Discount_per' ,'sales_per',
             'ratings', 'Rating_level','no_of_ratings' ]
data=data[columns]
```

[269] :
```
data.select_dtypes(include=['object','category'])
```

```
[269] :                                                         name  Brand_Name  
main_category        sub_category Rating_level
0        Lloyd 1.5 Ton 3 Star Inverter Split Ac (5 In 1...        Lloyd
appliances   Air Conditioners      Top rated
1        LG 1.5 Ton 5 Star AI DUAL Inverter Split AC (C...           LG
appliances   Air Conditioners      Top rated
2        LG 1 Ton 4 Star Ai Dual Inverter Split Ac (Cop...           LG
appliances   Air Conditioners      Top rated
3        LG 1.5 Ton 3 Star AI DUAL Inverter Split AC (C...           LG
appliances   Air Conditioners        Average
4        Carrier 1.5 Ton 3 Star Inverter Split AC (Copp...      Carrier
appliances   Air Conditioners      Top rated
...                                                    ...          ...
...                  ...           ...
551580               Adidas Regular Fit Men's Track Tops      Adidas  sports &
fitness                   Yoga           Low
551581         Redwolf Noice Toit Smort - Hoodie (Black)     Redwolf  sports &
fitness                   Yoga           Low
551582    Redwolf Schrute Farms B&B - Hoodie (Navy Blue)     Redwolf  sports &
fitness                   Yoga       Average
551583                                  Puma Men Shorts        Puma  sports &
fitness                   Yoga     Top rated
551584 Mothercare Printed Cotton Elastane Girls Infan...   Mothercare  sports &
fitness                   Yoga     Top rated

[551585 rows x 5 columns]
```

---

```
[270] : data.columns
```

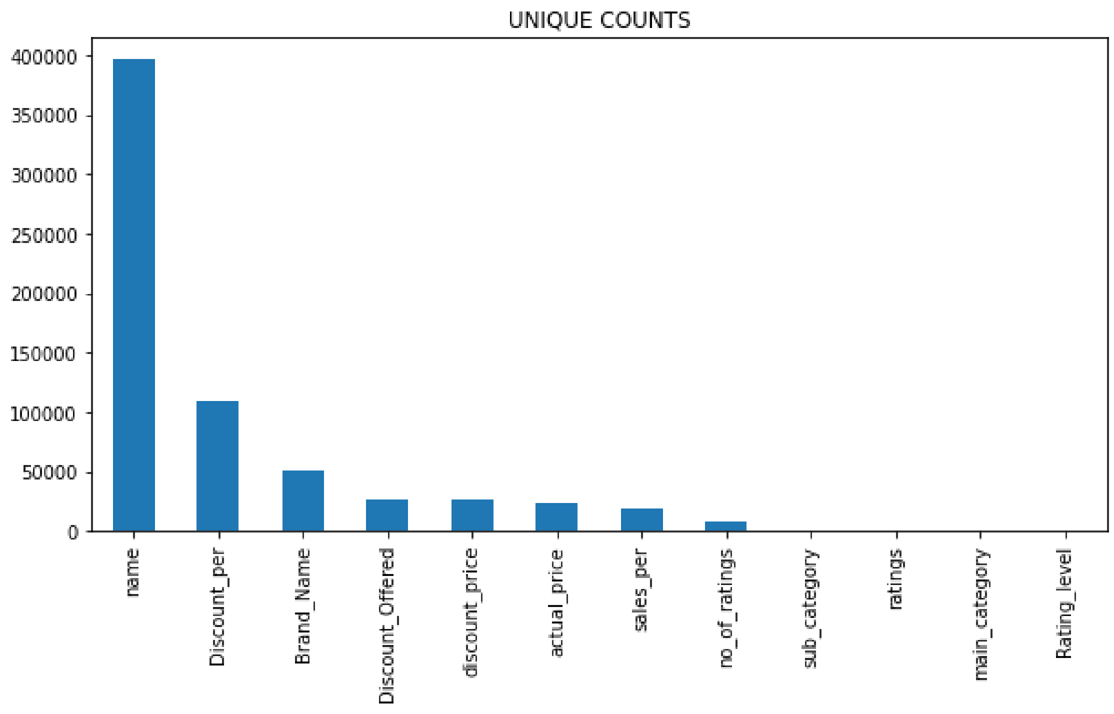
```
[270] : Index(['name', 'Brand_Name', 'main_category', 'sub_category', 'actual_price',
              'discount_price', 'Discount_Offered', 'Discount_per', 'sales_per',
              'ratings', 'Rating_level', 'no_of_ratings'],
             dtype='object')
```

```
[271] : data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 551585 entries, 0 to 551584
Data columns (total 12 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   name              551585 non-null  object
 1   Brand_Name        551585 non-null  object
 2   main_category     551585 non-null  object
 3   sub_category      551585 non-null  object
 4   actual_price      551585 non-null  float64
 5    discount_price       551585 non-null   float64
 6    Discount_Offered     551585 non-null   float64
 7    Discount_per         551585 non-null   float64
 8    sales_per            551585 non-null   float64
 9    ratings              551585 non-null   float64
 10   Rating_level         551585 non-null   object
 11   no_of_ratings        551585  non-null float64
dtypes:  float64(7),  object(5)
memory usage: 50.5+ MB
```

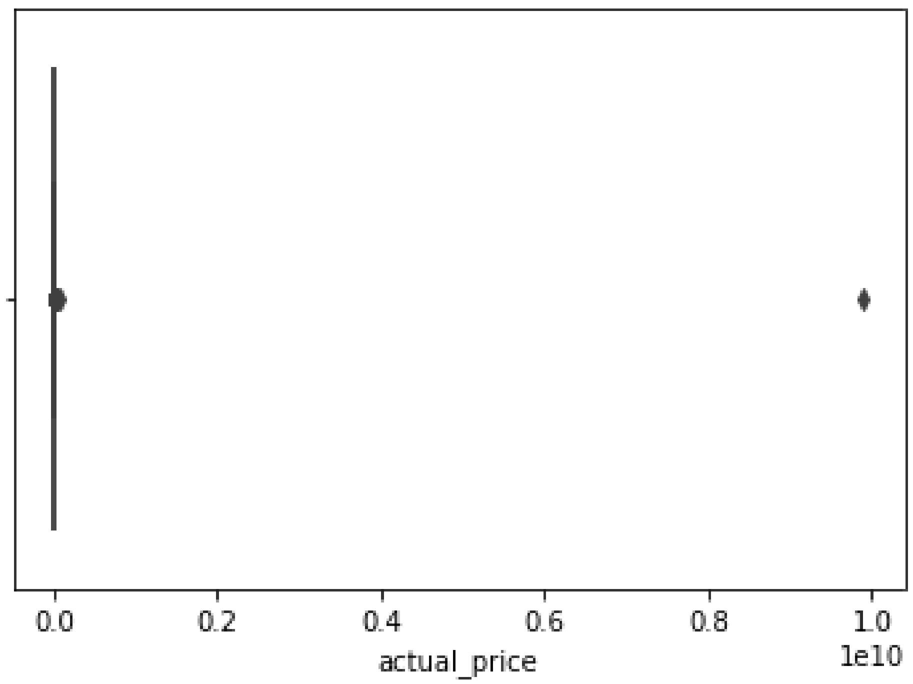
[272] :
```
data.nunique().sort_values(ascending=False).
 plot(kind='bar',figsize=(10,5),title = 'UNIQUE COUNTS')
```

[272] :
```
<Axes: title={'center': 'UNIQUE COUNTS'}>
```



## 15 OUTLIERS DETECTION

[273] :
```
sb.boxplot(data=data,x='actual_price')
plt.show()
```



[274] :
```
max(data['actual_price'])
```

[274]:
```
9899999999.0
```

[275] :
```
min(data['actual_price'])
```

[275]:
```
4.0
```

[276] :
```
x=    np.quantile(data['actual_price'],(0.25,0.75))
Q3=x[1]
Q1=x[0]
print("Q1: ",Q1)
print("Q3: ",Q3)
```

```
Q1:  999.0
Q3:  2999.0
```

[277] :
```
IQR  = Q3-Q1
uw= Q3+ 1.5*IQR
lw =Q1-1.5 * IQR
```

[278] :
```
data['actual_price']=np.where(data['actual_price']>uw,uw,data['actual_price'])
```

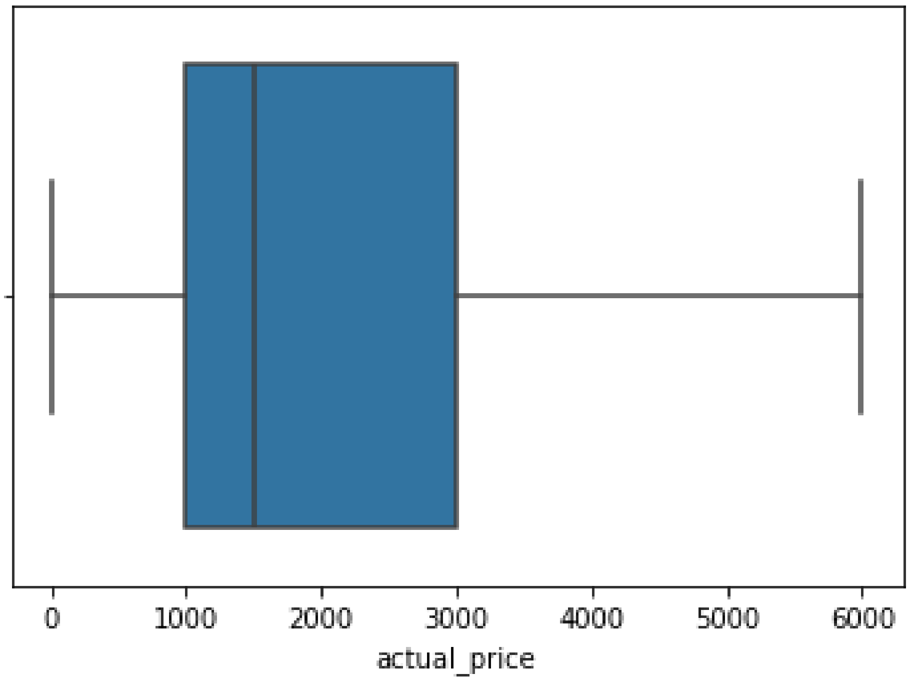
[279] :
```
data['actual_price']=np.where(data['actual_price']<lw,lw,data['actual_price'])
```

[280] :
```
max(data['actual_price'])
```

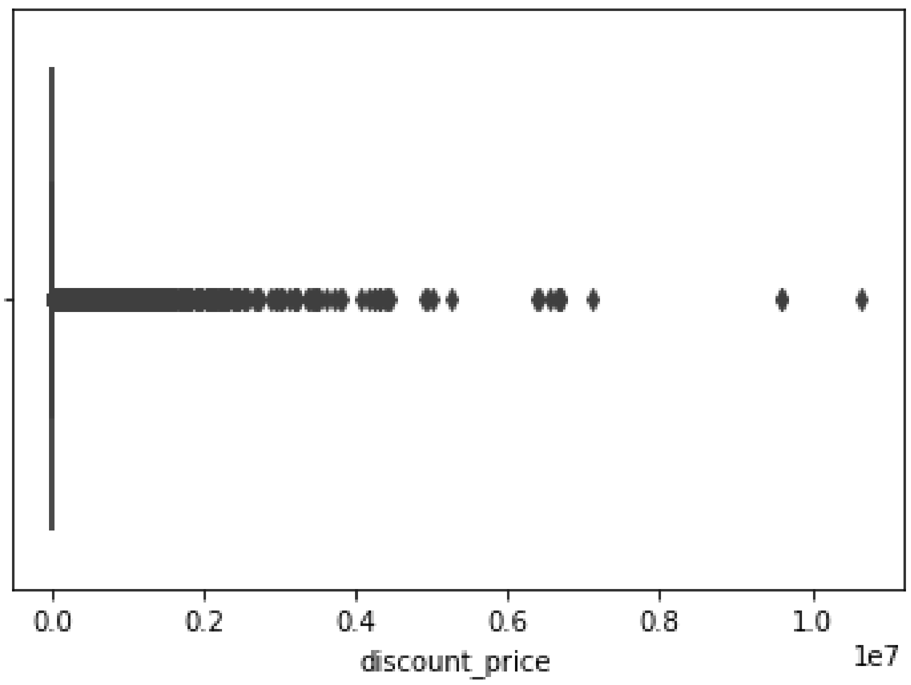
[280]:
```
5999.0
```

[281] :
```
sb.boxplot(data=data,x='actual_price')
```

[281] :
```
<Axes: xlabel='actual_price'>
```



[282] :
```
sb.boxplot(data=data,x='discount_price')
plt.show()
```



```
[283] : x=   np.quantile(data['discount_price'],(0.25,0.75))
        Q3=x[1]
        Q1=x[0]
        print("Q1: ",Q1)
        print("Q3: ",Q3)
        IQR  = Q3-Q1
        uw= Q3+ 1.5*IQR
        lw =Q1-1.5 * IQR
```

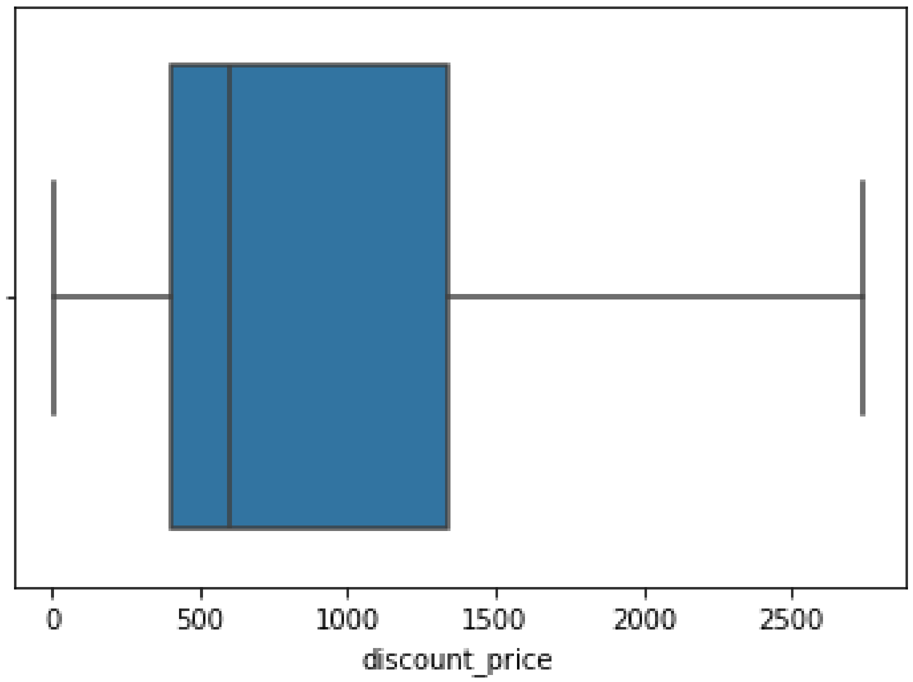
```
Q1:  405.0
Q3:  1340.0
```

```
[284] : data['discount_price']=np.
          ↳where(data['discount_price']>uw,uw,data['discount_price'])
```

```
[285] : data['discount_price']=np.
          ↳where(data['discount_price']<lw,lw,data['discount_price'])
```

```
[286] : sb.boxplot(data=data,x='discount_price')
```

```
[286] : <Axes: xlabel='discount_price'>
```



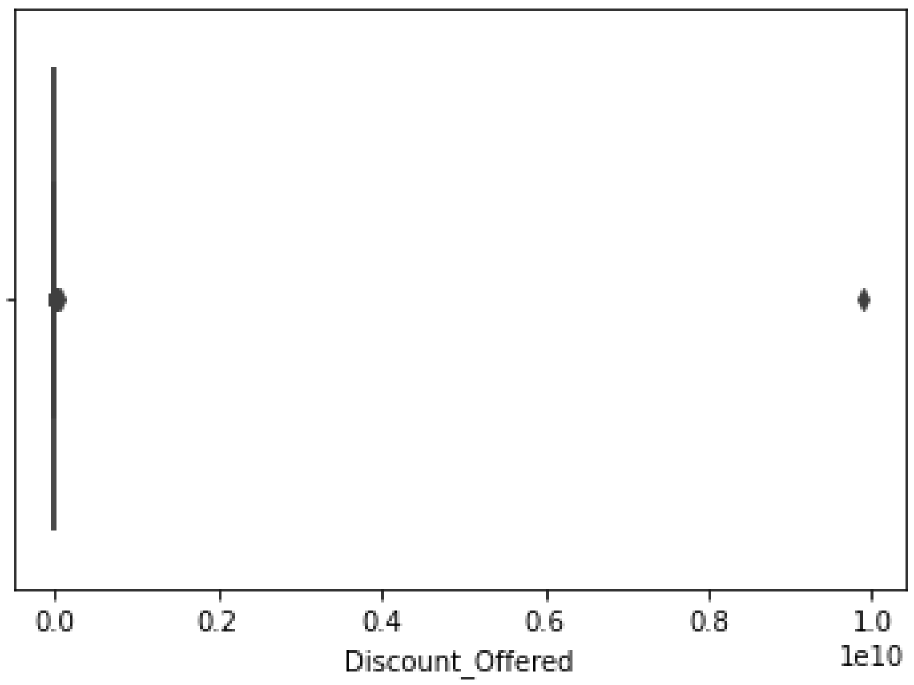
```
[287] : max(data['discount_price'])
```

```
[287]:  2742.5
```

```
[288] : min(data['discount_price'])
```

```
[288]:  8.0
```

```
[289] : sb.boxplot(data=data,x='Discount_Offered')
        plt.show()
```



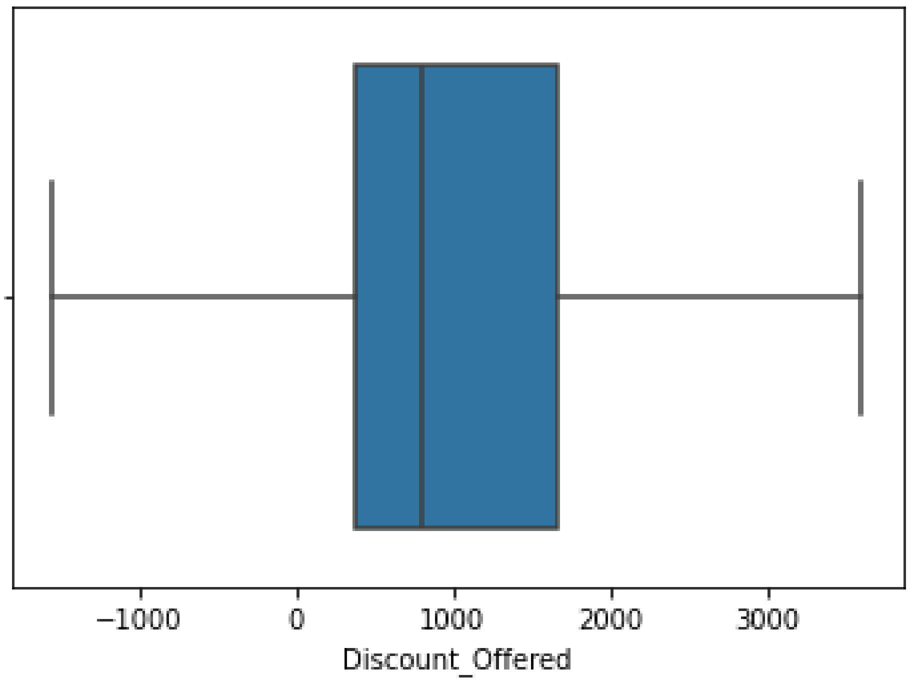
[290] :
```
x=  np.quantile(data['Discount_Offered'],(0.25,0.75))
Q3=x[1]
Q1=x[0]
print("Q1: ",Q1)
print("Q3: ",Q3)
IQR = Q3-Q1
uw= Q3+ 1.5*IQR
lw =Q1-1.5 * IQR
```

```
Q1:  375.0
Q3:  1665.0
```

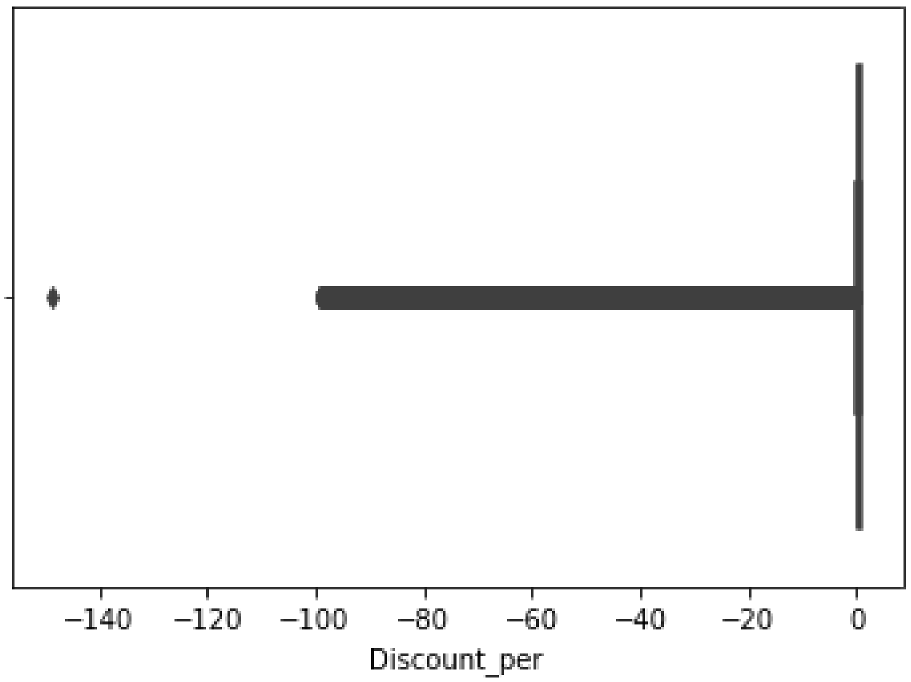
[291] :
```
data['Discount_Offered']=np.
    where(data['Discount_Offered']>uw,uw,data['Discount_Offered'])
```

[292] :
```
data['Discount_Offered']=np.
    where(data['Discount_Offered']<lw,lw,data['Discount_Offered'])
```

[293] :
```
sb.boxplot(data=data,x='Discount_Offered')
plt.show()
```



[294] :
```
sb.boxplot(data=data,x='Discount_per')
plt.show()
```



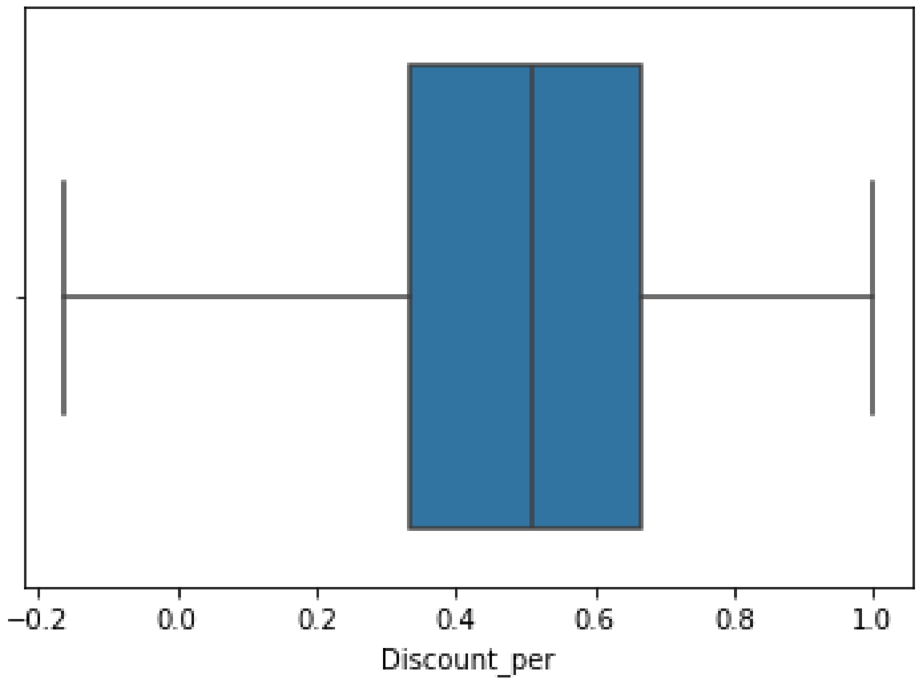
[295] :
```
x=  np.quantile(data['Discount_per'],(0.25,0.75))
Q3=x[1]
Q1=x[0]
print("Q1: ",Q1)
print("Q3: ",Q3)
IQR = Q3-Q1
uw= Q3+ 1.5*IQR
lw =Q1-1.5 * IQR
```

```
Q1:  0.3337041156840934
Q3:  0.6653326663331666
```

[296] :
```
data['Discount_per']=np.where(data['Discount_per']>uw,uw,data['Discount_per'])
```

[297] :
```
data['Discount_per']=np.where(data['Discount_per']<lw,lw,data['Discount_per'])
```

[298] :
```
sb.boxplot(data=data,x='Discount_per')
plt.show()
```



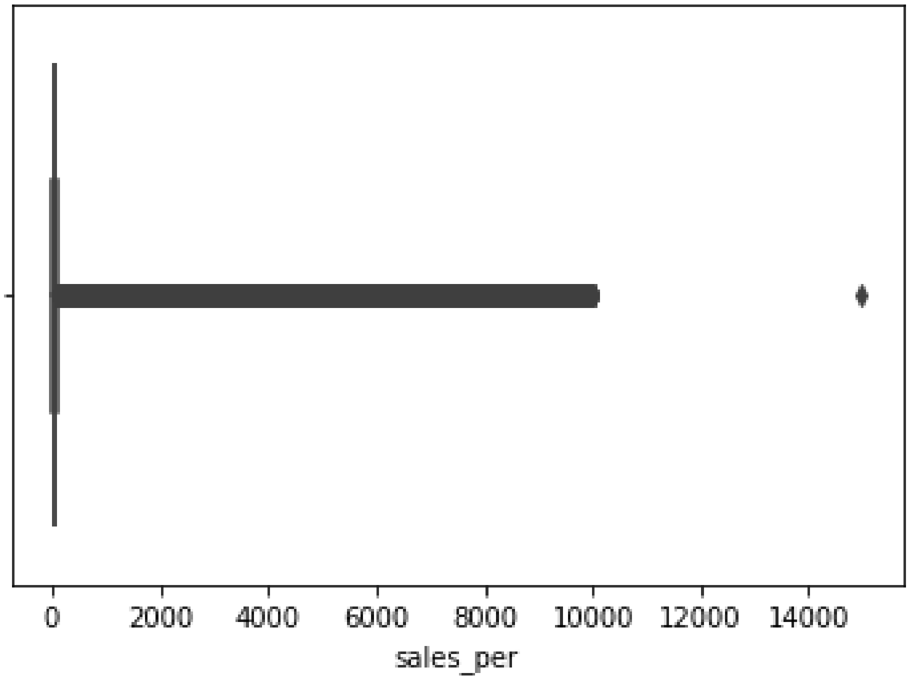
[96]:
```
data.columns
```

[96]:
```
Index(['name', 'Brand_Name', 'main_category', 'sub_category', 'actual_price',
       'discount_price', 'Discount_Offered', 'Discount_per', 'sales_per',
       'ratings', 'Rating_level', 'no_of_ratings'],
      dtype='object')
```

[299] :
```
data['sales_per'].unique()
```

[299] :
```
array([   55.94,    61.18,    55.64, ..., 7425.69, 3972.21, 2713.67])
```

[300] :
```
sb.boxplot(data=data,x='sales_per')
plt.show()
```

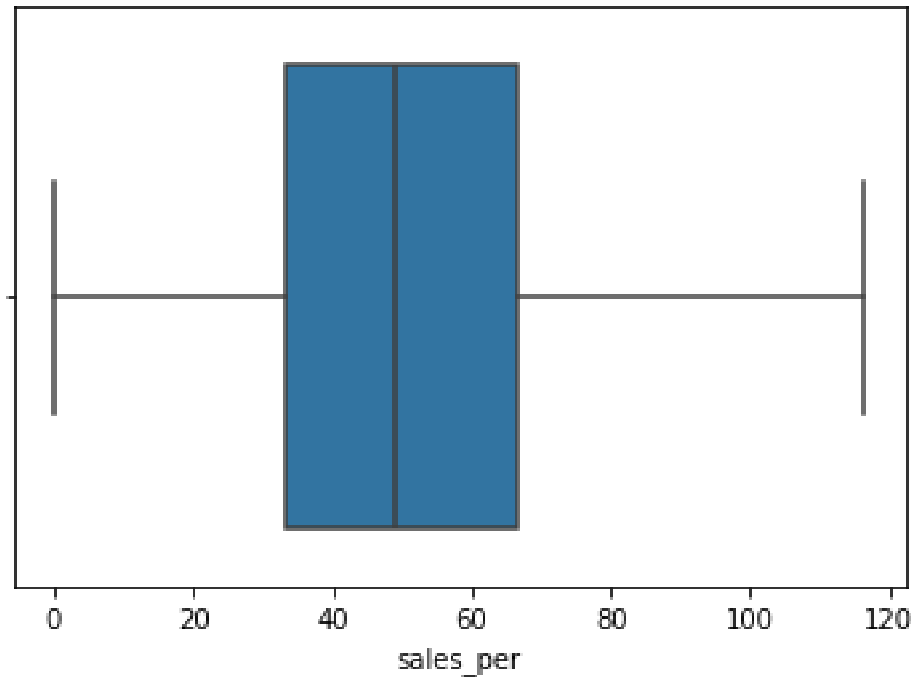


[301] :
```
x=  np.quantile(data['sales_per'],(0.25,0.75))
Q3=x[1]
Q1=x[0]
print("Q1: ",Q1)
print("Q3: ",Q3)
IQR = Q3-Q1
uw= Q3+ 1.5*IQR
lw =Q1-1.5 * IQR
```

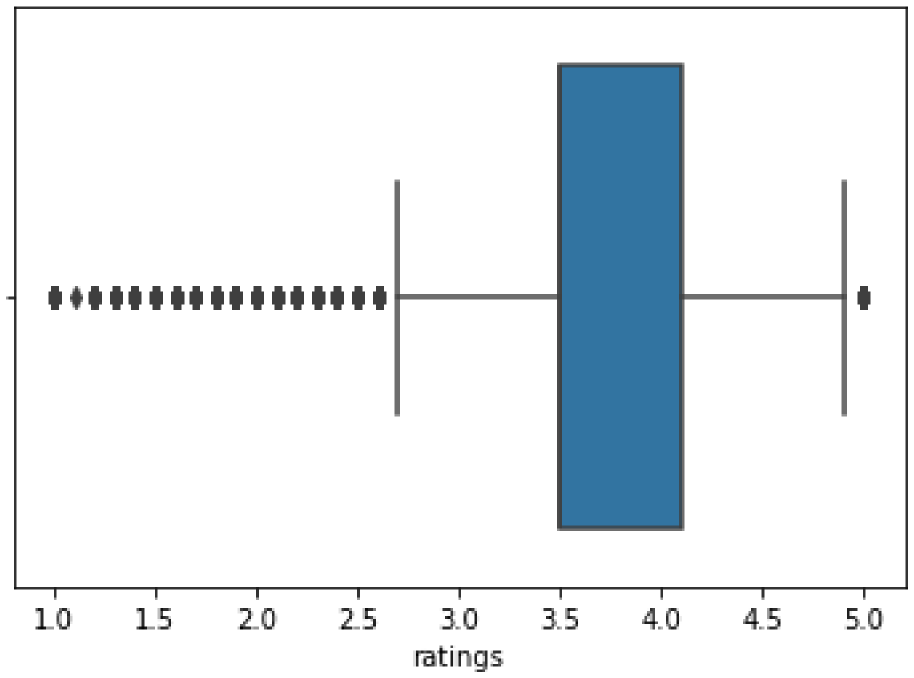
```
Q1:  33.47
Q3:  66.63
```

[302] :
```
data['sales_per']=np.where(data['sales_per']>uw,uw,data['sales_per'])
data['sales_per']=np.where(data['sales_per']<lw,lw,data['sales_per'])
```

[303] :
```
sb.boxplot(data=data,x='sales_per')
plt.show()
```



[304] :
```
sb.boxplot(data=data, x='ratings')
plt.show()
```

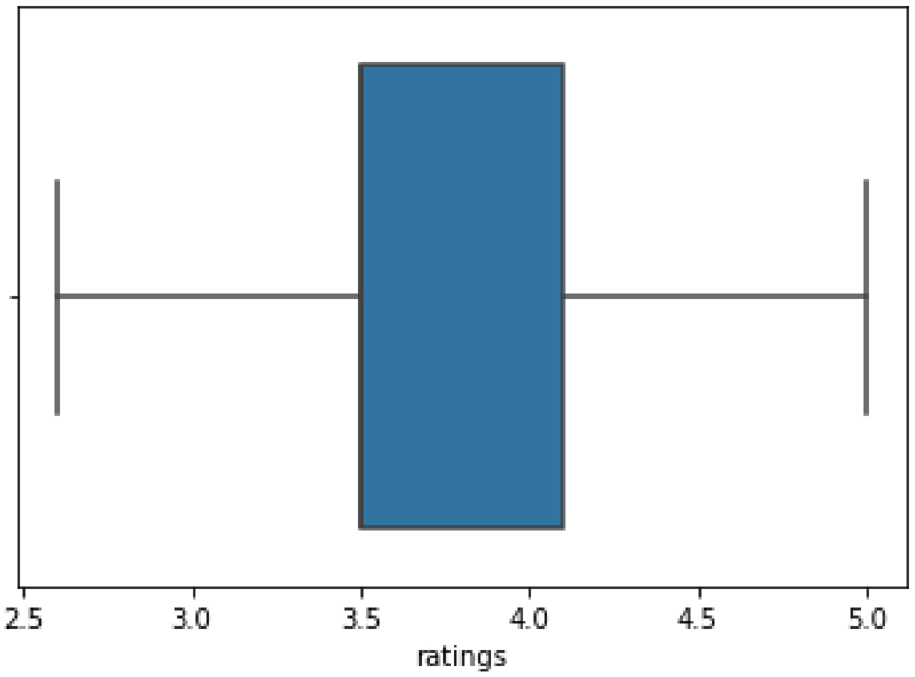


[305] :
```
x=   np.quantile(data['ratings'],(0.25,0.75))
Q3=x[1]
Q1=x[0]
print("Q1: ",Q1)
print("Q3: ",Q3)
IQR = Q3-Q1
uw= Q3+ 1.5*IQR
lw =Q1-1.5 * IQR
```

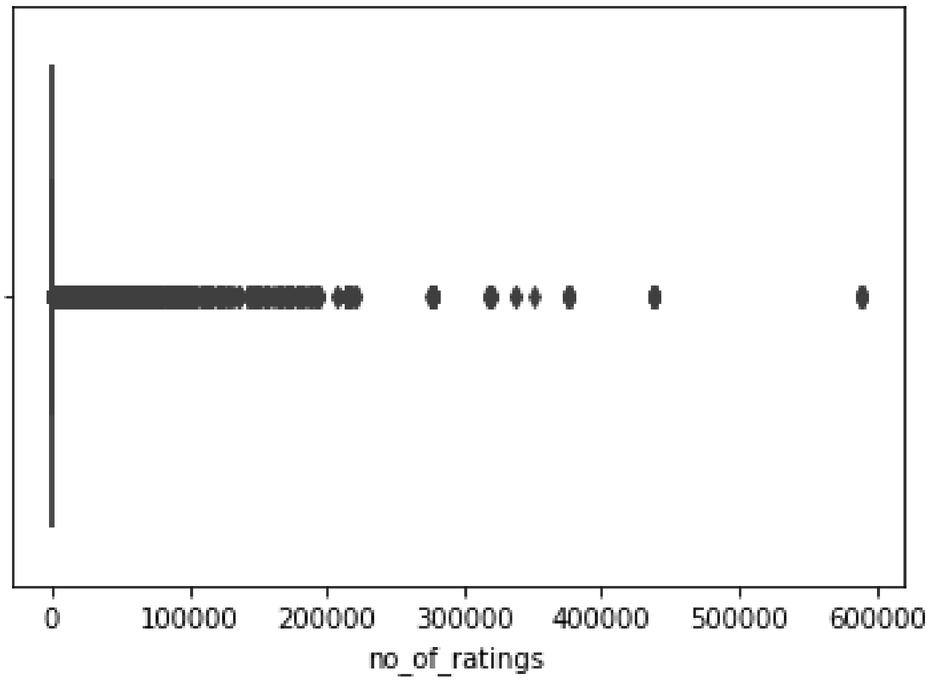
```
Q1:  3.5
Q3:  4.1
```

[306] :
```
data['ratings']=np.where(data['ratings']>uw,uw,data['ratings'])
data['ratings']=np.where(data['ratings']<lw,lw,data['ratings'])
```

[307] :
```
sb.boxplot(data=data, x='ratings')
plt.show()
```



[308] :
```
sb.boxplot(data=data, x='no_of_ratings')
plt.show()
```

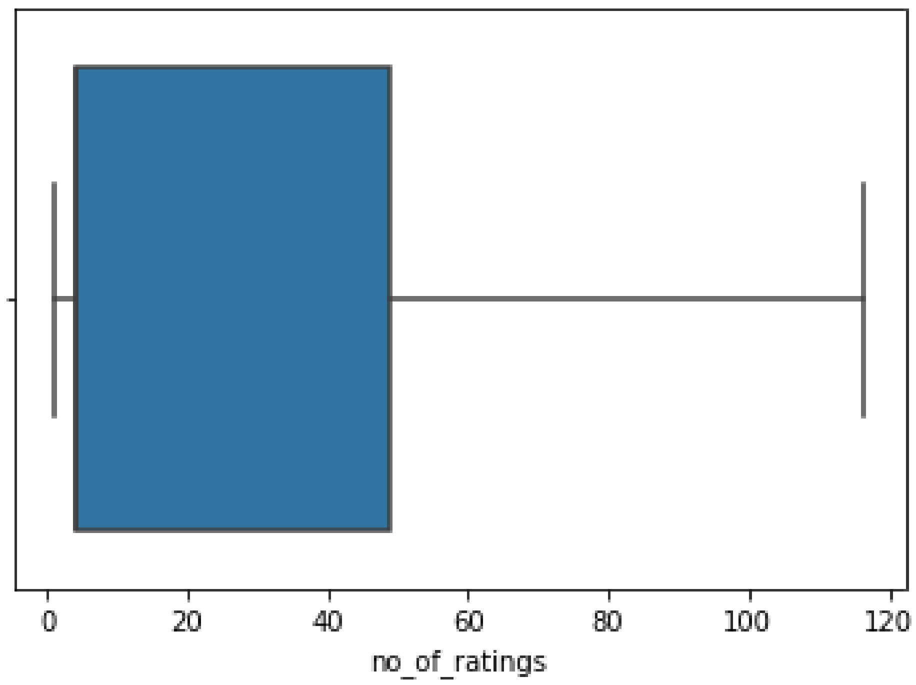


[309] :
```
x=  np.quantile(data['no_of_ratings'],(0.25,0.75))
Q3=x[1]
Q1=x[0]
print("Q1: ",Q1)
print("Q3: ",Q3)
IQR = Q3-Q1
uw= Q3+ 1.5*IQR
lw =Q1-1.5 * IQR
```

```
Q1:  4.0
Q3:  49.0
```

[310] :
```
data['no_of_ratings']=np.
  ↪where(data['no_of_ratings']>uw,uw,data['no_of_ratings'])
data['no_of_ratings']=np.
  ↪where(data['no_of_ratings']<lw,lw,data['no_of_ratings'])
```

[311] :
```
sb.boxplot(data=data, x='no_of_ratings')
plt.show()
```



[110]:
```
data.describe()
```

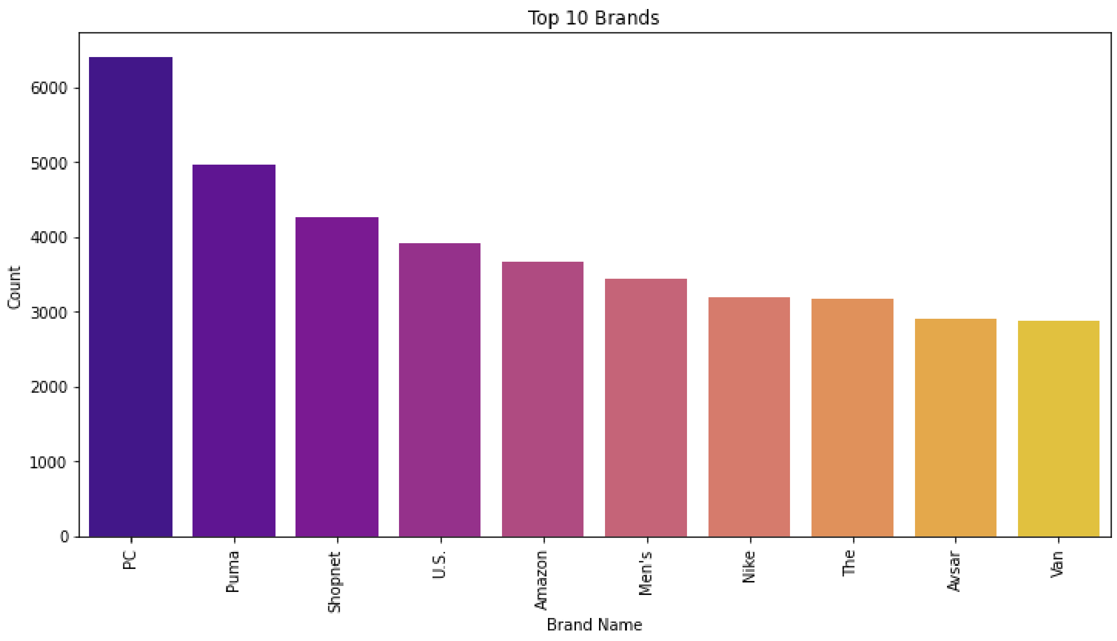
[110] :
```
             actual_price   discount_price  Discount_Offered   Discount_per
sales_per           ratings    no_of_ratings
count    551585.000000  551585.000000     551585.000000 551585.000000
551585.000000 551585.000000 551585.000000
mean       1848.812283      793.116512          941.544378          0.506078
49.393072          3.730468          18.680979
std          831.466278      375.898884          522.558493          0.133618
13.361010          0.262305          19.354328
min          999.000000      405.000000          375.000000          0.333704
33.470000          3.500000           4.000000
25%          999.000000      405.000000          375.000000          0.333704
33.470000          3.500000           4.000000
50%        1499.000000      599.000000          802.000000          0.510308
48.970000          3.500000           4.000000
75%        2999.000000     1340.000000         1665.000000          0.665333
66.630000          4.100000          49.000000
max        2999.000000     1340.000000         1665.000000          0.665333
66.630000          4.100000          49.000000
```

#

UNI-VARIENT ANALYSIS OF BRAND_NAME

### 15.1 What are the top 10 brands with respective to their product counts?

[111] :
```
brand_counts = data['Brand_Name'].value_counts()
top_brands = brand_counts.head(10)   # Display top 10 brands
plt.figure(figsize=(12, 6))
sb.barplot(x=top_brands.index, y=top_brands.values, palette='plasma')
plt.xticks(rotation=90)
plt.xlabel('Brand Name')
plt.ylabel('Count')
plt.title('Top 10 Brands')
plt.show()
```



- The given barchart demonstrates the top 10 brands which have high product count

---

#### 15.1.1 which company products are buying more?

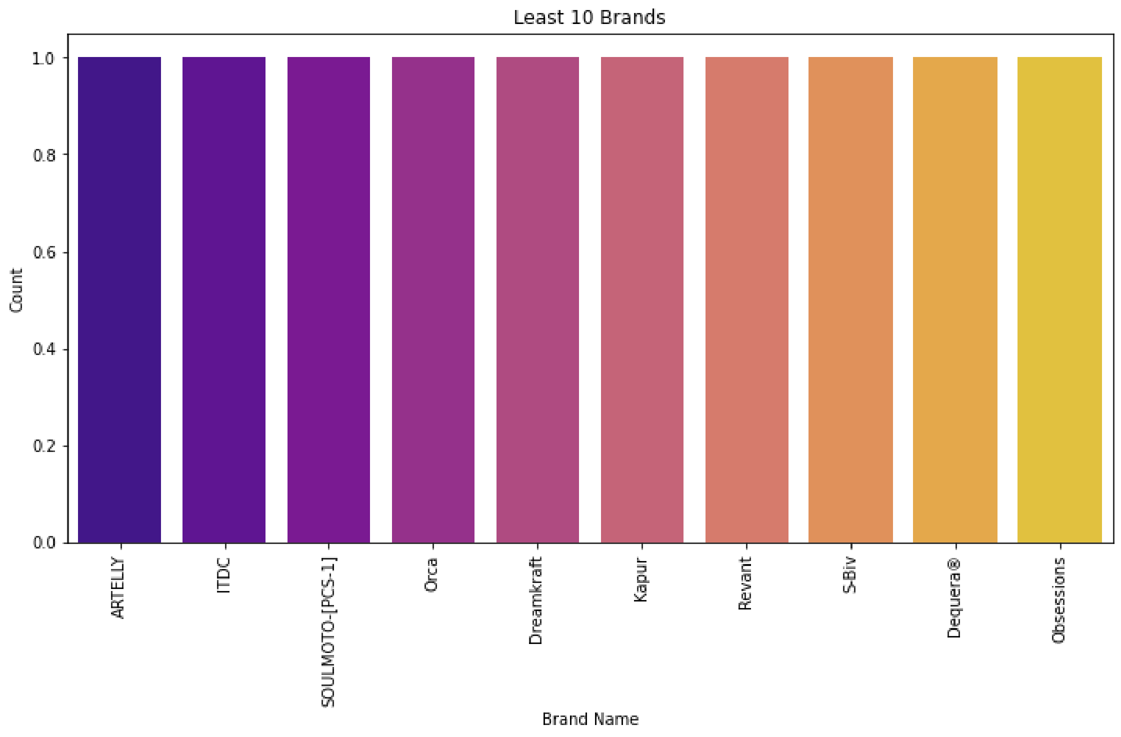
Ans:- PC Company products are buying more

---

### 15.2 What are the bottom 10 brands with respective to thier Product counts?

[112] :
```
brand_counts = data['Brand_Name'].value_counts()
low_brands = brand_counts.tail(10)     # Display top 10 brands
plt.figure(figsize=(12, 6))
sb.barplot(x=low_brands.index, y=low_brands.values, palette='plasma')
```

```
plt.xticks(rotation=90)
plt.xlabel('Brand Name')
plt.ylabel('Count')
plt.title('Least 10 Brands')
plt.show()
```



## 15.3 What are the main categories of the products in terms of their counts?

[113] :
```
data['main_category'].value_counts().reset_index()
```

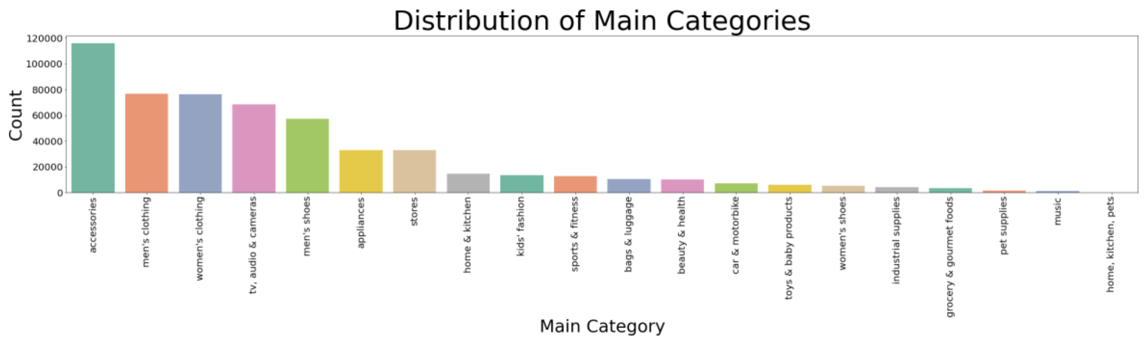
[113]:
```
          main_category   count
0           accessories  116141
1         men's clothing  76656
2       women's clothing  76512
3    tv, audio & cameras  68659
4            men's shoes  57456
5             appliances  33096
6                 stores  32903
7         home & kitchen  14568
8          kids' fashion  13488
9       sports & fitness  12648
10        bags & luggage  10416
11       beauty & health  10122
```

```
12            car & motorbike    7080
13       toys & baby products    6216
14              women's shoes    5472
15        industrial supplies    4104
16   grocery & gourmet foods    3312
17               pet supplies    1632
18                      music    1080
19        home, kitchen, pets      24
```

[114] :
```
data['main_category'].unique()
```

[114] :
```
array(['appliances', 'car & motorbike', 'tv, audio & cameras',
       'sports & fitness', 'grocery & gourmet foods', 'home & kitchen',
       'pet supplies', 'stores', 'toys & baby products', "kids' fashion",
       'bags & luggage', 'accessories', "women's shoes",
       'beauty & health', "men's shoes", "women's clothing",
       'industrial supplies', "men's clothing", 'music',
       'home, kitchen, pets'], dtype=object)
```

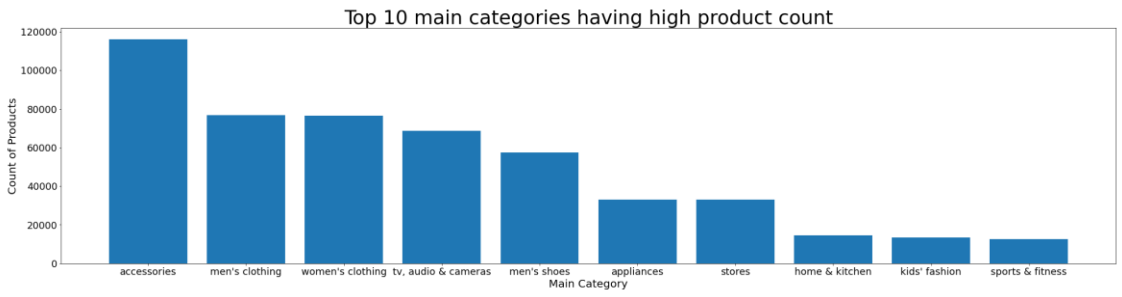
[115] :
```
# Column: 'main_category' - Main Product Category
# Univariate analysis for categorical data
category_counts = data['main_category'].value_counts()
plt.figure(figsize=(40, 6))
sb.barplot(x=category_counts.index, y=category_counts.values, palette='Set2')
plt.xticks(rotation=90,fontsize=20)
plt.yticks(fontsize=20)
plt.xlabel('Main Category',fontsize=36)
plt.ylabel('Count',fontsize=36)
plt.title('Distribution of Main Categories',fontsize=56)
plt.show()
```



[116] :
```
main_cat=data['main_category'].value_counts().reset_index().head(10)
main_cat
```

[116]:
```
        main_category   count
0         accessories  116141
1       men's clothing  76656
2     women's clothing  76512
3  tv, audio & cameras  68659
4          men's shoes  57456
5           appliances  33096
6               stores  32903
7        home & kitchen  14568
8         kids' fashion  13488
9      sports & fitness  12648
```

[117]:
```
plt.figure(figsize=(35,8))
plt.bar(main_cat['main_category'],main_cat['count'])
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.title("Top 10 main categories having high product count",fontsize=36)
plt.xlabel('Main Category', fontsize = 20 )
plt.ylabel('Count of Products',fontsize=20)
plt.show()
```



- Accessories have more no.of products in Main category

---

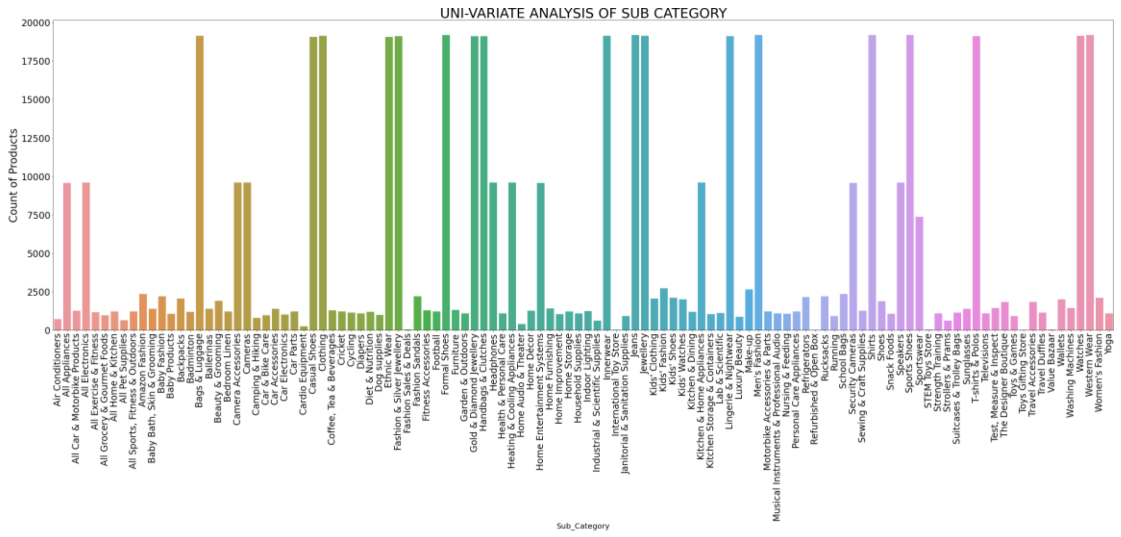
### 15.4 What are the sub-categories of the products with respect to their product count?

[118]:
```
data['sub_category'].value_counts().reset_index()
```

[118]:
```
     sub_category  count
0          Shirts  19200
1    Sports Shoes  19200
2           Jeans  19200
3    Western Wear  19200
4    Men's Fashion 19200
..            ...    ...
```

```
107             STEM Toys Store      48
108       Fashion Sales & Deals      44
109           Toys Gifting Store      24
110   International Toy Store      24
111     Refurbished & Open Box      24

[112 rows x 2 columns]
```

[119]:
```
plt.figure(figsize=(50,15))
sb.countplot(data=data,x='sub_category')
plt.xticks(rotation=90,fontsize=24)
plt.yticks(fontsize=24)
plt.xlabel('Sub_Category',fontsize=20)
plt.ylabel('Count of Products', fontsize=28)
plt.title("UNI-VARIATE ANALYSIS OF SUB CATEGORY",fontsize=36)
plt.show()
```



### 15.4.1 Top 10 sub categories

[120]:
```
sub_cat=data['sub_category'].value_counts().reset_index().head(10)
sub_cat
```

[120]:
```
     sub_category  count
0          Shirts  19200
1    Sports Shoes  19200
2           Jeans  19200
3    Western Wear  19200
4   Men's Fashion  19200
5    Formal Shoes  19200
```

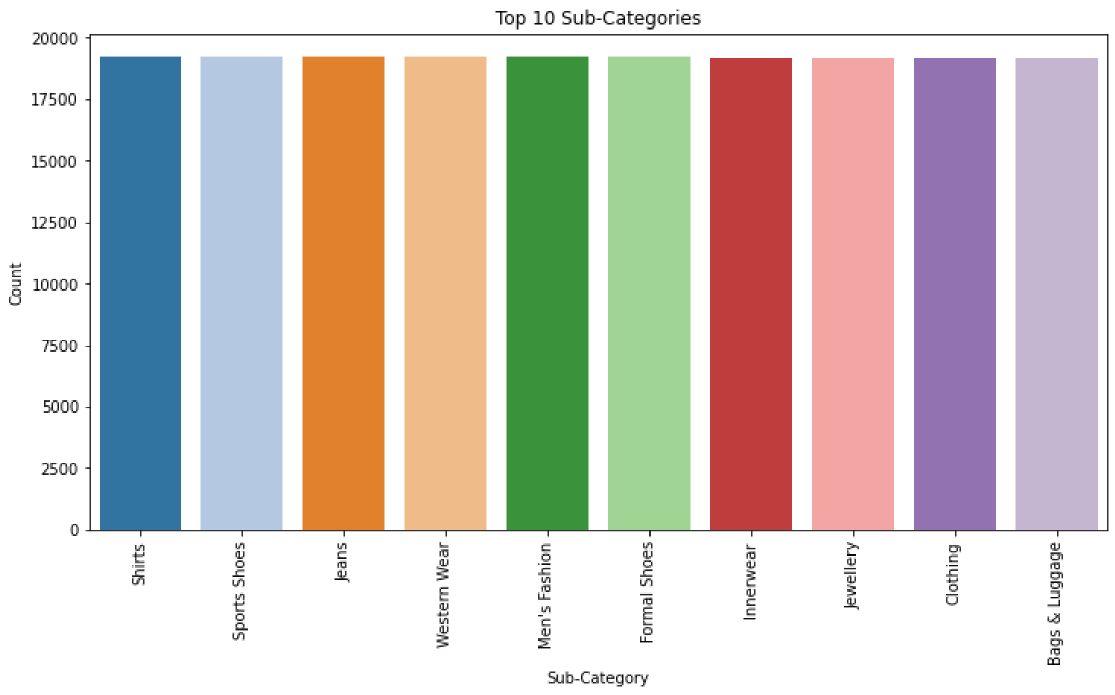
```
6        Innerwear  19152
7        Jewellery  19152
8         Clothing  19152
9   Bags & Luggage  19152
```

[121]:
```
sub_cat.sub_category.values
```

[121] :
```
array(['Shirts', 'Sports Shoes', 'Jeans', 'Western Wear', "Men's Fashion",
       'Formal Shoes', 'Innerwear', 'Jewellery', 'Clothing',
       'Bags & Luggage'], dtype=object)
```

### 15.5 What are the top 10 sub categories with respect to product count?

[122] :
```
# Column: 'sub_category' - Sub-Category
# Univariate analysis for categorical data
sub_category_counts = data['sub_category'].value_counts()
top_sub_categories = sub_category_counts.head(10) # Display top 10␣
 ↪sub-categories
plt.figure(figsize=(12, 6))
sb.barplot(x=top_sub_categories.index, y=top_sub_categories.values,␣
 ↪palette='tab20')
plt.xticks(rotation=90)
plt.xlabel('Sub-Category')
plt.ylabel('Count')
plt.title('Top 10 Sub-Categories')
plt.show()
```



## 15.6

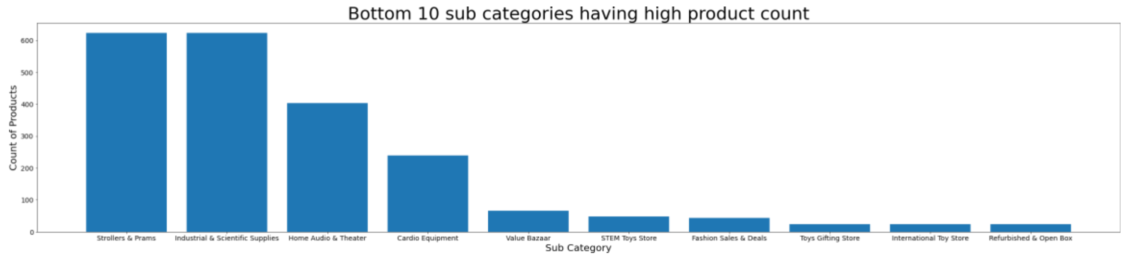
### 15.6.1 Bottom 10 sub categories

[123] :
```
sub_cat2=data['sub_category'].value_counts().reset_index().tail(10)
sub_cat2
```

[123]:
```
                          sub_category  count
102                   Strollers & Prams    624
103  Industrial & Scientific Supplies    624
104               Home Audio & Theater    403
105                   Cardio Equipment    240
106                        Value Bazaar     66
107                     STEM Toys Store     48
108                Fashion Sales & Deals     44
109                  Toys Gifting Store     24
110             International Toy Store     24
111            Refurbished & Open Box     24
```

## 15.7 What are the bottom 10 sub categories with respect to product count?

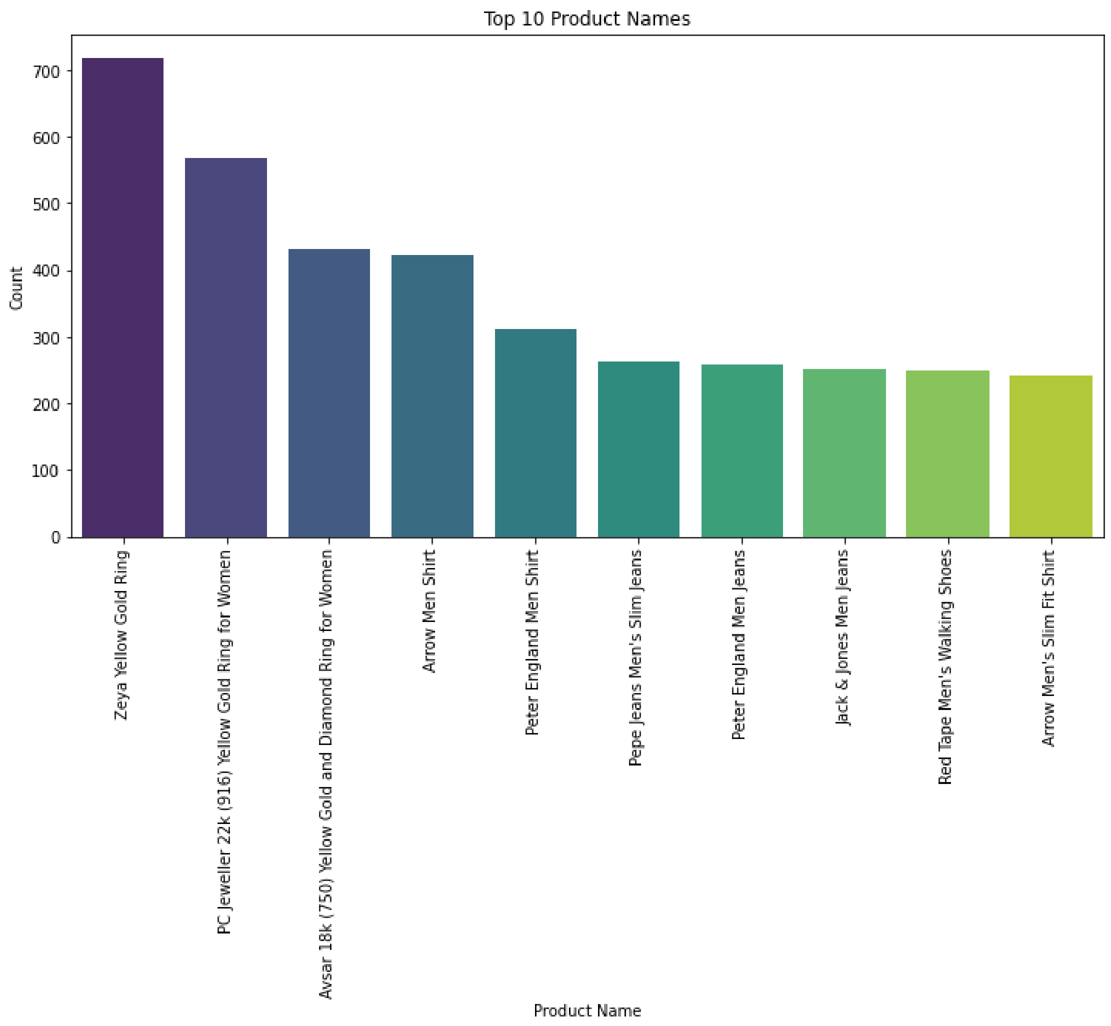
[124] :
```
plt.figure(figsize=(40,8))
plt.bar(sub_cat2['sub_category'],sub_cat2['count'])
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Bottom 10 sub categories having high product count",fontsize=36)
plt.xlabel('Sub Category', fontsize = 20 )
plt.ylabel('Count of Products',fontsize=20)
plt.show()
```



[125] :
```
data.columns
```

```
[125] : Index(['name', 'Brand_Name', 'main_category', 'sub_category', 'actual_price',
               'discount_price', 'Discount_Offered', 'Discount_per', 'sales_per',
               'ratings', 'Rating_level', 'no_of_ratings'],
              dtype='object')
```

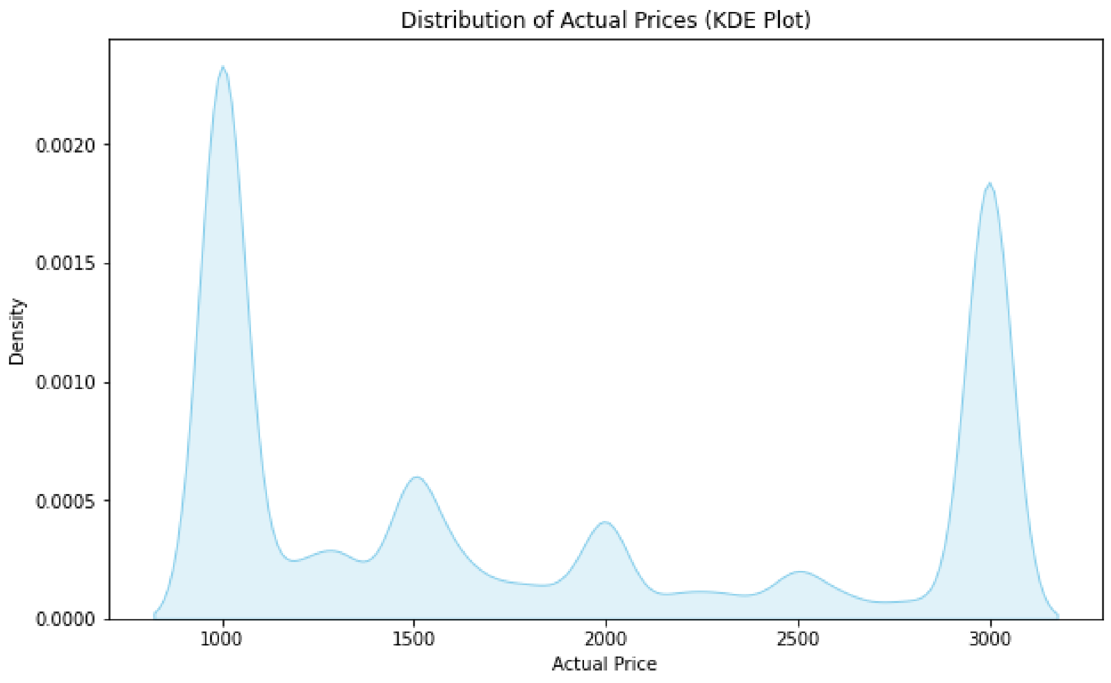
```
[126] : name_counts = data['name'].value_counts()
        top_names = name_counts.head(10)    # Display top 10 product names
        plt.figure(figsize=(12, 6))
        sb.barplot(x=top_names.index, y=top_names.values, palette='viridis')
        plt.xticks(rotation=90)
        plt.xlabel('Product Name')
        plt.ylabel('Count')
        plt.title('Top 10 Product Names')
        plt.show()
```



[127] :
```
top_names.index
```

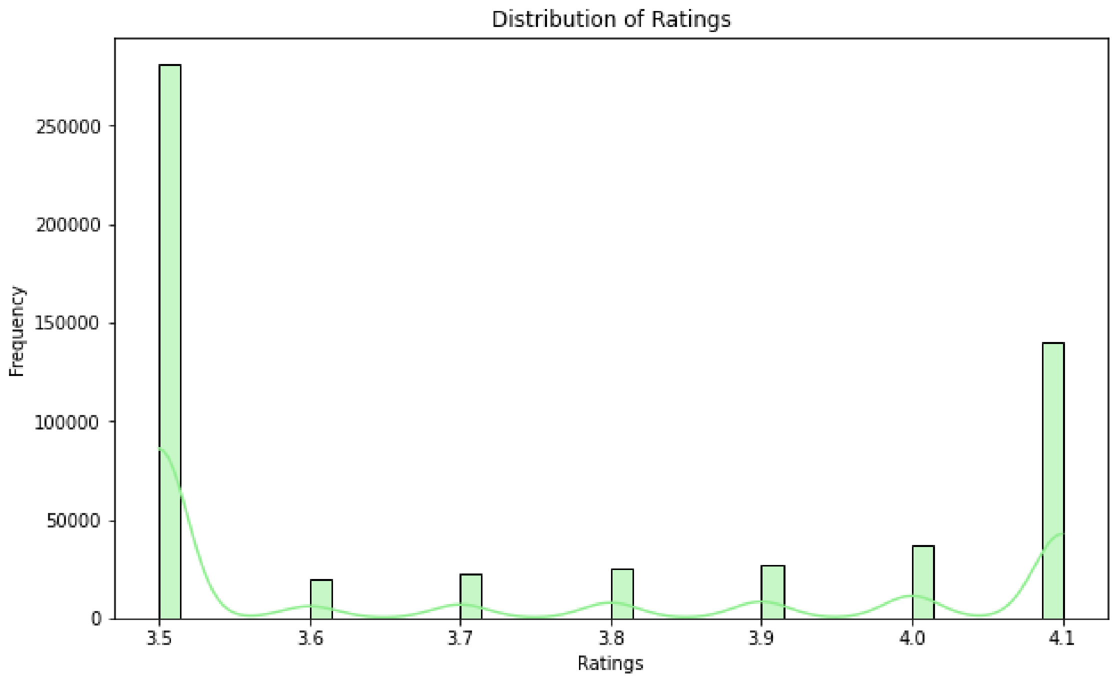
[127] :
```
Index(['Zeya Yellow Gold Ring',
       'PC Jeweller 22k (916) Yellow Gold Ring for Women',
       'Avsar 18k (750) Yellow Gold and Diamond Ring for Women',
       'Arrow Men Shirt', 'Peter England Men Shirt',
       'Pepe Jeans Men's Slim Jeans', 'Peter England Men Jeans',
       'Jack & Jones Men Jeans', 'Red Tape Men's Walking Shoes',
       'Arrow Men's Slim Fit Shirt'],
      dtype='object', name='name')
```

[128] :
```
# Column: 'actual_price' - Actual Price
# Univariate analysis for numerical data (KDE plot)
plt.figure(figsize=(10, 6))
sb.kdeplot(data['actual_price'], color='skyblue', fill=True)
plt.xlabel('Actual Price')
plt.ylabel('Density')
plt.title('Distribution of Actual Prices (KDE Plot)')
plt.show()
```



The KDE plot of the 'actual_price' column shows that the distribution of actual prices appears to be bimodal, with two peaks around 1000 and 2500. This suggests that there are two groups in the data with different price ranges. The density is highest at these two peaks, indicating that most of the prices are around these values. The density decreases as the price increases or decreases from these values, indicating fewer data points in those ranges. This could imply that items priced around 1000 and 2500 are more common.

```
[129]: # Column: 'ratings' - Ratings (Numerical)
       # Univariate analysis for numerical data (histogram)
       plt.figure(figsize=(10, 6))
       sb.histplot(data['ratings'], kde=True, color='lightgreen')
       plt.xlabel('Ratings')
       plt.ylabel('Frequency')
       plt.title('Distribution of Ratings')
       plt.show()
```

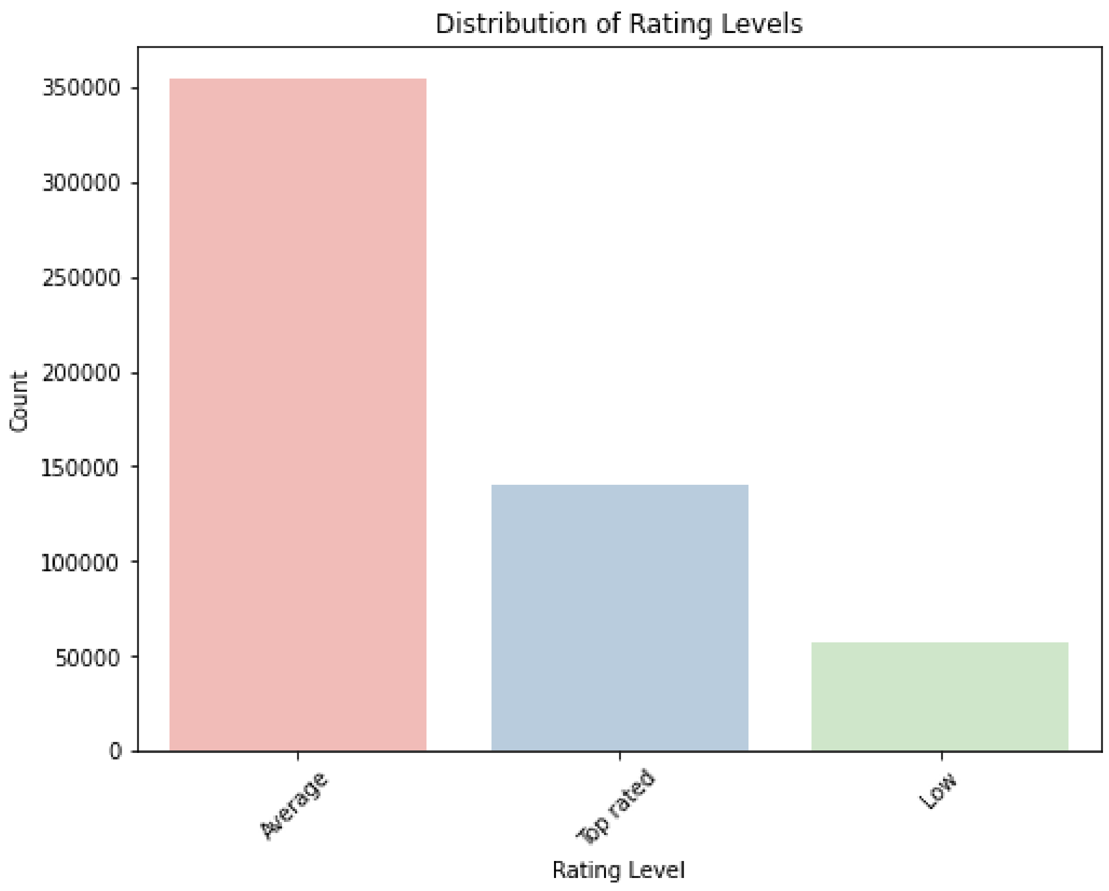


The histogram of the ratings column apperas that distribution be bimodal

```
[130]: data['ratings'].unique()
```

```
[130]: array([4.1, 4. , 3.9, 3.8, 3.5, 3.7, 3.6])
```

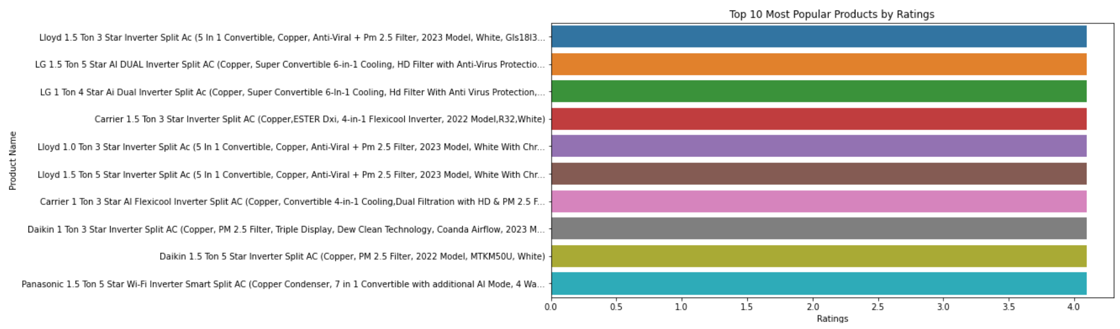
```
[131]: # Column: 'Rating_level' - Rating Level (Categorical)
       # Univariate analysis for categorical data

       rating_level_counts = data['Rating_level'].value_counts()
       plt.figure(figsize=(8, 6))
       sb.barplot(x=rating_level_counts.index, y=rating_level_counts.values,␣
        ↪palette='Pastel1')
       plt.xticks(rotation=45)
       plt.xlabel('Rating Level')
plt.ylabel('Count')
plt.title('Distribution of Rating Levels')
plt.show()
```



## 16 BI-VARIATE ANALYSIS

Most Popular Products:

[132] :
```
top_rated_products = data.nlargest(10, 'ratings')
plt.figure(figsize=(12, 6))
sb.barplot(data=top_rated_products, x='ratings', y='name')
plt.title('Top 10 Most Popular Products by Ratings')
plt.xlabel('Ratings')
plt.ylabel('Product Name')
plt.show()
```

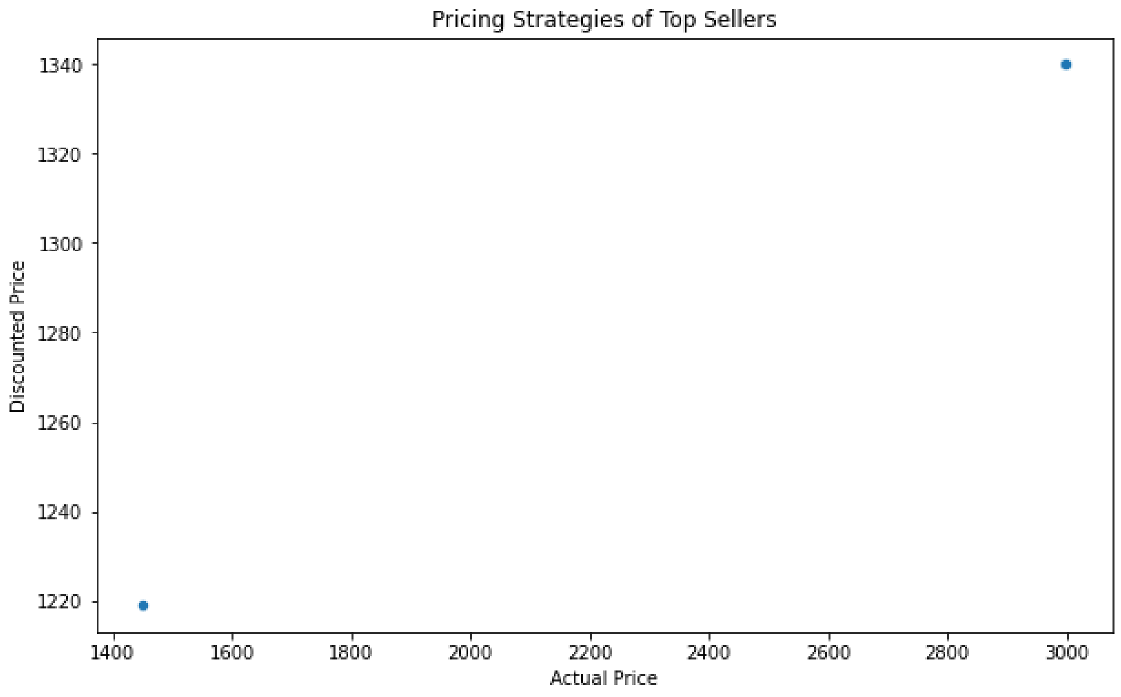


[133] :
```
top_rated_products.name.values
```

[133] :
```
array(['Lloyd 1.5 Ton 3 Star Inverter Split Ac (5 In 1 Convertible, Copper,
 Anti-Viral + Pm 2.5 Filter, 2023 Model, White, Gls18I3…',
        'LG 1.5 Ton 5 Star AI DUAL Inverter Split AC (Copper, Super Convertible
 6-in-1 Cooling, HD Filter with Anti-Virus Protectio…',
        'LG 1 Ton 4 Star Ai Dual Inverter Split Ac (Copper, Super Convertible
 6-In-1 Cooling, Hd Filter With Anti Virus Protection,…',
        'Carrier 1.5 Ton 3 Star Inverter Split AC (Copper,ESTER Dxi, 4-in-1
 Flexicool Inverter, 2022 Model,R32,White)',
        'Lloyd 1.0 Ton 3 Star Inverter Split Ac (5 In 1 Convertible, Copper,
 Anti-Viral + Pm 2.5 Filter, 2023 Model, White With Chr…',
        'Lloyd 1.5 Ton 5 Star Inverter Split Ac (5 In 1 Convertible, Copper,
 Anti-Viral + Pm 2.5 Filter, 2023 Model, White With Chr…',
        'Carrier 1 Ton 3 Star AI Flexicool Inverter Split AC (Copper, Convertible
 4-in-1 Cooling,Dual Filtration with HD & PM 2.5 F…',
        'Daikin 1 Ton 3 Star Inverter Split AC (Copper, PM 2.5 Filter, Triple
 Display, Dew Clean Technology, Coanda Airflow, 2023 M…',
        'Daikin 1.5 Ton 5 Star Inverter Split AC (Copper, PM 2.5 Filter, 2022
 Model, MTKM50U, White)',
        'Panasonic 1.5 Ton 5 Star Wi-Fi Inverter Smart Split AC (Copper
 Condenser, 7 in 1 Convertible with additional AI Mode, 4 Wa…'],
       dtype=object)
```

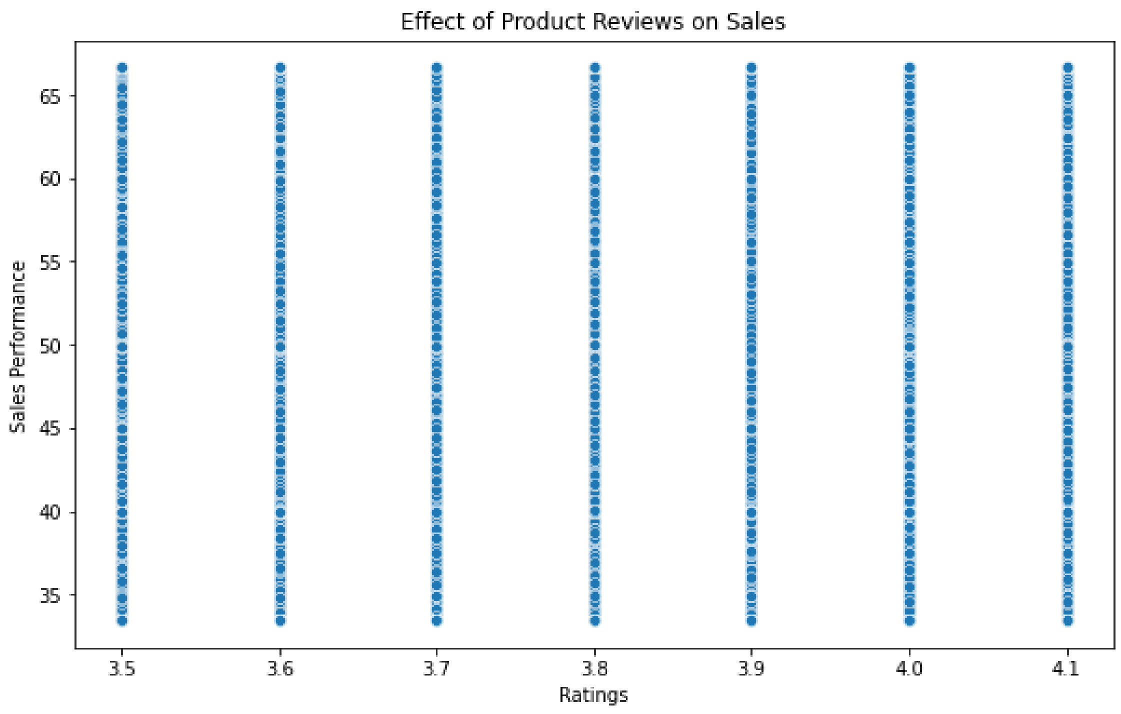
---

[134] :
```
# Create a scatter plot to compare actual_price and discount_price for␣
 ↪top-selling products
top_selling_products = data.nlargest(100, 'sales_per')   # Assuming you consider␣
 ↪the top 100 products as top sellers
plt.figure(figsize=(10, 6))
sb.scatterplot(data=top_selling_products, x='actual_price', y='discount_price')
plt.title('Pricing Strategies of Top Sellers')
plt.xlabel('Actual Price')
plt.ylabel('Discounted Price')
plt.show()
# Explain the pricing strategies you observe among top sellers.
```



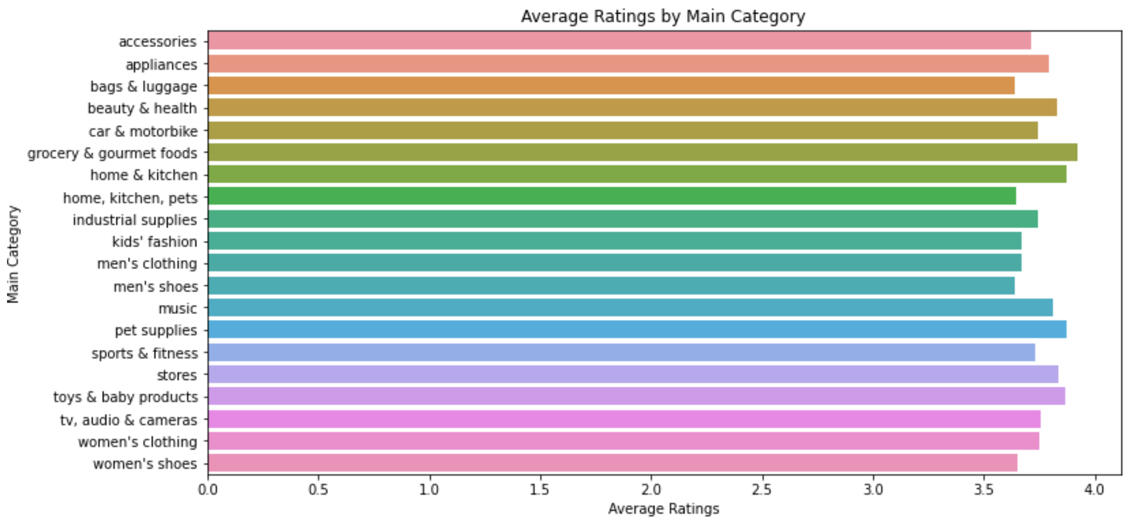
---

[135] :
```
# Create a scatter plot to analyze the correlation between ratings and sales_per
plt.figure(figsize=(10, 6))
sb.scatterplot(data=data, x='ratings', y='sales_per')
plt.title('Effect of Product Reviews on Sales')
plt.xlabel('Ratings')
plt.ylabel('Sales Performance')
plt.show()
# Explain the relationship between ratings and sales performance.
```

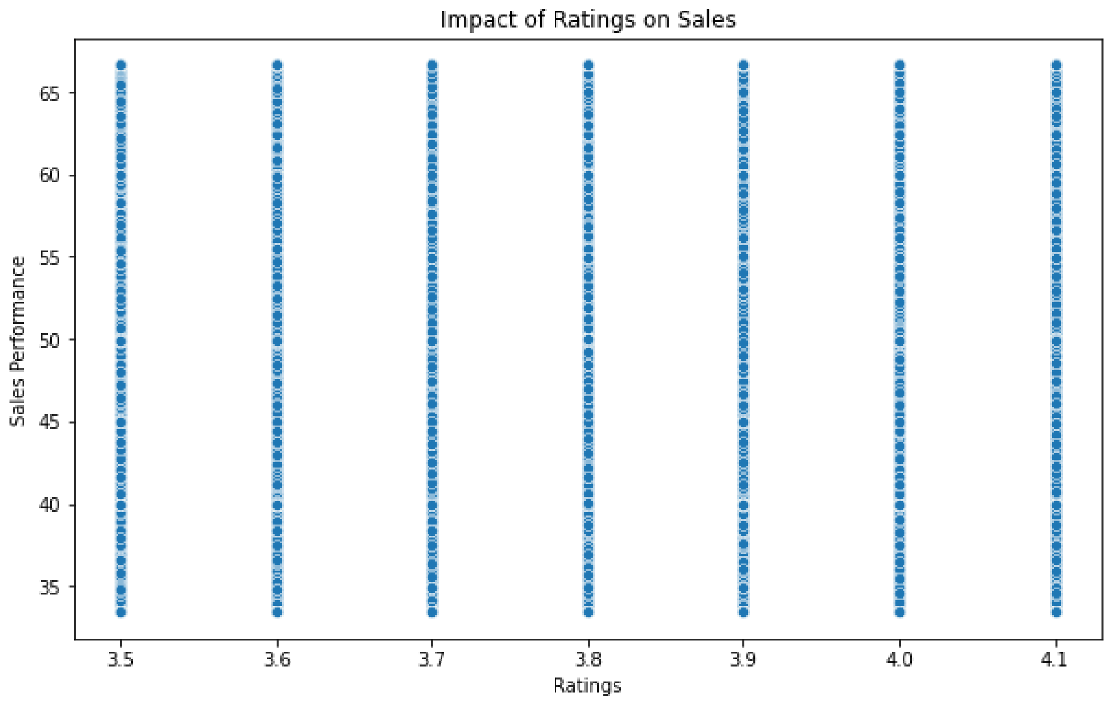


---

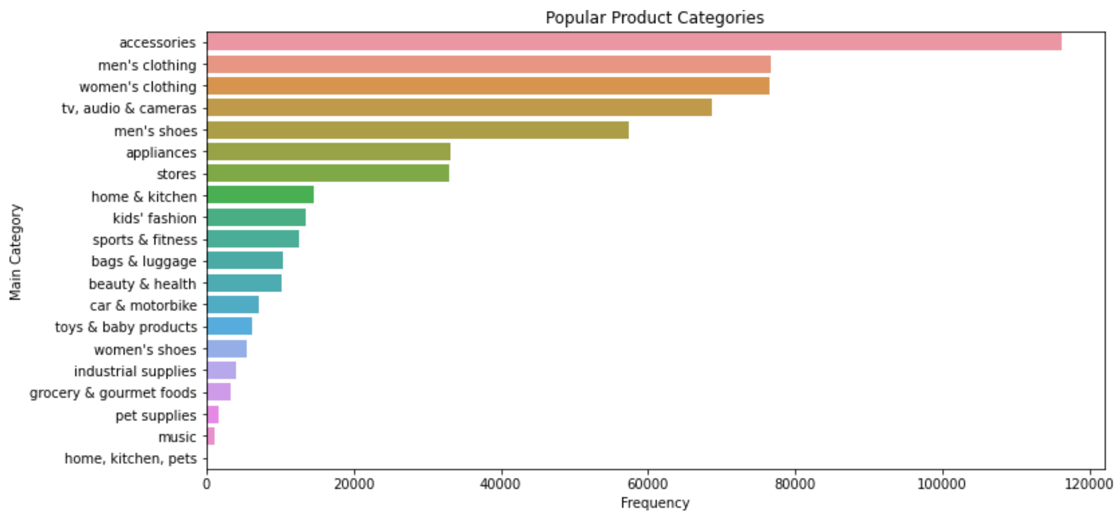
[136]:
```
# Create a bar chart to show the average ratings by main_category
avg_ratings_by_category = data.groupby('main_category')['ratings'].mean().
 ↪reset_index()
plt.figure(figsize=(12, 6))
sb.barplot(data=avg_ratings_by_category, x='ratings', y='main_category')
plt.title('Average Ratings by Main Category')
plt.xlabel('Average Ratings')
plt.ylabel('Main Category')
plt.show()
# Explain which main categories have high average ratings.
```



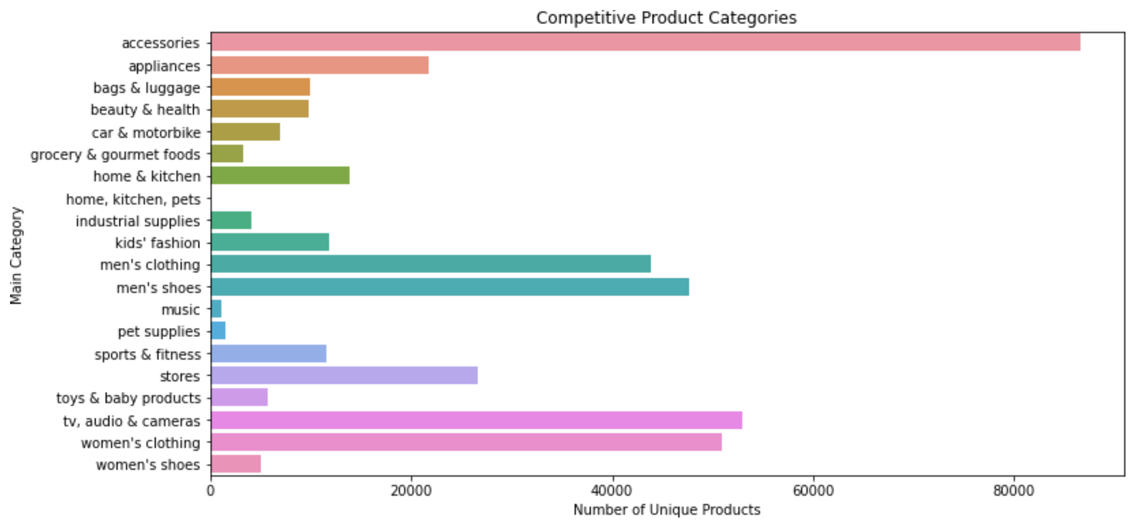
[137] :
```
# Create a scatter plot to analyze the correlation between ratings and sales_per
plt.figure(figsize=(10, 6))
sb.scatterplot(data=data, x='ratings', y='sales_per')
plt.title('Impact of Ratings on Sales')
plt.xlabel('Ratings')
plt.ylabel('Sales Performance')
plt.show()
# Explain how ratings impact the sales performance of products.
```



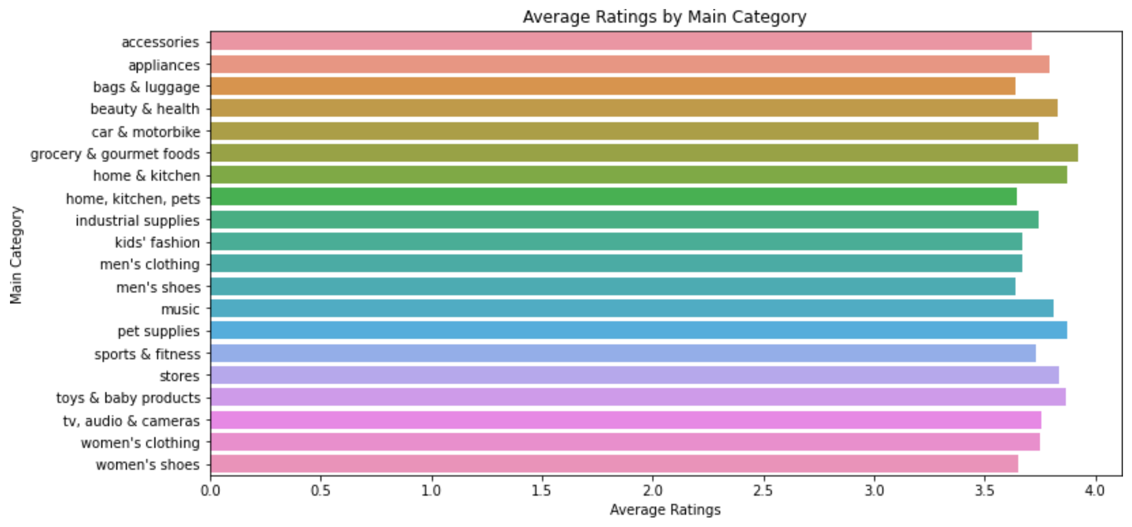
[138] :
```
# Create a count plot to show the frequency of each main_category
plt.figure(figsize=(12, 6))
sb.countplot(data=data, y='main_category', order=data['main_category'].
    ↪value_counts().index)
plt.title('Popular Product Categories')
plt.xlabel('Frequency')
plt.ylabel('Main Category')
plt.show()
# Explain which main categories are the most popular based on frequency.
```



[139] :
```
# Create a count plot to show the number of unique products or sellers within␣
    ↪each main_category
unique_products_per_category = data.groupby('main_category')['name'].nunique().
    ↪reset_index()
plt.figure(figsize=(12, 6))
sb.barplot(data=unique_products_per_category, x='name', y='main_category')
plt.title('Competitive Product Categories')
plt.xlabel('Number of Unique Products')
plt.ylabel('Main Category')
plt.show()
# Explain which main categories are the most competitive based on the number of␣
    ↪unique products.
```



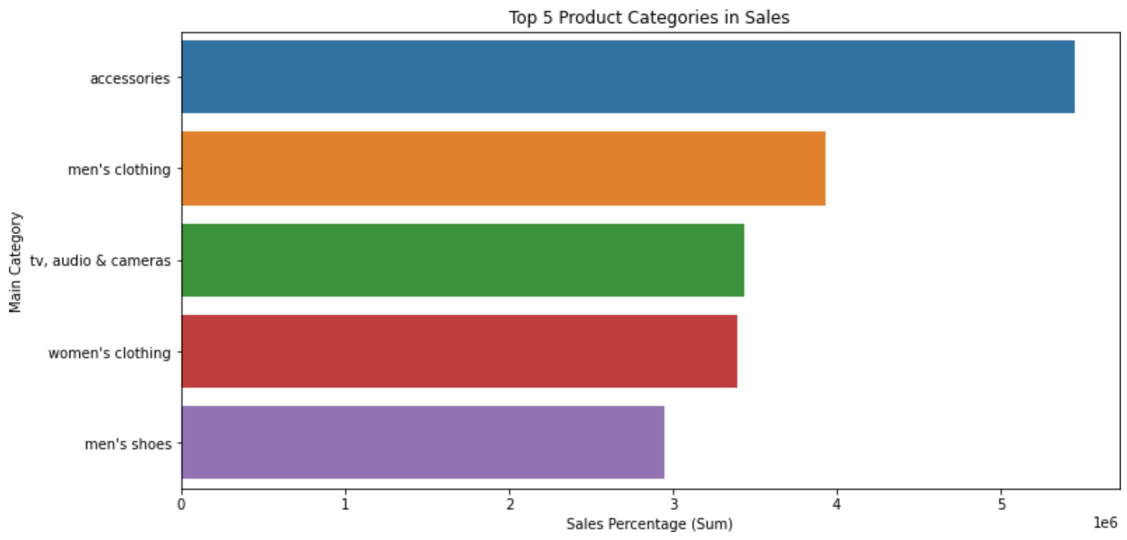
[140] :
```
# Create a bar chart to show the average ratings by main_category
avg_ratings_by_category  =  data.groupby('main_category')['ratings'].mean().
  ↪reset_index()
plt.figure(figsize=(12, 6))
sb.barplot(data=avg_ratings_by_category, x='ratings', y='main_category')
plt.title('Average Ratings by Main Category')
plt.xlabel('Average Ratings')
plt.ylabel('Main Category')
plt.show()
# Explain the relationship between average ratings and main categories.
```



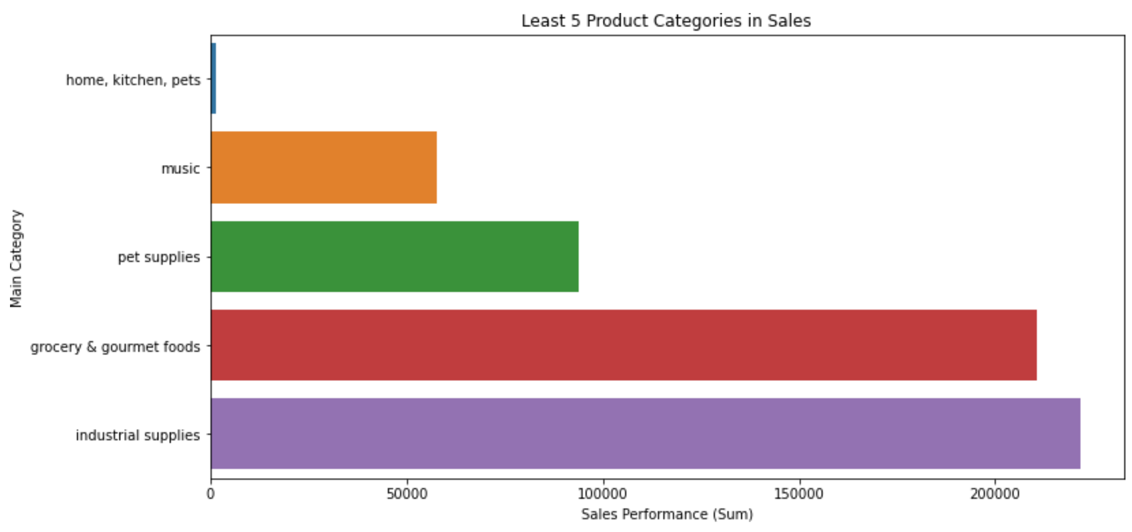
[141] :
```
data.columns
```

```
[141] : Index(['name', 'Brand_Name', 'main_category', 'sub_category', 'actual_price',
               'discount_price', 'Discount_Offered', 'Discount_per', 'sales_per',
               'ratings', 'Rating_level', 'no_of_ratings'],
              dtype='object')
```

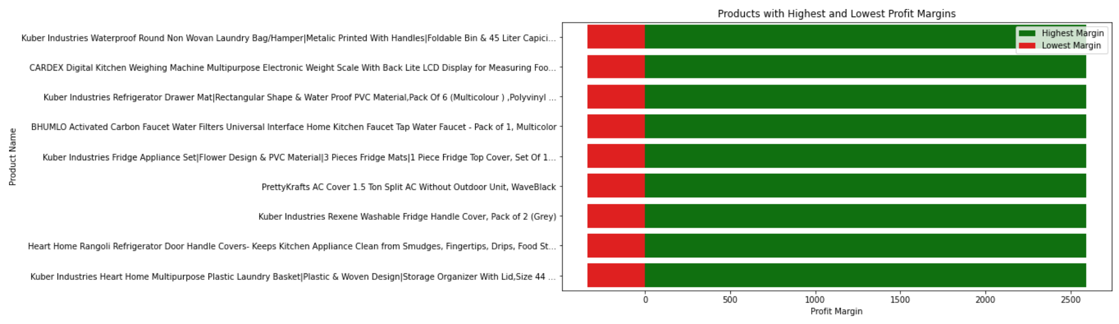
```
[142] : # Create a bar chart to show the top 5 main_category by sales_metric
        top_5_categories   =   data.groupby('main_category')['sales_per'].sum().nlargest(5).
          ↪reset_index()
        plt.figure(figsize=(12, 6))
        sb.barplot(data=top_5_categories, x='sales_per', y='main_category')
        plt.title('Top 5 Product Categories in Sales')
        plt.xlabel('Sales Percentage (Sum)')
        plt.ylabel('Main Category')
        plt.show()
        # Explain the top 5 main categories with the highest sales_metric.
```



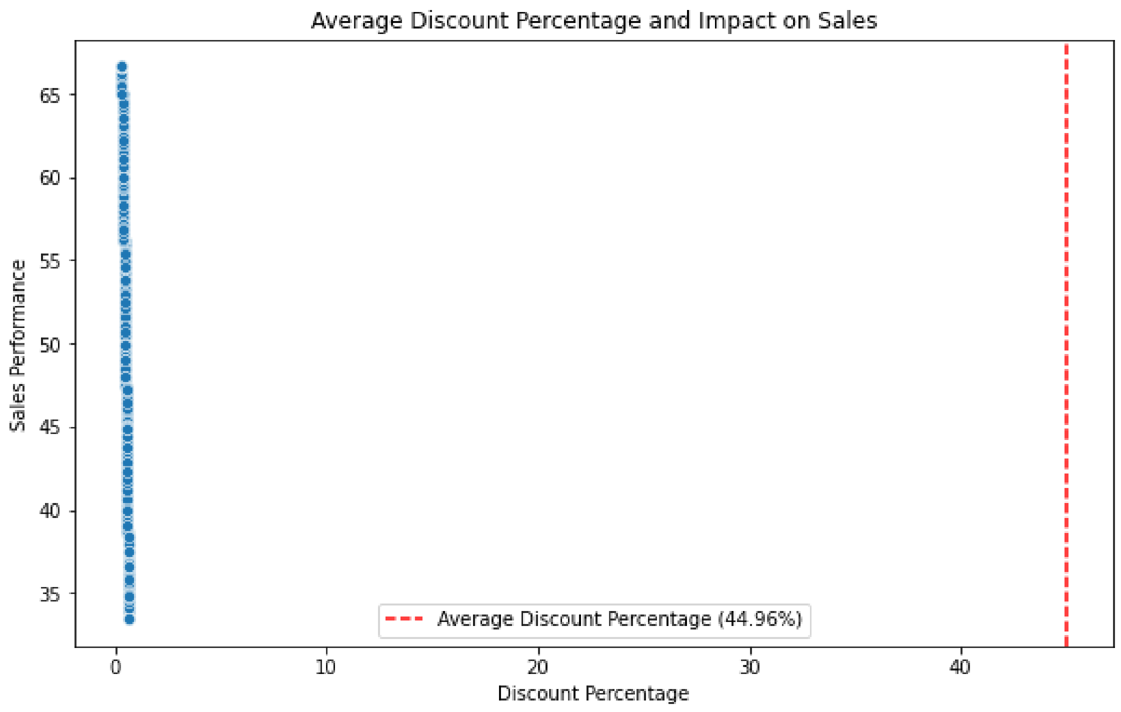
```
[143] : # Create a bar chart to show the least 5 main_category by sales_metric
        least_5_categories = data.groupby('main_category')['sales_per'].sum().
          ↪nsmallest(5).reset_index()
        plt.figure(figsize=(12, 6))
        sb.barplot(data=least_5_categories, x='sales_per', y='main_category')
        plt.title('Least 5 Product Categories in Sales')
        plt.xlabel('Sales Performance (Sum)')
        plt.ylabel('Main Category')
        plt.show()
        # Explain the least 5 main categories with the lowest sales_metric.
```



[144]:
```
# Calculate profit margins and create a bar chart or list products with the highest and lowest margins
data['profit_margin'] = data['actual_price'] - data['discount_price']
highest_margin_products = data.nlargest(10, 'profit_margin')
lowest_margin_products = data.nsmallest(10, 'profit_margin')
plt.figure(figsize=(12, 6))
sb.barplot(data=highest_margin_products, x='profit_margin', y='name', color='green', label='Highest Margin')
sb.barplot(data=lowest_margin_products, x='profit_margin', y='name', color='red', label='Lowest  Margin')
plt.title('Products with Highest and Lowest Profit Margins')
plt.xlabel('Profit Margin')
plt.ylabel('Product Name')
plt.legend()
plt.show()
# Explain the products with the highest and lowest profit margins.
```

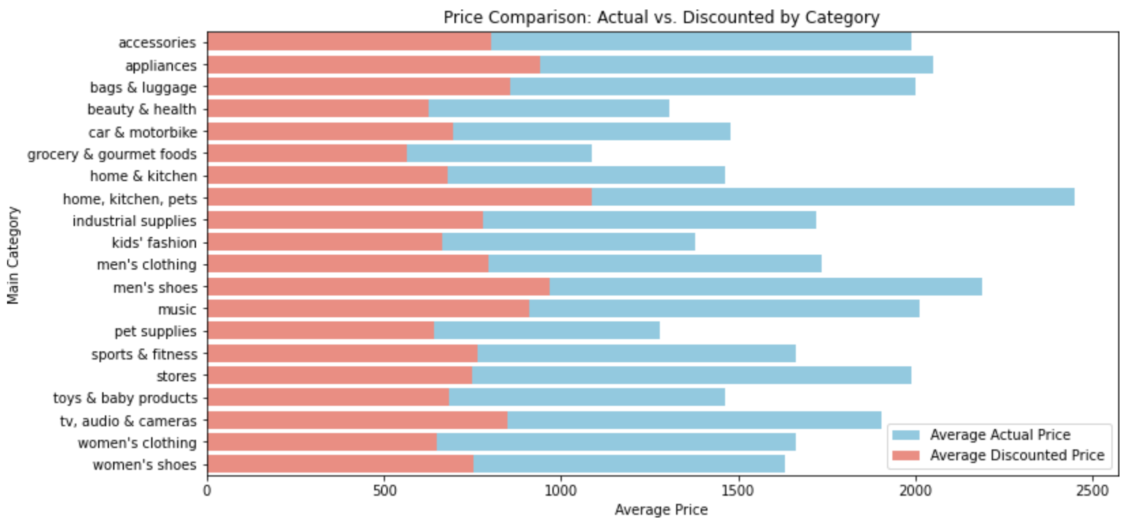


[145]:
```
# Calculate average discount percentage for products and analyze its impact on␣
↪sales_metric
avg_discount_percentage = (data['discount_price'] / data['actual_price']).
↪mean() * 100
plt.figure(figsize=(10, 6))
sb.scatterplot(data=data, x='Discount_per', y='sales_per')
plt.axvline(x=avg_discount_percentage, color='red', linestyle='--',␣
↪label=f'Average Discount Percentage ({avg_discount_percentage:.2f}%)')
plt.title('Average Discount Percentage and Impact on Sales')
plt.xlabel('Discount Percentage')
plt.ylabel('Sales Performance')
plt.legend()
plt.show()
# Explain the relationship between discount percentage and sales_metric.
```



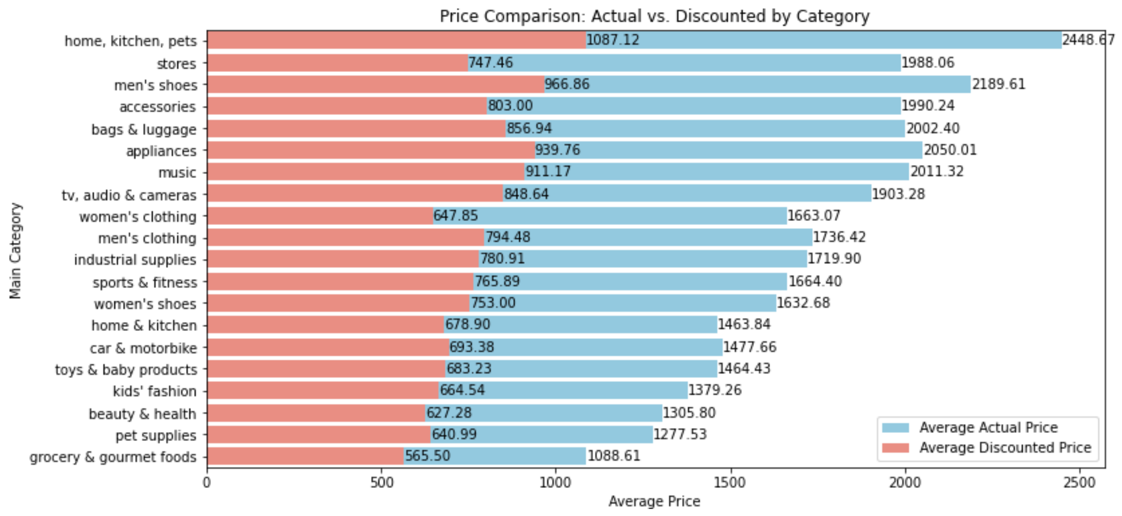
[146]:
```
# Create a bar chart to compare the average actual prices and discounted prices␣
↪by main_category
avg_actual_prices = data.groupby('main_category')['actual_price'].mean().
↪reset_index()
avg_discounted_prices = data.groupby('main_category')['discount_price'].mean().
↪reset_index()
plt.figure(figsize=(12, 6), FigureClass=plt.figure)
```

```
sb.barplot(data=avg_actual_prices,  x='actual_price',  y='main_category',␣
↪label='Average Actual Price', color='skyblue')
sb.barplot(data=avg_discounted_prices, x='discount_price', y='main_category',␣
↪label='Average  Discounted  Price',  color='salmon')
plt.title('Price Comparison: Actual vs. Discounted by Category')
plt.xlabel('Average Price')
plt.ylabel('Main Category')
plt.legend()
plt.show()
# Explain how actual prices compare to discounted prices by category.
```



[147] :
```
avg_actual_prices   =   data.groupby('main_category')['actual_price'].mean().
↪reset_index()
avg_discounted_prices  =  data.groupby('main_category')['discount_price'].mean().
↪reset_index()

avg_price_diff = avg_actual_prices['actual_price'] -␣
↪avg_discounted_prices['discount_price']
avg_price_diff_df = pd.DataFrame({'main_category':␣
↪avg_actual_prices['main_category'], 'avg_price_diff': avg_price_diff})

max_diff_category    =    avg_price_diff_df.loc[avg_price_diff_df['avg_price_diff'].
↪idxmax(), 'main_category']

plt.figure(figsize=(12, 6))

# Define the order of categories based on average price difference
order  =  avg_price_diff_df.sort_values(by='avg_price_diff',␣
↪ascending=False)['main_category']
```

```
# Plotting average actual prices
ax = sb.barplot(data=avg_actual_prices, x='actual_price',␣
↪y='main_category',order=order ,label='Average Actual Price', color='skyblue')

# Plotting average discounted prices
sb.barplot(data=avg_discounted_prices, x='discount_price', y='main_category',␣
↪order=order,label='Average Discounted Price', color='salmon')

# Adding annotations for actual prices
for p in ax.patches:
    ax.annotate(f'{p.get_width():.2f}', (p.get_width(), p.get_y() + p.
↪get_height() / 2),
                ha='left', va='center', color='black', fontsize=10)

plt.title('Price Comparison: Actual vs. Discounted by Category')
plt.xlabel('Average Price')
plt.ylabel('Main Category')
plt.legend()
plt.show()
```

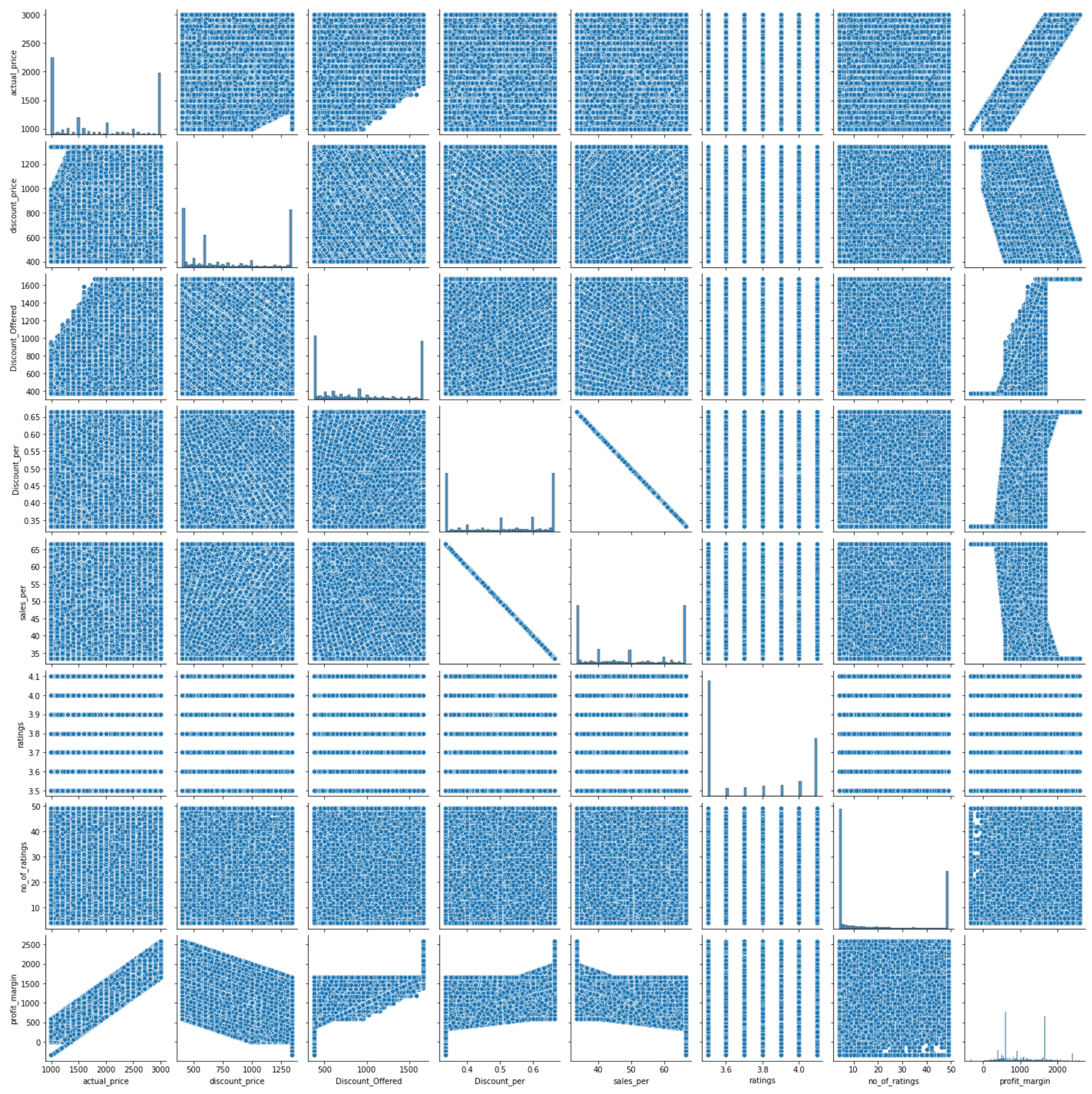


## 17 Multi variate analysis

[151]:
```
numeric_columns=data.select_dtypes(include="number").columns
```

[152]:
```
sb.pairplot(data[numeric_columns])
```

[152]:
```
<seaborn.axisgrid.PairGrid at 0x1e9c22dd810>
```

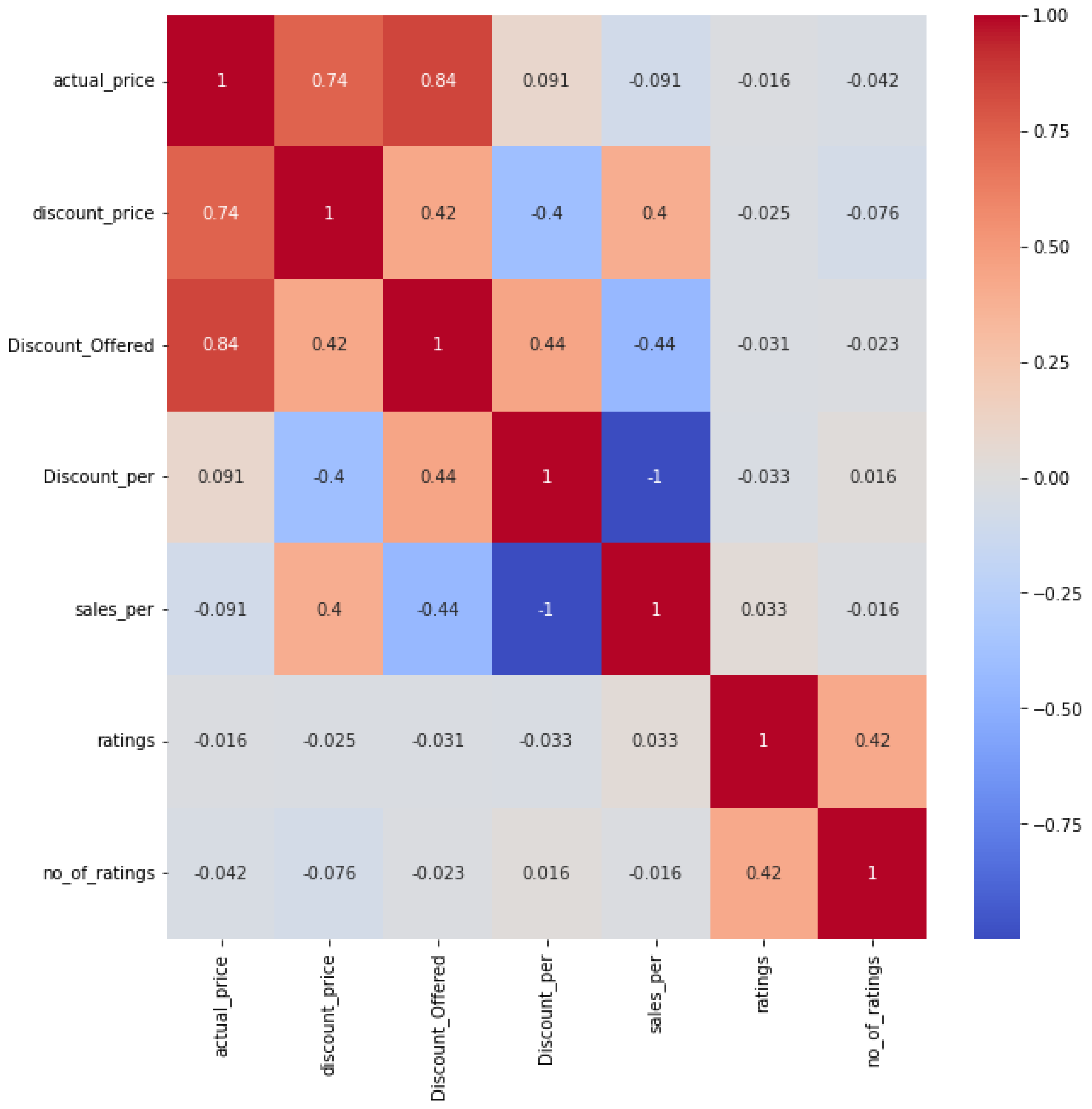


[319]:
```
numeric_columns = data.select_dtypes(include="number").columns
print(numeric_columns)
```

```
Index(['actual_price', 'discount_price', 'Discount_Offered', 'Discount_per',
       'sales_per', 'ratings', 'no_of_ratings'],
      dtype='object')
```

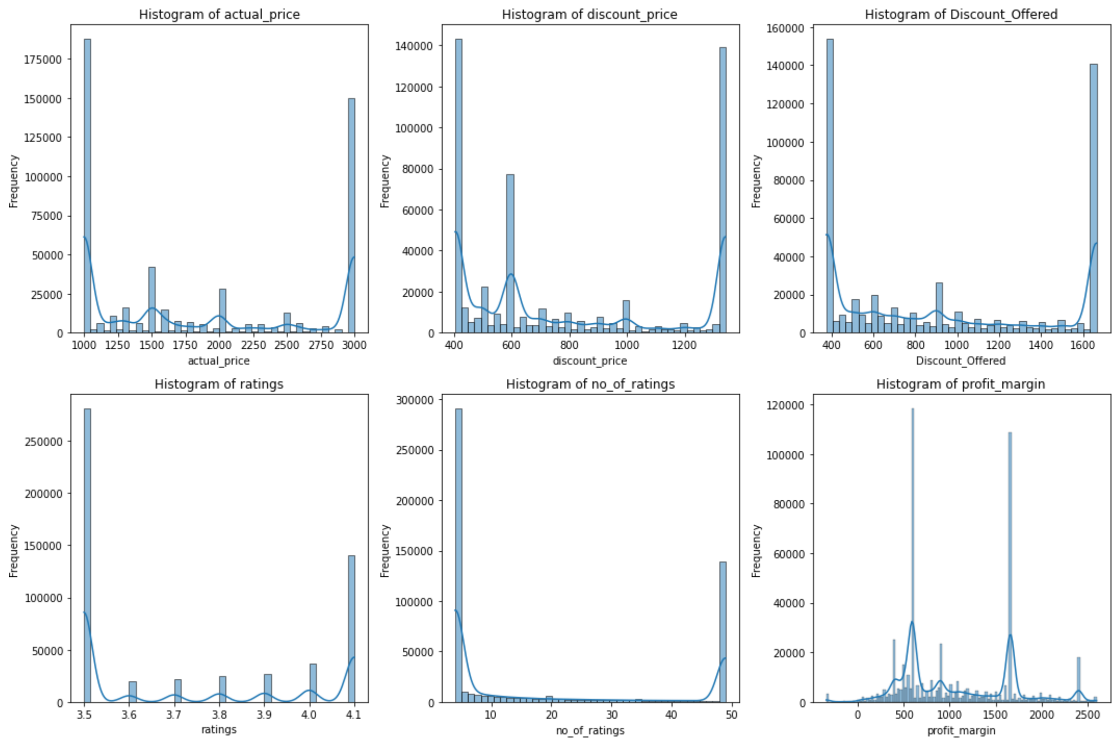
[320]:
```
corr = data[numeric_columns].corr()
plt.figure(figsize=(10,10))
sb.heatmap(corr, annot=True, cmap='coolwarm')
```

[320]:
```
<Axes: >
```



## 17.1 Distribution

[155]:
```
numerical_columns = ['actual_price', 'discount_price', 'Discount_Offered',␣
↪'ratings', 'no_of_ratings', 'profit_margin']

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))   # Define the␣
↪layout of subplots

for i, column in enumerate(numerical_columns):
    row = i // 3   # Calculate the row for the subplot
    col = i % 3    # Calculate the column for the subplot
    sb.histplot(data[column], kde=True, ax=axes[row, col])
    axes[row, col].set_title(f'Histogram of {column}')
    axes[row, col].set_xlabel(column)
    axes[row, col].set_ylabel('Frequency')

plt.tight_layout()   # Adjust layout to prevent overlapping
plt.show()
```



## 17.2 Insight

Here by seeing the graph, we can say that non of the graph is normally distributed - Actual_price, Discount_price, Discount_Offered, Ratings, No.of_ratings have right skewed data distribution because, we can observe that the frequency of the data is high in right hand side of graph

[156]:
```
# Statistical measures
for column in numerical_columns:
    print(f"Statistics for {column}:")
    print(f"Mean: {data[column].mean()}")
    print(f"Median: {data[column].median()}")
    print(f"Standard Deviation: {data[column].std()}\n")
```

```
Statistics for actual_price:
Mean: 1848.8122827850648
Median: 1499.0
Standard Deviation: 831.4662775417376
```

```
Statistics for discount_price:
Mean: 793.1165115077458
Median: 599.0
Standard Deviation: 375.8988836825849

Statistics for Discount_Offered:
Mean: 941.5443784729462
Median: 802.0
Standard Deviation: 522.5584926091415

Statistics for ratings:
Mean: 3.7304678336067876
Median: 3.5
Standard Deviation: 0.26230483485834805

Statistics for no_of_ratings:
Mean: 18.68097935948222
Median: 4.0
Standard Deviation: 19.354327837621558

Statistics for profit_margin:
Mean: 1055.695771277319
Median: 900.0
Standard Deviation: 609.2370568987421
```

### 17.3 Insight

1) By using these statistical measures, we surely conclude that the data is distributes positive in the entire data set.
2) Because we observe that mean value is greater than median
3) According to positive skewness order of statistical measures
   - mode< median< mean

[160]:
```python
import scipy.stats as stats
```

[322]:
```python
# Identify distribution types
for column in numerical_columns:
    print(f"Distribution type for {column}: {stats.normaltest(data[column])}")
```

```
Distribution type for actual_price:
NormaltestResult(statistic=71328.19009633732, pvalue=0.0)
Distribution type for discount_price:
NormaltestResult(statistic=80501.54805452723, pvalue=0.0)
Distribution type for Discount_Offered:
NormaltestResult(statistic=43129.065936540464, pvalue=0.0)
Distribution type for ratings: NormaltestResult(statistic=17967.26587022675,
pvalue=0.0)
Distribution type for no_of_ratings:
NormaltestResult(statistic=88277.43134758572, pvalue=0.0)
```

AS p_val is less than 0.05 we reject the null hypothesis.so the data doesnot follow a normal distribution in of these columns

[323]:
```
data['normalized_ratings'] = data['ratings'].apply(lambda x: np.log(x) if x > 0␣
↪else 0)
```

[338]:
```
print(f"Distribution type for normalized_ratings: {stats.
↪normaltest(data['normalized_ratings'])}")
```

```
Distribution type for normalized_ratings:
NormaltestResult(statistic=3770.009681021775, pvalue=0.0)
```

[339]:
```
data['normalized_discount_price'] = data['discount_price'].apply(lambda x: np.
↪log(x) if x > 0 else 0)
```

[340]:
```
print(f"Distribution type for normalized_ratings: {stats.
↪normaltest(data['normalized_discount_price'])}")
```

```
Distribution type for normalized_ratings:
NormaltestResult(statistic=3194.2390113311684, pvalue=0.0)
```

---

## 18 Hypothesis Testing

[163]:
```
import scipy.stats as stats
```

## 19 1. ANOVA TESTING

The ANOVA (Analysis of Variance) test is a way to find out if survey or experiment results are significant

#### 19.0.1 Outputs

F-statistics: Ratio of the variance between the groups to the variance within the groups

P-value: Probability of obtainging an F-statistic extreme than the observed value

A comparison of average sales percentages across different main categories using ANOVA test. - we only get the result for “Is there any significant differance betweeen each group” - but we cant specify which two group differ from each other - for this we need to perform post hoc test

#### 19.0.2 Assumptions

- Null hypothesis: There’s no difference in the means of brands

- Alternative hypothesis: Theres is a differnace in the means of brands

[165]:
```
# Group the data by main category and calculate the mean sales percentage by␣
↪each main category
main_cat_groups = data.groupby('main_category')['sales_per'].mean()

# Perform ANOVA test
f_val, p_val = stats.f_oneway(*[data.loc[data['main_category'] == category,␣
↪'sales_per'] for category in main_cat_groups.index])
#  "*" is used for unpacking the list of series into separate arguments
# here we are only selecting the rows that is equal to category(loc used to␣
↪select a subset of the dataframe based on certain condition)

f_val, p_val
```

[165]:
```
(2005.7992183106674, 0.0)
```

Here we perform the one way ANOVA test, which tests the null hypothesis that two or more groups have the same populaltion mean.

[166]:
```
print(p_val)
```

```
0.0
```

[167]:
```
if p_val<0.05:
    print("reject null hypothesis")
else:
    print("accept null hypothesis")
```

```
reject null hypothesis
```

So , we conclude that there is a differance in the means of the main categories

---

### 19.1 2. T-TEST

performing a two-sample t-test to compare the average discount offered on products between Brand A and Brand B

[168]:
```
brand_list=list(data['Brand_Name'].unique())
['Lloyd', 'LG', 'Carrier', 'Voltas', 'Daikin', 'Panasonic', 'Whirlpool',␣
↪'Samsung', 'Godrej', 'Blue', 'IFB', 'Cruise', 'AmazonBasics', 'Haier',␣
↪'Hitachi', 'Amazon',
 'OGENERAL', 'Small', 'ALLWIN', 'Hexzone', 'Candy', 'O-General', 'ONIDA']
# print(brand_list)
print("Select any two from above")
Brand_A = input("Enter Brand A name(Check Capitals and small letters): ")
Brand_B = input("Enter Brand B name(Check Capitals and small letters): ")
```

```
if Brand_A in brand_list and Brand_B in brand_list:
    brand_a_data = data[data['Brand_Name'] == Brand_A]
    brand_b_data = data[data['Brand_Name'] == Brand_B]

    # Extract the discount offered for each brand
    discount_offered_a = brand_a_data['Discount_Offered']
    discount_offered_b = brand_b_data['Discount_Offered']

    # Perform the two-sample t-test
    t_statistic, p_value =stats.ttest_ind(discount_offered_a,␣
 ↪discount_offered_b)

    # Print the results
    print('t-statistic:', t_statistic)
    print('p-value:', p_value)

    # Interpret the results
    if p_value < 0.05:
        print('There is a significant difference in the average discount␣
 ↪offered on products between Brand A ({}) and Brand B ({}).'.format(Brand_A,␣
 ↪Brand_B))
    else:
        print('There is no significant difference in the average discount␣
 ↪offered on products between Brand A ({}) and Brand B ({}).'.format(Brand_A,␣
 ↪Brand_B))


else:
    print("Give the brand names correctly, please go throught the list once␣
 ↪again")
```

```
Select any two from above
t-statistic: -1.1876546353254425
p-value: 0.23502624767374233
There is no significant difference in the average discount offered on products
between Brand A (Pc) and Brand B (Puma).
```

```
C:\Users\uppada satwik\AppData\Local\Temp\ipykernel_11328\2975331360.py:18:
RuntimeWarning: Precision loss occurred in moment calculation due to
catastrophic cancellation. This occurs when the data are nearly identical.
Results may be unreliable.
  t_statistic, p_value =stats.ttest_ind(discount_offered_a, discount_offered_b)
```

### 19.1.1 Conclusion

- if the p_val is lessthan 0.05 then
  - statistically, the two brands offer similar discounts on their products
- else

- statistically, the two brands does not offer similar discounts on their products

---

## 20 3. PEARSON CORRELATION

[169]:
```
# Perform the Pearson correlation test
correlation_coefficient, p_value = stats.pearsonr(data['sales_per'],␣
↪data['no_of_ratings'])

# Print the results
print('Correlation coefficient:', correlation_coefficient)
print('p-value:', p_value)

if abs(correlation_coefficient) > 0.3 and p_value < 0.05:
    print('There is a weak to moderate significant relationship between the␣
↪average sales percentage product and the number of ratings a product has.')
elif abs(correlation_coefficient) > 0.5 and p_value < 0.05:
     print('There is a moderate to strong significant relationship between the␣
↪average sales percentage product and the number of ratings a product has.')
elif abs(correlation_coefficient) > 0.8 and p_value < 0.05:
    print('There is a strong significant relationship between the average sales␣
↪percentage product and the number of ratings a product has.')
else:
    print('There is no significant relationship between the average sales␣
↪percentage product and the number of ratings a product has.')
```

```
Correlation coefficient: -0.016155496373994296
p-value: 3.585563054203087e-33
There is no significant relationship between the average sales percentage
product and the number of ratings a product has.
```

The correlation coefficient measures the strength and direction of the linear relationship between two variables - -0.016 indicate a very weak correlation is statistically significant

---

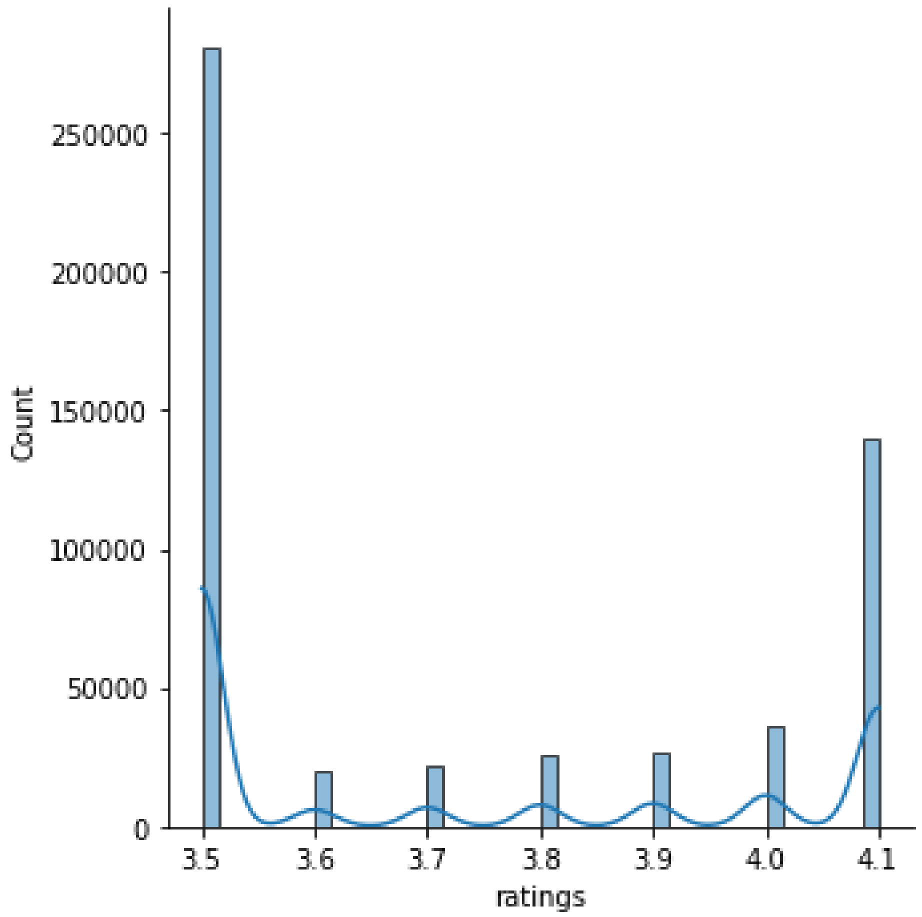
## 21 CENTRAL LIMIT THEOREM

[170]:
```
df= data[['ratings']]
df
```

[170]:
```
   ratings
0      4.1
1      4.1
2      4.1
3      4.0
4      4.1
```

```
...          ...
551580       3.5
551581       3.5
551582       4.0
551583       4.1
551584       4.1

[551585 rows x 1 columns]
```

[171] :
```
sb.displot(df.ratings,kde=True)
plt.show()
```



### 21.1 Population Mean

[172] :
```
df.ratings.mean()
```

[172]:
```
3.7304678336067876
```

This is the true mean weight of the population. This is the population parameter, the ground truth.

Let's take a random sample from this data, and see what mean we get.

## 21.2 Sample Mean

```
[173] : sample_size= 30
```

```
[174] : df.ratings.sample(sample_size).mean()
```

```
[174]:  3.7133333333333325
```

Slight different from the population mean, right?
Let's take another sample.

```
[175] : df.ratings.sample(sample_size).mean()
```

```
[175]:  3.7966666666666664
```

```
[176] : df.ratings.sample(sample_size).mean()
```

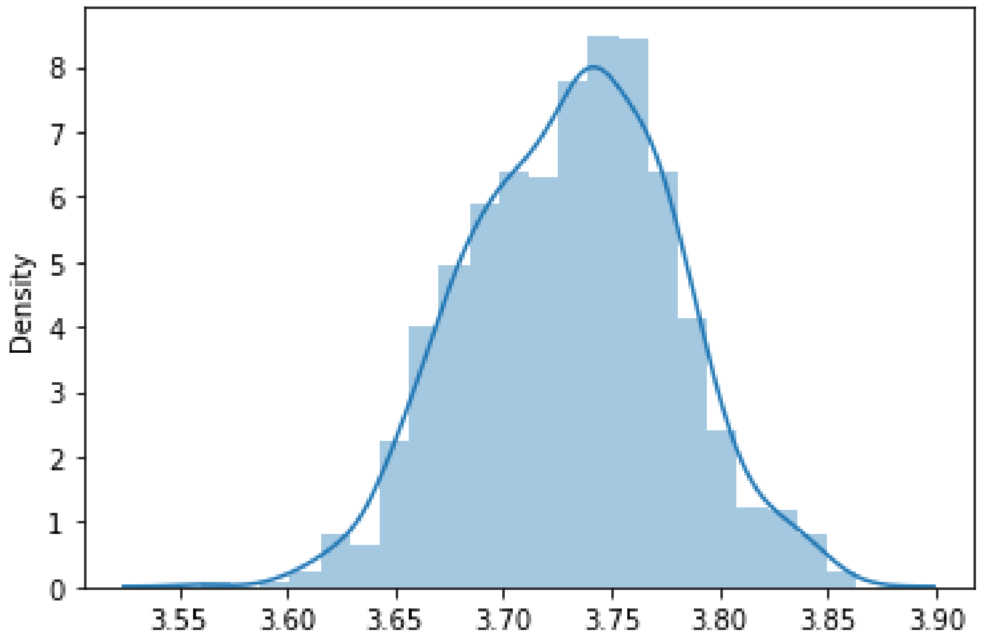
```
[176]:  3.6966666666666666
```

Each time we take a sample, our mean value is different. There is variability in the sample mean itself. Does the sample mean itself follow a distribution? Let's assess this.
We'll take many samples from the data, and plot a histogram of the same.

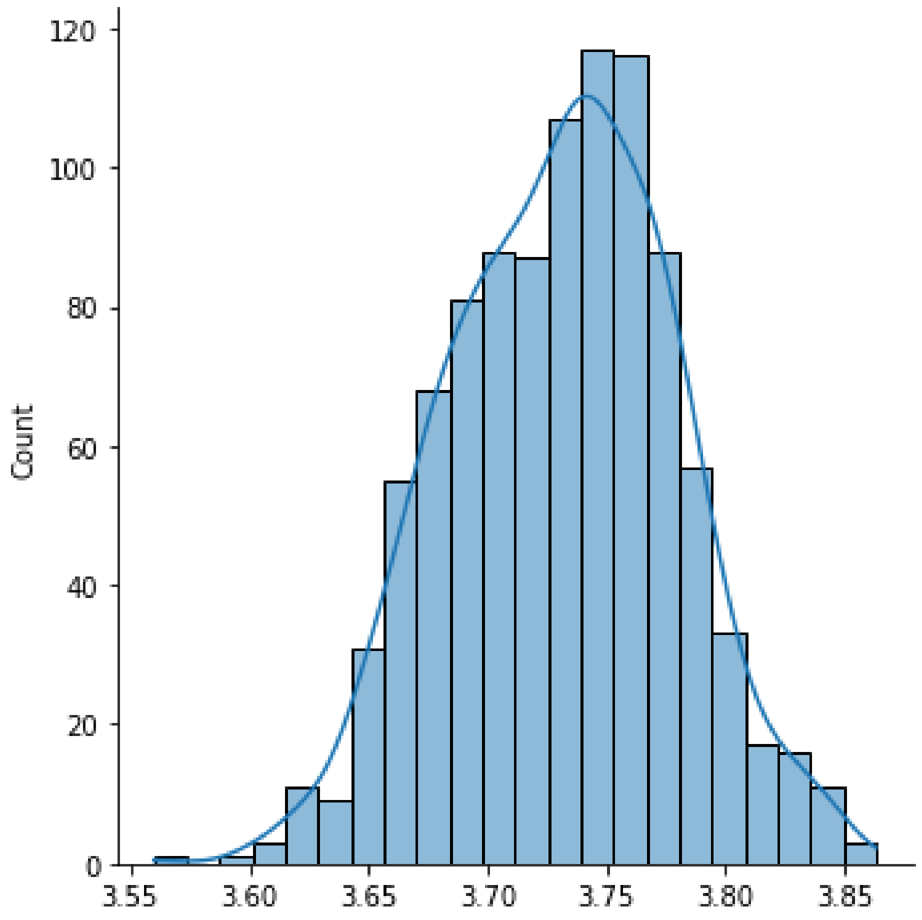
```
[177] : sample_means = [df.ratings.sample(sample_size).mean() for i in range(1000)]
        sample_means = pd.Series(sample_means)
```

```
[178] : sb.distplot(sample_means)
        plt.show()
```

```
C:\Users\uppada satwik\AppData\Local\Temp\ipykernel_11328\2426809856.py:1:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sb.distplot(sample_means)
```



[179] :
```
sb.displot(sample_means,kde=True)
plt.show()
```



It is a distribution of sample means, which is a histogram with a Kernel Density Estimate (KDE) overlay. This type of distribution is known as a Sampling Distribution.

From the plot, it appears that the distribution of sample means follows a **Normal Distribution** (also known as a Gaussian Distribution), which is characterized by its bell shape and symmetry around the mean. This is in line with the **Central Limit Theorem**, which states that the distribution of sample means will approximate a normal distribution as the sample size becomes large, regardless of the shape of the population distribution.

The mean of this distribution is approximately **3.75**, which is the peak of the bell curve. This suggests that the average rating across all samples is around 3.75. The spread of the distribution gives us an idea about the variability of the ratings. The narrower the bell curve, the less variability there is in the ratings.

## 21.3 Mean of the sample means

[180] :
```
sample_means.mean()
```

[180]:
```
3.7312999999999996
```

## 21.4 Standard deviation of your sample means

[181] :
```
sample_means.std()
```

[181]:
```
0.047334454159212
```

## 21.5 standard error of the population

Population std vs. std of sampling mean

[182] :
```
df.ratings.std()/np.sqrt(sample_size)
```

[182]:
```
0.047890091664861544
```

## 21.6 CONCLUSION

- The values were very close, which supports the CLT. This implies that the sample means provide a good estimate of the population mean, and the spread of the sample means around this estimate is captured by the standard deviation of the sample means (or equivalently, the standard error of the population).
- The analysis provides evidence that the 'ratings' data is well-behaved and suitable for statistical inference