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

data = pd.read_csv('BigBasket Products.csv')
print(data.head())
print(data.describe())
print(data.info())
```

```
   index                                            product  \
0      1                Garlic Oil - Vegetarian Capsule 500 mg
1      2                             Water Bottle - Orange
2      3                     Brass Angle Deep - Plain, No.2
3      4  Cereal Flip Lid Container/Storage Jar - Assort...
4      5                   Creme Soft Soap - For Hands & Body

                category           sub_category             brand  \
0         Beauty & Hygiene              Hair Care  Sri Sri Ayurveda
1  Kitchen, Garden & Pets  Storage & Accessories         Mastercook
2     Cleaning & Household           Pooja Needs                Trm
3     Cleaning & Household   Bins & Bathroom Ware             Nakoda
4         Beauty & Hygiene        Bath & Hand Wash              Nivea

   sale_price  market_price                       type  rating  \
0       220.0         220.0          Hair Oil & Serum     4.1
1       180.0         180.0    Water & Fridge Bottles     2.3
2       119.0         250.0           Lamp & Lamp Oil     3.4
3       149.0         176.0  Laundry, Storage Baskets     3.7
4       162.0         162.0       Bathing Bars & Soaps     4.4

                                         description
0  This Product contains Garlic Oil that is known...
1  Each product is microwave safe (without lid), ...
2  A perfect gift for all occasions, be it your m...
3  Multipurpose container with an attractive desi...
4  Nivea Creme Soft Soap gives your skin the best...
              index    sale_price  market_price        rating
count  27555.00000   27549.000000  27555.000000  18919.000000
mean   13778.00000     334.648391    382.056664      3.943295
std     7954.58767    1202.102113    581.730717      0.739217
min        1.00000       2.450000      3.000000      1.000000
25%     6889.50000      95.000000    100.000000      3.700000
50%    13778.00000     190.320000    220.000000      4.100000
75%    20666.50000     359.000000    425.000000      4.300000
max    27555.00000  112475.000000  12500.000000      5.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27555 entries, 0 to 27554
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   index          27555 non-null  int64
 1   product        27554 non-null  object
 2   category       27555 non-null  object
 3   sub_category   27555 non-null  object
 4   brand          27554 non-null  object
 5   sale_price     27549 non-null  float64
 6   market_price   27555 non-null  float64
 7   type           27555 non-null  object
 8   rating         18919 non-null  float64
 9   description    27440 non-null  object
dtypes: float64(3), int64(1), object(6)
memory usage: 2.1+ MB
None
```

In [ ]:
```

```

In [15]:
```
data['product'].fillna("Unknown Product", inplace=True)
data['brand'].fillna("Unknown Brand", inplace=True)
data['sale_price'].fillna(data['sale_price'].median(), inplace=True)
data['rating'].fillna(data['rating'].median(),inplace=True)
data['description'].fillna("No Description Available", inplace=True)
data.isnull().sum()
```

```
C:\Users\toshi\AppData\Local\Temp\ipykernel_3112\405246921.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace meth
od.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace o
n the original object.


  data['product'].fillna("Unknown Product", inplace=True)
C:\Users\toshi\AppData\Local\Temp\ipykernel_3112\405246921.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace meth
od.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace o
n the original object.


  data['brand'].fillna("Unknown Brand", inplace=True)
C:\Users\toshi\AppData\Local\Temp\ipykernel_3112\405246921.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace meth
od.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace o
n the original object.


  data['sale_price'].fillna(data['sale_price'].median(), inplace=True)
C:\Users\toshi\AppData\Local\Temp\ipykernel_3112\405246921.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace meth
od.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace o
n the original object.


  data['rating'].fillna(data['rating'].median(), inplace=True)
C:\Users\toshi\AppData\Local\Temp\ipykernel_3112\405246921.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace meth
od.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace o
n the original object.


  data['description'].fillna("No Description Available", inplace=True)
```

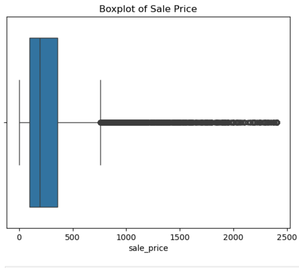
Out[15]:
```
index                0
product              0
category             0
sub_category         0
brand                0
sale_price           0
market_price         0
type                 0
rating               0
description          0
discount_percentage    0
dtype: int64
```

In [19]:
```
top_sales = data.groupby('product')['sale_price'].sum().sort_values(ascending=False).head(5)
least_sales = data.groupby('product')['sale_price'].sum().sort_values().head(5)
```

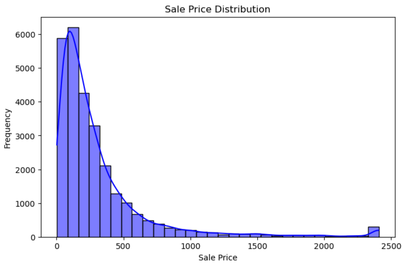
In [7]:
```
data['discount_percentage'] = ((data['market_price'] - data['sale_price']) / data['market_price']) * 100
print(data[['product', 'sale_price', 'market_price', 'discount_percentage']].head())
```

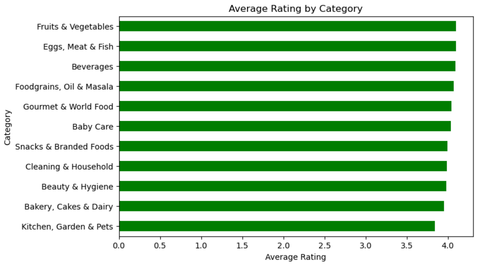
```
                                             product  sale_price  \
0              Garlic Oil - Vegetarian Capsule 500 mg       220.0
1                               Water Bottle - Orange       180.0
2                       Brass Angle Deep - Plain, No.2       119.0
3  Cereal Flip Lid Container/Storage Jar - Assort...       149.0
4                     Creme Soft Soap - For Hands & Body       162.0

   market_price  discount_percentage
0         220.0             0.000000
1         180.0             0.000000
2         250.0            52.400000
3         176.0            15.340909
4         162.0             0.000000
```

In [21]:
```
sns.boxplot(data=data, x='sale_price')
plt.title('Boxplot of Sale Price')
plt.show()
q99 = data['sale_price'].quantile(0.99)
data['sale_price'] = np.where(data['sale_price'] > q99, q99, data['sale_price'])
```



In [21]:
```
plt.figure(figsize=(8, 5))
sns.histplot(data['sale_price'], bins=30, color='blue', kde=True)
plt.title('Sale Price Distribution')
plt.xlabel('Sale Price')
plt.ylabel('Frequency')
plt.show()

category_ratings = data.groupby('category')['rating'].mean().sort_values()
category_ratings.plot(kind='barh', figsize=(8, 5), color='green')
plt.title('Average Rating by Category')
plt.xlabel('Average Rating')
plt.ylabel('Category')
plt.show()
```





In [ ]:
```

```