Problem Statement:

Develop an integrated system for analyzing and interacting with a dataset of products from an online marketplace, known as the "BigBasket Products.csv," with the goal of enhancing user experience and extracting valuable insights. The dataset contains various attributes of products, such as category, description, product name, sale prices, market prices, and ratings.

Key Objectives:

Data Management and Quality Assurance:

Import necessary libraries and modules for data manipulation, visualization, and machine learning. Ensure data quality by handling missing values and applying appropriate preprocessing techniques. Data Exploration and Visualization:

Explore the dataset by generating visualizations that provide insights into product distribution, pricing trends, and the relationships between key attributes. Examine correlations and patterns in the dataset to uncover actionable insights for businesses. Text Data Processing:

Perform text preprocessing on textual attributes, including 'description' and 'product_name,' to make them suitable for text classification. Apply techniques such as lowercasing, HTML tag removal, special character and digit removal, tokenization, stopword removal, and lemmatization. Text Classification:

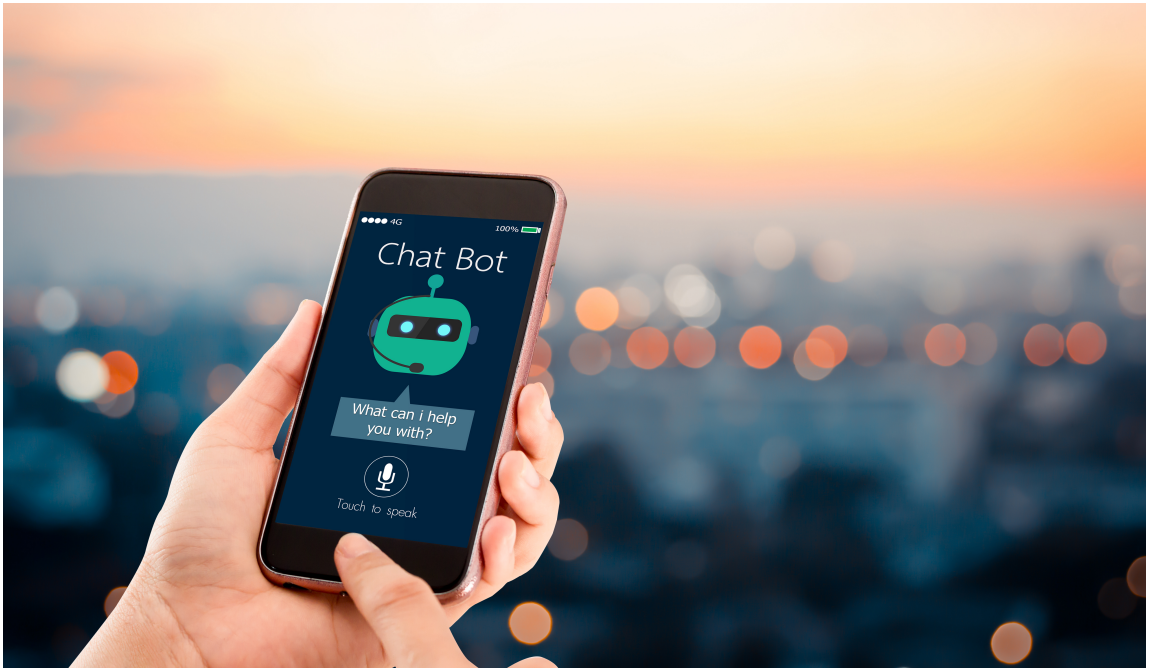
Implement text classification algorithms, including Multinomial Naive Bayes, Logistic Regression, Random Forest, and Support Vector Machine (SVM). Utilize TF-IDF vectorization to convert text data into numerical features. Evaluate the performance of each classifier in terms of accuracy and classification reports. Hyperparameter Tuning:

Fine-tune the classifiers using RandomizedSearchCV to identify the best hyperparameters. Select the best-performing classifier to improve text classification accuracy. Chatbot for Product Details:

Develop a chatbot that allows users to input a product name and retrieve its details from the dataset. Ensure user-friendly interactions and informative responses to user queries. Extended Chatbot Functionality:

Extend the chatbot's capabilities to include listing available product categories and an exit option. Improve user experience and assist users in exploring the dataset more effectively. Final Model Training and Deployment:

Select the best-tuned SVM model for text classification. Perform TF-IDF vectorization on the entire dataset to create a robust text classification model. Develop a chatbot function to accept user input, predict the category of new text inputs, and display results.



In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import re
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Ignore warning messages
warnings.filterwarnings("ignore")
```

Step 1: Importing Libraries and Ignoring Warnings

This step imports the necessary libraries for data analysis, visualization, and machine learning. Installing and downloading the spacy library and en_core_web_sm model suggests the intention to perform natural language processing tasks.

In [2]:
```
!pip install spacy
```

```
Requirement already satisfied: spacy in c:\users\pakbo\anaconda3\lib\site-packages
(3.7.1)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in c:\users\pakbo\anaconda3
\lib\site-packages (from spacy) (2.27.1)
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in c:\users\pakbo\anaconda
3\lib\site-packages (from spacy) (6.4.0)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in c:\users\pakbo\anaconda3\l
ib\site-packages (from spacy) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.1.8 in c:\users\pakbo\anaconda3\lib
\site-packages (from spacy) (8.2.1)
Requirement already satisfied: weasel<0.4.0,>=0.1.0 in c:\users\pakbo\anaconda3\li
b\site-packages (from spacy) (0.3.2)
Requirement already satisfied: setuptools in c:\users\pakbo\anaconda3\lib\site-pac
kages (from spacy) (61.2.0)
Requirement already satisfied: packaging>=20.0 in c:\users\pakbo\anaconda3\lib\sit
e-packages (from spacy) (21.3)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in c:\users\pakbo\anaconda3
\lib\site-packages (from spacy) (3.3.0)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in c:\users\pakbo\anaco
nda3\lib\site-packages (from spacy) (3.0.12)
Requirement already satisfied: typer<0.10.0,>=0.3.0 in c:\users\pakbo\anaconda3\li
b\site-packages (from spacy) (0.4.2)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in c:\users\pakbo\anaconda3
\lib\site-packages (from spacy) (2.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in c:\users\pakbo\anaconda3\lib
\site-packages (from spacy) (2.0.8)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in c:\users\pakbo\anacond
a3\lib\site-packages (from spacy) (1.0.10)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in c:\users\pakbo\anaconda3\lib
\site-packages (from spacy) (2.4.8)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in c:\users\pakbo\anaconda3\lib
\site-packages (from spacy) (4.65.0)
Requirement already satisfied: jinja2 in c:\users\pakbo\anaconda3\lib\site-package
s (from spacy) (2.11.3)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in c:\users\pakbo\anaco
nda3\lib\site-packages (from spacy) (1.0.5)
Requirement already satisfied: pathy>=0.10.0 in c:\users\pakbo\anaconda3\lib\site-
packages (from spacy) (0.10.2)
Requirement already satisfied: numpy>=1.19.0 in c:\users\pakbo\anaconda3\lib\site-
packages (from spacy) (1.22.4)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in c:\users\pakbo\anaconda3\li
b\site-packages (from spacy) (1.1.2)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in c:\users\pa
kbo\anaconda3\lib\site-packages (from spacy) (1.10.7)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\pakbo\anaconda
3\lib\site-packages (from packaging>=20.0->spacy) (3.0.4)
Requirement already satisfied: typing-extensions>=4.2.0 in c:\users\pakbo\anaconda
3\lib\site-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (4.5.0)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\pakbo\anacond
a3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\pakbo\anaconda3\lib\site-p
ackages (from requests<3.0.0,>=2.13.0->spacy) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\pakbo\anaconda3\lib
\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\pakbo\anaconda3\l
ib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (1.26.9)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in c:\users\pakbo\anaconda3\lib
\site-packages (from thinc<8.3.0,>=8.1.8->spacy) (0.7.11)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in c:\users\pakbo\anaconda
3\lib\site-packages (from thinc<8.3.0,>=8.1.8->spacy) (0.1.3)
Requirement already satisfied: colorama in c:\users\pakbo\anaconda3\lib\site-packa
ges (from tqdm<5.0.0,>=4.38.0->spacy) (0.4.6)
Requirement already satisfied: click<9.0.0,>=7.1.1 in c:\users\pakbo\anaconda3\lib
\site-packages (from typer<0.10.0,>=0.3.0->spacy) (8.0.4)
```

```
Requirement already satisfied: cloudpathlib<0.16.0,>=0.7.0 in c:\users\pakbo\anaco
nda3\lib\site-packages (from weasel<0.4.0,>=0.1.0->spacy) (0.15.1)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\pakbo\anaconda3\lib\si
te-packages (from jinja2->spacy) (2.0.1)
```

In [3]:
```
!python -m spacy download en_core_web_sm
```

```
Collecting en-core-web-sm==3.7.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_
web_sm-3.7.0/en_core_web_sm-3.7.0-py3-none-any.whl (12.8 MB)
Requirement already satisfied: spacy<3.8.0,>=3.7.0 in c:\users\pakbo\anaconda3\lib
\site-packages (from en-core-web-sm==3.7.0) (3.7.1)
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in c:\users\pakbo\anaconda
3\lib\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (6.4.0)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in c:\users\pakbo\anaconda3\lib
\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (2.4.8)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in c:\users\pakbo\anaconda3\lib
\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (4.65.0)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in c:\users\pakbo\anaconda3
\lib\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (2.0.10)
Requirement already satisfied: pathy>=0.10.0 in c:\users\pakbo\anaconda3\lib\site-
packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (0.10.2)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in c:\users\pakbo\anaconda3\li
b\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (1.1.2)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in c:\users\pakbo\anacond
a3\lib\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (1.0.10)
Requirement already satisfied: setuptools in c:\users\pakbo\anaconda3\lib\site-pac
kages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (61.2.0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in c:\users\pa
kbo\anaconda3\lib\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0)
(1.10.7)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in c:\users\pakbo\anaco
nda3\lib\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (1.0.5)
Requirement already satisfied: weasel<0.4.0,>=0.1.0 in c:\users\pakbo\anaconda3\li
b\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (0.3.2)
Requirement already satisfied: typer<0.10.0,>=0.3.0 in c:\users\pakbo\anaconda3\li
b\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (0.4.2)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in c:\users\pakbo\anaconda3\lib
\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (2.0.8)
Requirement already satisfied: packaging>=20.0 in c:\users\pakbo\anaconda3\lib\sit
e-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (21.3)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in c:\users\pakbo\anaconda3
\lib\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (3.3.0)
Requirement already satisfied: numpy>=1.19.0 in c:\users\pakbo\anaconda3\lib\site-
packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (1.22.4)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in c:\users\pakbo\anaconda3
\lib\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (2.27.1)
Requirement already satisfied: thinc<8.3.0,>=8.1.8 in c:\users\pakbo\anaconda3\lib
\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (8.2.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in c:\users\pakbo\anaco
nda3\lib\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (3.0.12)
Requirement already satisfied: jinja2 in c:\users\pakbo\anaconda3\lib\site-package
s (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (2.11.3)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in c:\users\pakbo\anaconda3\l
ib\site-packages (from spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (3.0.9)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\pakbo\anaconda
3\lib\site-packages (from packaging>=20.0->spacy<3.8.0,>=3.7.0->en-core-web-sm==3.
7.0) (3.0.4)
Requirement already satisfied: typing-extensions>=4.2.0 in c:\users\pakbo\anaconda
3\lib\site-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy<3.8.0,>=3.7.
0->en-core-web-sm==3.7.0) (4.5.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\pakbo\anaconda3\lib\site-p
ackages (from requests<3.0.0,>=2.13.0->spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0)
(3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\pakbo\anaconda3\l
ib\site-packages (from requests<3.0.0,>=2.13.0->spacy<3.8.0,>=3.7.0->en-core-web-s
m==3.7.0) (1.26.9)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\pakbo\anacond
a3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy<3.8.0,>=3.7.0->en-core-w
eb-sm==3.7.0) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\pakbo\anaconda3\lib
```

```
\site-packages (from requests<3.0.0,>=2.13.0->spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (2022.12.7)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in c:\users\pakbo\anaconda3\lib\site-packages (from thinc<8.3.0,>=8.1.8->spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (0.7.11)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in c:\users\pakbo\anaconda3\lib\site-packages (from thinc<8.3.0,>=8.1.8->spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (0.1.3)
Requirement already satisfied: colorama in c:\users\pakbo\anaconda3\lib\site-packages (from tqdm<5.0.0,>=4.38.0->spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (0.4.6)
Requirement already satisfied: click<9.0.0,>=7.1.1 in c:\users\pakbo\anaconda3\lib\site-packages (from typer<0.10.0,>=0.3.0->spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (8.0.4)
Requirement already satisfied: cloudpathlib<0.16.0,>=0.7.0 in c:\users\pakbo\anaconda3\lib\site-packages (from weasel<0.4.0,>=0.1.0->spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (0.15.1)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\pakbo\anaconda3\lib\site-packages (from jinja2->spacy<3.8.0,>=3.7.0->en-core-web-sm==3.7.0) (2.0.1)
[+] Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

In [4]:
```
# Load the product data from a CSV file
df = pd.read_csv('BigBasket Products.csv')
```

In [5]:
```
df.head(10)
```

Out[5]:

	index	product	category	sub_category	brand	sale_price	market_price	
0	1	Garlic Oil - Vegetarian Capsule 500 mg	Beauty & Hygiene	Hair Care	Sri Sri Ayurveda	220.0	220.0	Hair (Se
1	2	Water Bottle - Orange	Kitchen, Garden & Pets	Storage & Accessories	Mastercook	180.0	180.0	Wat Fr Bc
2	3	Brass Angle Deep - Plain, No.2	Cleaning & Household	Pooja Needs	Trm	119.0	250.0	Lan Lam
3	4	Cereal Flip Lid Container/Storage Jar - Assort...	Cleaning & Household	Bins & Bathroom Ware	Nakoda	149.0	176.0	Lau Sto Ba
4	5	Creme Soft Soap - For Hands & Body	Beauty & Hygiene	Bath & Hand Wash	Nivea	162.0	162.0	Bat Ba S
5	6	Germ - Removal Multipurpose Wipes	Cleaning & Household	All Purpose Cleaners	Nature Protect	169.0	199.0	Disinfe Spr Clea
6	7	Multani Mati	Beauty & Hygiene	Skin Care	Satinance	58.0	58.0	Face
7	8	Hand Sanitizer - 70% Alcohol Base	Beauty & Hygiene	Bath & Hand Wash	Bionova	250.0	250.0	Hand V Sanit
8	9	Biotin & Collagen Volumizing Hair Shampoo + Bi...	Beauty & Hygiene	Hair Care	StBotanica	1098.0	1098.0	Shan Conditi
9	10	Scrub Pad - Anti-Bacterial, Regular	Cleaning & Household	Mops, Brushes & Scrubs	Scotch brite	20.0	20.0	Ut Scrub C

Step 2: Loading Data

The code loads data from a CSV file named 'BigBasket Products.csv' into a Pandas DataFrame df. Displaying the first 10 rows of the DataFrame provides an initial view of the data.

In [6]:
```
df.info()
```

```
<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     27555 non-null  float64
 6   market_price   27555 non-null  float64
 7   type           27555 non-null  object 
 8   rating         18929 non-null  float64
 9   description    27440 non-null  object 
dtypes: float64(3), int64(1), object(6)
memory usage: 2.1+ MB
```

In [7]:
```
df.describe()
```

Out[7]:

	index	sale_price	market_price	rating
count	27555.00000	27555.000000	27555.000000	18929.000000
mean	13778.00000	322.514808	382.056664	3.943410
std	7954.58767	486.263116	581.730717	0.739063
min	1.00000	2.450000	3.000000	1.000000
25%	6889.50000	95.000000	100.000000	3.700000
50%	13778.00000	190.000000	220.000000	4.100000
75%	20666.50000	359.000000	425.000000	4.300000
max	27555.00000	12500.000000	12500.000000	5.000000

In [8]:
```
# Check for missing values in each column
missing_values = df.isnull().sum()

# Calculate the total number of rows in the DataFrame
total_rows = len(df)

# Calculate the percentage of missing values for each column
percentage_missing = (missing_values / total_rows) * 100

# Create a new DataFrame to display the missing values and their percentages
missing_data = pd.DataFrame({'Missing Values': missing_values, 'Percentage Missing

print(missing_data)
```

```
              Missing Values  Percentage Missing (%)
index                      0                0.000000
product                    1                0.003629
category                   0                0.000000
sub_category               0                0.000000
brand                      1                0.003629
sale_price                 0                0.000000
market_price               0                0.000000
type                       0                0.000000
rating                  8626               31.304663
description              115                0.417347
```

In [9]:
```
df.shape
```

Out[9]: (27555, 10)

Step 3: Checking Data Information and Missing Values

The df.info() function gives an overview of the DataFrame's structure, data types, and non-null values in each column. This helps in understanding the dataset's dimensions. The df.describe() function provides summary statistics for numerical columns, giving insights into data distribution. Calculating missing values and their percentages reveals potential data quality issues, which may require further handling.

In [10]:
```
# Data Exploration and Preprocessing
df['category'] = df['category'].str.replace("&", ",")
df['index'] = range(1, len(df) + 1)
df.drop(columns='index', inplace=True)
```

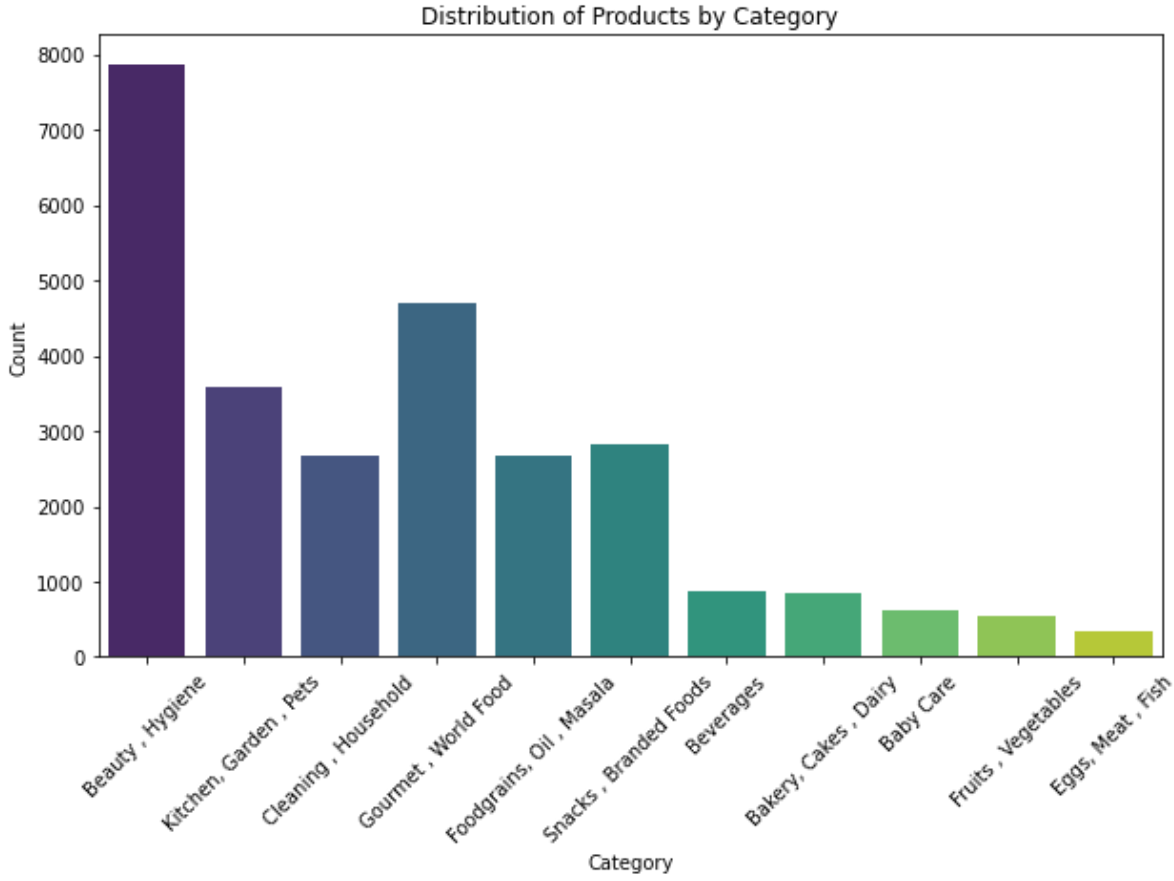
In [11]:
```
# Replace missing 'description' values with a combination of other columns
empty_space_indices = df.loc[df['description'].isna()].index
df['New_Description'] = df['product'] + ' | ' + df['category'] + ' | ' + df['sub_ca
df['description'].fillna(df['New_Description'], inplace=True)
df.drop(columns=['New_Description'], inplace=True)
```

In [12]:
```
# Extract product name from the 'product' column
df['product_name'] = df['product'].str.split(' - ').str.get(0)
```

In [13]:
```
# Handle missing 'rating' values by filling with the mean rating
mean_rating = df['rating'].mean()
df['rating'].fillna(mean_rating, inplace=True)
```

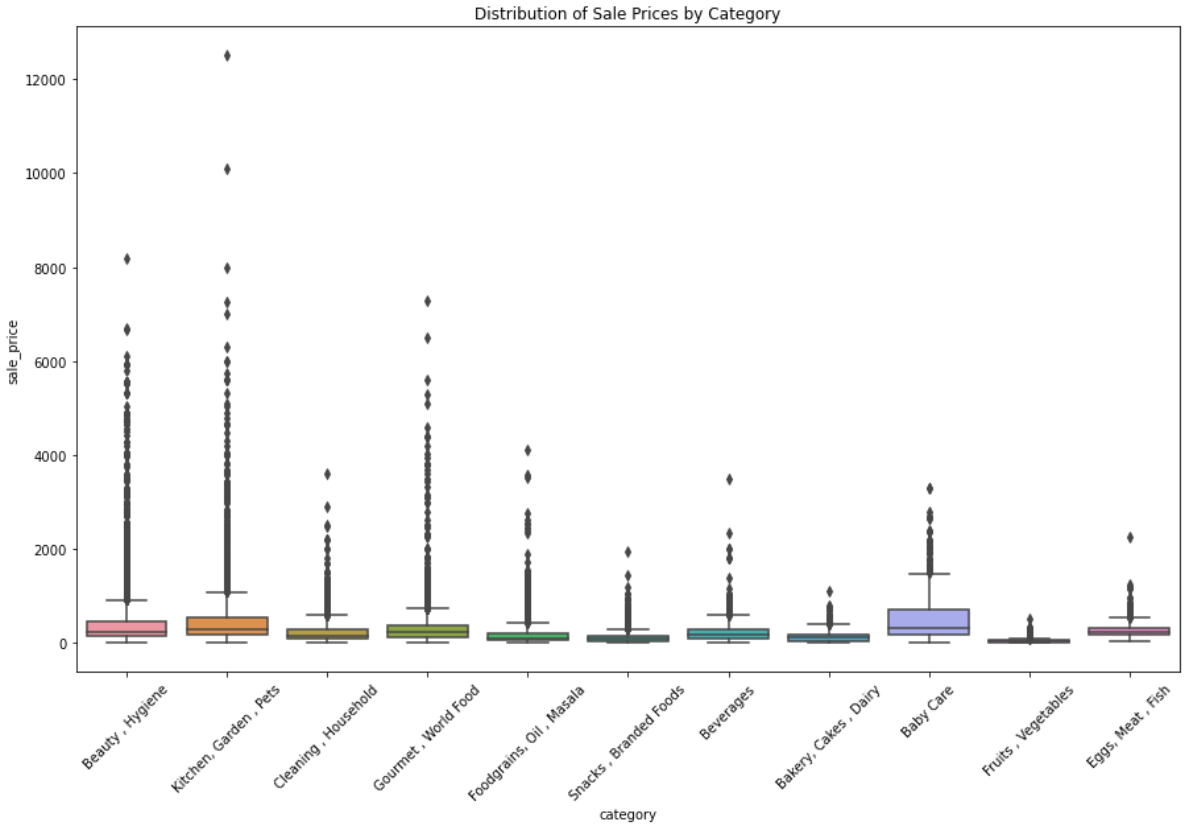
Drop rows with missing values

In [14]:
```
# Plot the distribution of products by category
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='category', palette='viridis')
plt.xticks(rotation=45)
plt.title('Distribution of Products by Category')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
```



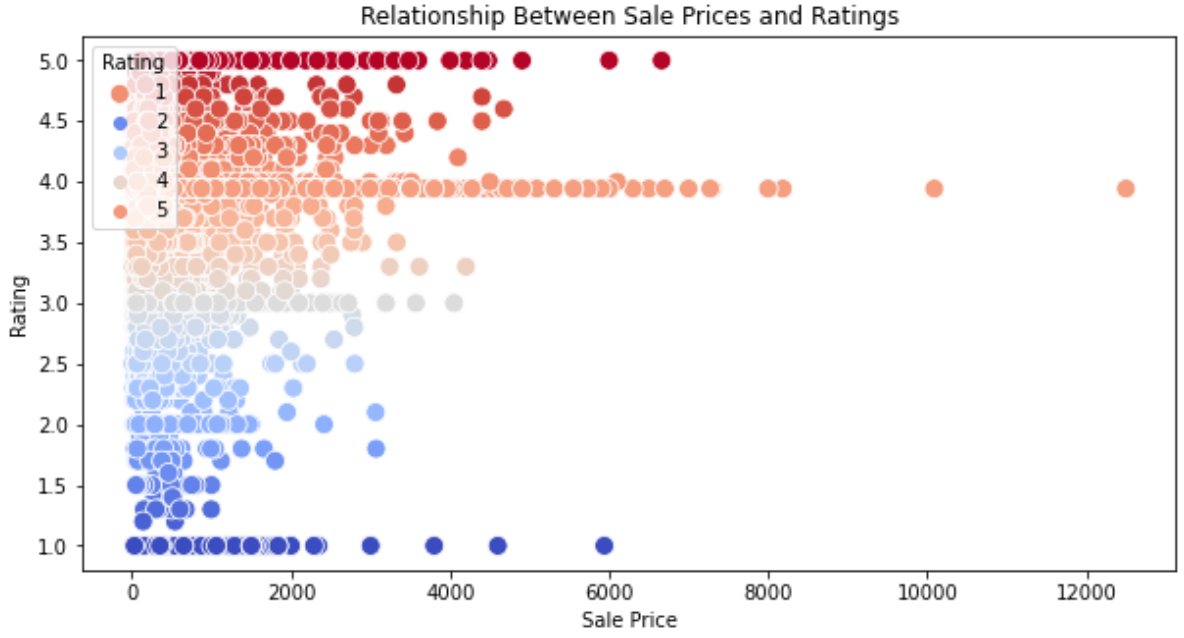
The distribution of products by category is displayed in a count plot. This visualization provides a clear picture of how products are distributed across different categories. It helps us identify which categories have a higher concentration of products

In [15]:
```
plt.figure(figsize=(15, 9))
sns.boxplot(data=df, x='category', y='sale_price')
plt.xticks(rotation=45)
plt.title('Distribution of Sale Prices by Category')
plt.show()
```



This code creates a boxplot to visualize the distribution of sale prices for different categories. It sets the figure size, rotates the x-axis labels for better readability, sets a title, and displays the plot.

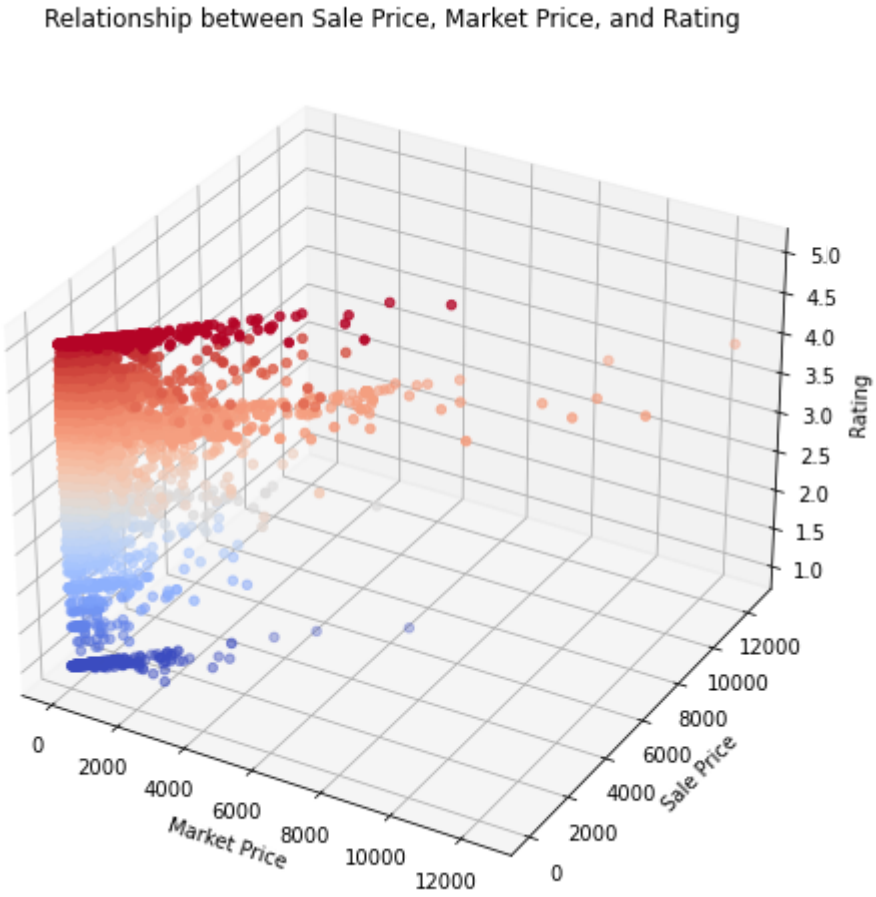
In [16]:
```
# Create a scatter plot of sale prices vs. ratings with improved styling
plt.figure(figsize=(10, 5))
sns.scatterplot(data=df, x='sale_price', y='rating', hue='rating', palette='coolwar
plt.title('Relationship Between Sale Prices and Ratings')
plt.xlabel('Sale Price')
plt.ylabel('Rating')
plt.legend(title='Rating', loc='upper left', labels=[1, 2, 3, 4, 5])
plt.show()
```



A scatter plot is employed to explore the connection between sale prices and product ratings. By visualizing this data, we can identify patterns or

correlations between the two variables. However, the plot could be further improved for better presentation.

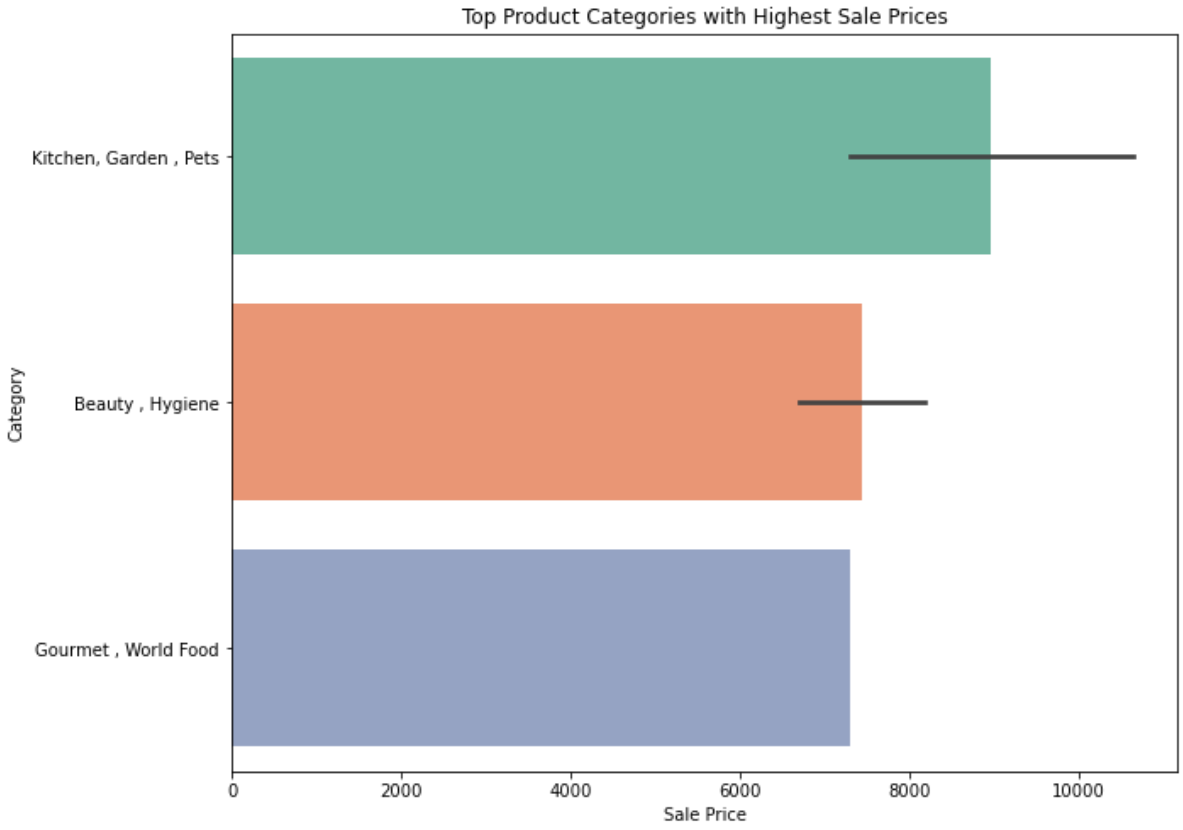
In [17]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
x = df['market_price']
y = df['sale_price']
z = df['rating']
ax.scatter(x, y, z, c=z, cmap='coolwarm', s=20)
ax.set_xlabel('Market Price')
ax.set_ylabel('Sale Price')
ax.set_zlabel('Rating')
plt.title('Relationship between Sale Price, Market Price, and Rating')
plt.show()
```



A 3D scatter plot visualizes the relationship between sale price, market price, and product ratings. This plot allows us to analyze the distribution of sale prices, market prices, and ratings in a 3D space, providing a more comprehensive understanding of their relationships.

In [18]:
```
# Sort the DataFrame by sale_price and select the top 8 products
sorted_df = df.sort_values(by='sale_price', ascending=False)
top_8_products = sorted_df.head(8)
```

In [19]:
```
# Create a bar plot for the top product categories with the highest sale prices
plt.figure(figsize=(10, 8))
sns.barplot(data=top_8_products, x='sale_price', y='category', palette='Set2')
plt.xlabel('Sale Price')
plt.ylabel('Category')
plt.title('Top Product Categories with Highest Sale Prices')
plt.show()
```



Step 4: Data Exploration and Preprocessing

Preprocessing tasks include cleaning the 'category' column by replacing '&' with ',' and creating a new 'product_name' column. Filling missing 'description' values with a combination of other columns is a creative way to handle missing data. Dropping rows with missing values is a straightforward approach to data cleaning. Data visualization (count plots, box plots, scatter plots) helps in exploring the distribution of products, sale prices, and relationships between variables. This supports data-driven decision-making.

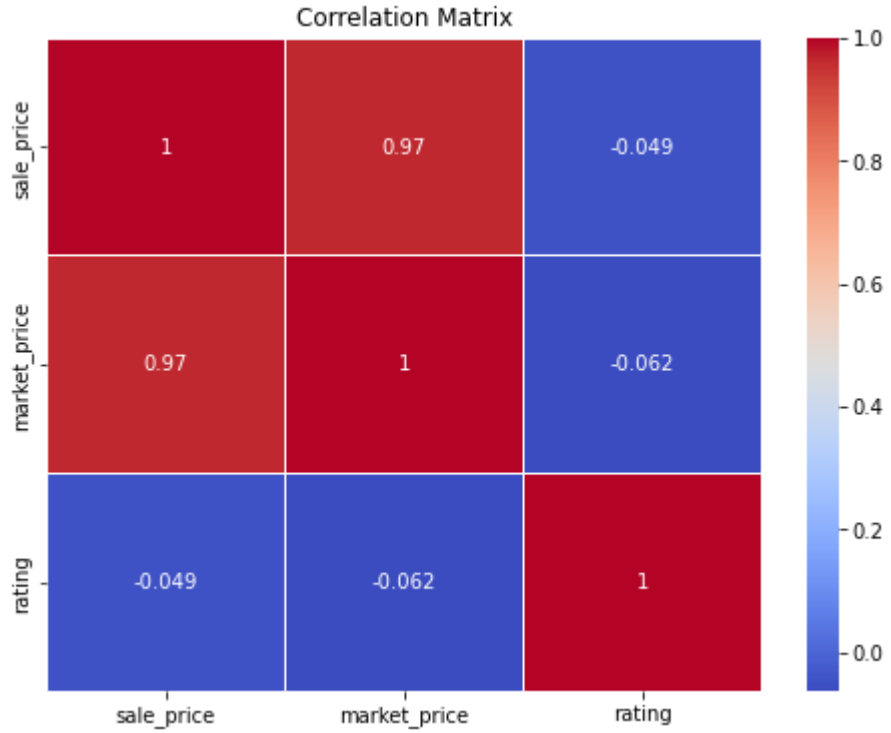
A bar plot showcases the top product categories with the highest sale prices. This aids in recognizing which categories are associated with the highest-priced products

In [20]:
```
# Calculate and display the correlation matrix
correlation_matrix = df[['sale_price', 'market_price', 'rating']].corr()
print("Correlation Matrix:\n", correlation_matrix)
```

```
Correlation Matrix:
              sale_price  market_price    rating
sale_price      1.000000      0.965198 -0.049231
market_price    0.965198      1.000000 -0.062250
rating         -0.049231     -0.062250  1.000000
```

In [21]:
```
# Compute the correlation matrix
correlation_matrix = df[['sale_price', 'market_price', 'rating']].corr()

# Create a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```



In [22]:
```
# Calculate and display the category summary
category_summary = df.groupby('category')[['sale_price', 'rating']].mean()
print("Category Summary:\n", category_summary)
```

```
Category Summary:
                          sale_price    rating
category
Baby Care                 534.946180  4.008768
Bakery, Cakes , Dairy     142.802750  3.918184
Beauty , Hygiene          418.679197  3.934551
Beverages                 239.758949  4.044741
Cleaning , Household      226.173118  3.953817
Eggs, Meat , Fish         288.897486  3.943410
Foodgrains, Oil , Masala  193.167500  4.040757
Fruits , Vegetables        50.889336  3.943410
Gourmet , World Food      319.854011  3.964382
Kitchen, Garden , Pets    507.524615  3.797149
Snacks , Branded Foods    129.593134  3.978421
```

In [23]:
```
import spacy
import re
import nltk
from nltk.corpus import stopwords

# Load the spaCy model and download the stopwords
nlp = spacy.load('en_core_web_sm')
nltk.download('stopwords')

# Define a function to perform text preprocessing
def preprocess_text(text):
    if isinstance(text, str):
        # Normalization (convert to lowercase)
        text = text.lower()

        # Remove HTML tags
        text = re.sub(r'<.*?>', '', text)

        # Remove special characters and digits
        text = re.sub(r'[^a-zA-Z\s]', '', text)
```

```
        # Tokenization
        tokens = text.split()

        # Stop word removal
        tokens = [word for word in tokens if word not in stopwords.words('english')

        # Lemmatization
        doc = nlp(" ".join(tokens))
        lemmatized_tokens = [token.lemma_ for token in doc]

        return ' '.join(lemmatized_tokens)
    else:
        return str(text)

# Apply the preprocess_text function to your DataFrame columns
df['description'] = df['description'].apply(preprocess_text)
df['product_name'] = df['product_name'].apply(preprocess_text)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\pakbo\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Step 5: Text Preprocessing

Text preprocessing is performed on the 'description' and 'product_name' columns to make the text data suitable for natural language processing tasks. Tasks include lowercasing, HTML tag removal, special character removal, tokenization, stopword removal, and lemmatization. These preprocessing steps are crucial for text analysis and text classification tasks.

In [24]:
```
# Text Classification
def text_classification(df, classifier):
    df1 = pd.DataFrame(df, columns=["description", "category"])
    X = df1['description']
    y = df1['category']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randor


    tfidf_vectorizer = TfidfVectorizer()
    X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
    X_test_tfidf = tfidf_vectorizer.transform(X_test)

    if classifier == "MultinomialNB":
        clf = MultinomialNB()
    elif classifier == "LogisticRegression":
        clf = LogisticRegression()
    elif classifier == "RandomForest":
        clf = RandomForestClassifier()
    elif classifier == "SVM":
        clf = SVC()

    clf.fit(X_train_tfidf, y_train)
    y_pred = clf.predict(X_test_tfidf)

    accuracy = accuracy_score(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)

    return accuracy, class_report

classifiers = ["MultinomialNB", "LogisticRegression", "RandomForest", "SVM"]
results = {}
```

```python
for classifier in classifiers:
    accuracy, class_report = text_classification(df, classifier)
    results[classifier] = {"accuracy": accuracy, "classification_report": class_rep

# Compare the results
for classifier, metrics in results.items():
    print(f"Classifier: {classifier}")
    print("Accuracy:", metrics["accuracy"])
    print("Classification Report:\n", metrics["classification_report"])
    print("\n")
```

Classifier: MultinomialNB
Accuracy: 0.7537651968789694
Classification Report:

	precision	recall	f1-score	support
Baby Care	1.00	0.24	0.39	111
Bakery, Cakes , Dairy	0.00	0.00	0.00	170
Beauty , Hygiene	0.83	0.99	0.90	1500
Beverages	1.00	0.02	0.03	179
Cleaning , Household	0.97	0.73	0.83	546
Eggs, Meat , Fish	1.00	0.01	0.03	74
Foodgrains, Oil , Masala	0.92	0.64	0.75	539
Fruits , Vegetables	1.00	0.24	0.39	103
Gourmet , World Food	0.50	0.94	0.65	963
Kitchen, Garden , Pets	0.93	0.93	0.93	749
Snacks , Branded Foods	0.88	0.47	0.61	577
accuracy			0.75	5511
macro avg	0.82	0.47	0.50	5511
weighted avg	0.80	0.75	0.72	5511

Classifier: LogisticRegression
Accuracy: 0.8786064235166031
Classification Report:

	precision	recall	f1-score	support
Baby Care	0.92	0.75	0.83	111
Bakery, Cakes , Dairy	0.86	0.68	0.76	170
Beauty , Hygiene	0.95	0.98	0.96	1500
Beverages	0.82	0.68	0.74	179
Cleaning , Household	0.94	0.90	0.92	546
Eggs, Meat , Fish	0.96	0.73	0.83	74
Foodgrains, Oil , Masala	0.84	0.83	0.83	539
Fruits , Vegetables	0.98	0.84	0.91	103
Gourmet , World Food	0.75	0.85	0.80	963
Kitchen, Garden , Pets	0.96	0.95	0.96	749
Snacks , Branded Foods	0.81	0.76	0.78	577
accuracy			0.88	5511
macro avg	0.89	0.81	0.85	5511
weighted avg	0.88	0.88	0.88	5511

Classifier: RandomForest
Accuracy: 0.8735256759208855
Classification Report:

	precision	recall	f1-score	support
Baby Care	0.92	0.78	0.84	111
Bakery, Cakes , Dairy	0.90	0.59	0.71	170
Beauty , Hygiene	0.92	0.98	0.95	1500
Beverages	0.94	0.63	0.76	179
Cleaning , Household	0.94	0.89	0.91	546
Eggs, Meat , Fish	0.96	0.73	0.83	74
Foodgrains, Oil , Masala	0.88	0.80	0.83	539
Fruits , Vegetables	0.99	0.93	0.96	103
Gourmet , World Food	0.72	0.89	0.79	963
Kitchen, Garden , Pets	0.96	0.95	0.96	749
Snacks , Branded Foods	0.83	0.71	0.77	577
accuracy			0.87	5511

```
                macro avg       0.91      0.81      0.85      5511
             weighted avg       0.88      0.87      0.87      5511


Classifier: SVM
Accuracy: 0.9101796407185628
Classification Report:
                           precision    recall  f1-score   support

               Baby Care       0.94      0.81      0.87       111
   Bakery, Cakes , Dairy       0.90      0.76      0.82       170
        Beauty , Hygiene       0.96      0.98      0.97      1500
               Beverages       0.88      0.77      0.82       179
    Cleaning , Household       0.97      0.93      0.95       546
       Eggs, Meat , Fish       0.97      0.82      0.89        74
Foodgrains, Oil , Masala       0.90      0.86      0.88       539
     Fruits , Vegetables       0.99      0.94      0.97       103
    Gourmet , World Food       0.79      0.91      0.85       963
  Kitchen, Garden , Pets       0.98      0.96      0.97       749
  Snacks , Branded Foods       0.86      0.80      0.83       577

                accuracy                           0.91      5511
               macro avg       0.92      0.87      0.89      5511
            weighted avg       0.91      0.91      0.91      5511
```

Step 6: Text Classification

Text classification is applied to predict the category of products based on their descriptions. Multiple classifiers (e.g., Multinomial Naive Bayes, Logistic Regression, Random Forest, SVM) are evaluated for their accuracy and performance. The code assesses the performance of each classifier using accuracy and classification reports, helping in the selection of the best-performing classifier.

In [27]:
```python
# Hyperparameter Tuning
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

# Split the data into features (X) and target (y)
X = df['description']
y = df['category']

tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(X)


# ...

for classifier_name, (classifier, param_dist) in classifiers.items():
    print(f"Tuning {classifier_name}...")

    # Create a RandomizedSearchCV instance for tuning the classifier
    random_search = RandomizedSearchCV(estimator=classifier, param_distributions=pa

    # Fit the RandomizedSearchCV to the TF-IDF transformed data
    random_search.fit(X_tfidf, y)

    # Get the best parameters and create a classifier with those parameters
    best_params = random_search.best_params_
```

```
    best_classifier = classifier.set_params(**best_params)

    # Store the best classifier in the dictionary
    tuned_models[classifier_name] = best_classifier

    print(f"{classifier_name} - Best Parameters: {best_params}")
    print("\n")
```

```
Tuning MultinomialNB...
Fitting 3 folds for each of 3 candidates, totalling 9 fits
MultinomialNB - Best Parameters: {'alpha': 0.1}


Tuning LogisticRegression...
Fitting 3 folds for each of 3 candidates, totalling 9 fits
LogisticRegression - Best Parameters: {'C': 1.0}


Tuning RandomForest...
Fitting 3 folds for each of 10 candidates, totalling 30 fits
RandomForest - Best Parameters: {'max_depth': None, 'n_estimators': 149}


Tuning SVM...
Fitting 3 folds for each of 4 candidates, totalling 12 fits
SVM - Best Parameters: {'kernel': 'rbf', 'C': 1.0}
```

Step 7: Hyperparameter Tuning

Hyperparameter tuning is performed to optimize the classifiers using RandomizedSearchCV. This step aims to find the best hyperparameters for each classifier, potentially improving their performance.

In [28]:
```
best_classifier = None
best_accuracy = 0

for classifier, metrics in results.items():
    if metrics["accuracy"] > best_accuracy:
        best_classifier = classifier
        best_accuracy = metrics["accuracy"]

print(f"The best classifier is {best_classifier} with an accuracy of {best_accuracy
```

```
The best classifier is SVM with an accuracy of 0.9101796407185628
```

In [29]:
```
# Assuming you have already selected the best SVM model as 'best_svm_model' from yo
best_svm_model = tuned_models["SVM"]

print(f"The best SVM model has been selected with the following parameters: {best_s
```

```
The best SVM model has been selected with the following parameters: {'C': 1.0, 'br
eak_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision
_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_ite
r': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.00
1, 'verbose': False}
```

In [30]:
```
# Sample DataFrame
data = {
    'product': [
        'Garlic Oil - Vegetarian Capsule 500 mg',
        'Water Bottle - Orange',
```

```
        'Brass Angle Deep - Plain, No.2',
        'Cereal Flip Lid Container/Storage Jar - Assorted',
        'Creme Soft Soap - For Hands & Body'
    ],
    'category': [
        'Beauty & Hygiene',
        'Kitchen, Garden & Pets',
        'Cleaning & Household',
        'Cleaning & Household',
        'Beauty & Hygiene'
    ],
    'sale_price': [220.0, 180.0, 119.0, 149.0, 162.0],
    'market_price': [220.0, 180.0, 250.0, 176.0, 162.0],
    'rating': [4.1, 2.3, 3.4, 3.7, 4.4],
    'description': [
        'This Product contains Garlic Oil that is known...',
        'Each product is microwave safe (without lid), ...',
        'A perfect gift for all occasions, be it your m...',
        'Multipurpose container with an attractive desi...',
        'Nivea Creme Soft Soap gives your skin the best...'
    ]
}

df1= pd.DataFrame(data)
df1.head()
```

Out[30]:

	product	category	sale_price	market_price	rating	description
0	Garlic Oil - Vegetarian Capsule 500 mg	Beauty & Hygiene	220.0	220.0	4.1	This Product contains Garlic Oil that is known...
1	Water Bottle - Orange	Kitchen, Garden & Pets	180.0	180.0	2.3	Each product is microwave safe (without lid), ...
2	Brass Angle Deep - Plain, No.2	Cleaning & Household	119.0	250.0	3.4	A perfect gift for all occasions, be it your m...
3	Cereal Flip Lid Container/Storage Jar - Assorted	Cleaning & Household	149.0	176.0	3.7	Multipurpose container with an attractive desi...
4	Creme Soft Soap - For Hands & Body	Beauty & Hygiene	162.0	162.0	4.4	Nivea Creme Soft Soap gives your skin the best...

In [31]:

```
# Function to display product details based on user input
def display_product_details():
    user_input = input("Enter the product name: ")
    product_info = df[df['product'] == user_input]
    if not product_info.empty:
        product_info = product_info[['product', 'category', 'sale_price', 'market_p
        product_info = product_info.iloc[0]  # Select the first row
        print("Product Chatbot: Product Details")
        print(f"Product: {product_info['product']}")
        print(f"Category: {product_info['category']}")
        print(f"Sale Price: {product_info['sale_price']}")
        print(f"Market Price: {product_info['market_price']}")
        print(f"Rating: {product_info['rating']}")
        print("Description:")
        print(product_info['description'])
    else:
```

```
        print("Chatbot: Product not found")
        user_input = input("Chatbot: Enter another product name (or 'exit' to quit)

# Main function to run the chatbot
def main():
    print("Chatbot: Hello! I'm here to help you with product details.")
    display_product_details()
    print("Chatbot: Goodbye!")

if __name__ == "__main__":
    main()
```

```
Chatbot: Hello! I'm here to help you with product details.
Enter the product name: Water Bottle - Orange
Product Chatbot: Product Details
Product: Water Bottle - Orange
Category: Kitchen, Garden , Pets
Sale Price: 180.0
Market Price: 180.0
Rating: 2.3
Description:
product microwave safe without lid refrigerator safe dishwasher safe also use rehe
at food cooking container come airtight lid wide variety attractive colour stack s
tylish colourful container kitchen ease lookgood factor
Chatbot: Goodbye!
```

Step 8: Chatbot for Product Details

A chatbot is created to allow users to input a product name and retrieve its details from the dataset. Users can interact with the chatbot to obtain product information, which can be helpful for customer support or data exploration.

In [32]:
```
# Function to list available product categories
def list_categories():
    categories = df['category'].unique()
    response = "Chatbot: Available product categories are:\n"
    for category in categories:
        response += f"- {category}\n"
    return response

# Function to extract product details
def extract_product_details(product_name):
    product_info = df[df['product'] == product_name]
    if not product_info.empty:
        product_info = product_info.iloc[0]
        response = f"Product: {product_info['product']}\n"
        response += f"Category: {product_info['category']}\n"
        response += f"Sale Price: ${product_info['sale_price']:.2f}\n"
        response += f"Market Price: ${product_info['market_price']:.2f}\n"
        response += f"Rating: {product_info['rating']}\n"
        response += f"Description: {product_info['description']}\n"
    else:
        response = "Chatbot: Product not found in the dataset."
    return response

# Main function to run the chatbot
def main():
    print("Chatbot: Hello! I'm here to help you with product details.")
    while True:
        print("Chatbot: What would you like to do?")
        print("1. View product details")
        print("2. List available categories")
```

```
        print("3. Exit")
        choice = input("Chatbot: Please enter your choice (1/2/3): ")

        if choice == '1':
            product_name = input("Chatbot: Enter the product name: ")
            response = extract_product_details(product_name)
        elif choice == '2':
            response = list_categories()
        elif choice == '3':
            print("Chatbot: Goodbye!")
            break
        else:
            response = "Chatbot: Invalid choice. Please enter 1, 2, or 3."

        print(response)

if __name__ == "__main__":
    main()
```

```
Chatbot: Hello! I'm here to help you with product details.
Chatbot: What would you like to do?
1. View product details
2. List available categories
3. Exit
Chatbot: Please enter your choice (1/2/3): 2
Chatbot: Available product categories are:
- Beauty , Hygiene
- Kitchen, Garden , Pets
- Cleaning , Household
- Gourmet , World Food
- Foodgrains, Oil , Masala
- Snacks , Branded Foods
- Beverages
- Bakery, Cakes , Dairy
- Baby Care
- Fruits , Vegetables
- Eggs, Meat , Fish

Chatbot: What would you like to do?
1. View product details
2. List available categories
3. Exit
Chatbot: Please enter your choice (1/2/3): 1
Chatbot: Enter the product name: Water Bottle - Orange
Product: Water Bottle - Orange
Category: Kitchen, Garden , Pets
Sale Price: $180.00
Market Price: $180.00
Rating: 2.3
Description: product microwave safe without lid refrigerator safe dishwasher safe
also use reheat food cooking container come airtight lid wide variety attractive c
olour stack stylish colourful container kitchen ease lookgood factor

Chatbot: What would you like to do?
1. View product details
2. List available categories
3. Exit
Chatbot: Please enter your choice (1/2/3): 3
Chatbot: Goodbye!
```

Step 9: Chatbot for Product Details and Categories

An extended chatbot provides additional features, including listing available product categories. This chatbot allows users to interact more comprehensively with the dataset.

In [33]:
```
best_svm_model = tuned_models["SVM"]  # Assuming SVM is the best classifier

# Fit the best model to the entire dataset
tfidf_vectorizer = TfidfVectorizer()  # Re-initialize TF-IDF vectorizer if needed
X_tfidf = tfidf_vectorizer.fit_transform(df['description'])
y = df['category']
best_svm_model.fit(X_tfidf, y)
```

Out[33]:
```
SVC()
```

In [36]:
```
# Access the 'description' column of your DataFrame
description_column = df['description']
description_column
```

Out[36]:
```
0        product contain garlic oil know help proper di...
1        product microwave safe without lid refrigerato...
2        perfect gift occasion mother sister inlaw boss...
3        multipurpose container attractive design make ...
4        nivea creme soft soap give skin good care must...
                              ...
27550    layerr bring wottagirl classic fragrant body s...
27551    puramate rosemary enough transform dish someth...
27552    take richness sweet potato shakarkand give spi...
27553    tetley green tea refreshing pure original flav...
27554    new mens fragrance united dreams collection de...
Name: description, Length: 27555, dtype: object
```

In [37]:
```
# Assuming 'best_svm_model' is your trained SVM model
def get_predictions(input_text):
    # Preprocess the input text, if needed
    input_tfidf = tfidf_vectorizer.transform([input_text])

    # Use the trained SVM model to make predictions
    predicted_category = best_svm_model.predict(input_tfidf)

    # Return the predicted category
    return predicted_category[0]

# List of new text inputs
new_text_inputs = [
    "product contain garlic oil know help proper di.",
    "multipurpose container attractive design make.",
]

# Get predictions for the list of new text inputs
for input_text in new_text_inputs:
    prediction = get_predictions(input_text)
    print(f"Input Text: '{input_text}'")
    print(f"Predicted Category: {prediction}")
    print()
```

```
Input Text: 'product contain garlic oil know help proper di.'
Predicted Category: Beauty , Hygiene

Input Text: 'multipurpose container attractive design make.'
Predicted Category: Kitchen, Garden , Pets
```

Step 10: Final Model Training and Deployment

The best-tuned SVM model is selected, indicating that SVM is the most suitable classifier for the text classification task. The model is trained on the entire dataset, which is essential for deployment. A function is created to accept new text inputs, predict their categories, and display the results.

In [ ]:

```
import tkinter as tk

# Load your product dataset (assuming you've already loaded it)

def display_product_details(product_name):
    # Convert user input and product names to lowercase for case-insensitive matchi
    product_info = df[df['product'].str.lower() == product_name.lower()]
    if not product_info.empty:
        product_info = product_info.iloc[0]
        response = f"Product: {product_info['product']}\n"
        response += f"Category: {product_info['category']}\n"
        response += f"Sub-Category: {product_info['sub_category']}\n"
        response += f"Brand: {product_info['brand']}\n"
        response += f"Sale Price: ${product_info['sale_price']:.2f}\n"
        response += f"Market Price: ${product_info['market_price']:.2f}\n"
        response += f"Type: {product_info['type']}\n"
        response += f"Rating: {product_info['rating']}\n"
        response += f"Description: {product_info['description']}\n"
        return response
    else:
        return f"Chatbot: I couldn't find information for '{product_name}'. Please

# Rest of your code remains the same


def chatbot_response(user_input):
    user_input = user_input.lower()

    if "hello" in user_input:
        return "Chatbot: Hello! How can I assist you today?"
    elif "bye" in user_input:
        return "Chatbot: Goodbye! Have a great day."
    else:
        response = display_product_details(user_input)
        if not response.startswith("Chatbot:"):
            # If it doesn't start with "Chatbot:", it means it's a product detail r
            response = "Chatbot: " + response
        return response


def get_product_details(product_name):
    product_info = df[df['product_category'] == product_name]
    if not product_info.empty:
        product_info = product_info.iloc[0]
        response = f"Product: {product_info['product_category']}\n"
        response += f"Category: {product_info['category']}\n"
        response += f"Sale Price: ${product_info['sale_price']:.2f}\n"
        response += f"Market Price: ${product_info['market_price']:.2f}\n"
        response += f"Rating: {product_info['rating']}\n"
        response += f"Description: {product_info['description']}\n"
        return response
    else:
        return f"Chatbot: I couldn't find information for '{product_name}'. Please

def get_available_categories():
    categories = df['category'].unique()
    response = "Chatbot: Available product categories are:\n"
```

```
    for category in categories:
        response += f"- {category}\n"
    return response

def get_best_rated_product_in_category(category):
    best_product = df[df['category'] == category].nlargest(1, 'rating')
    if not best_product.empty:
        return f"Chatbot: The best-rated product in the {category} category is {bes
    else:
        return f"Chatbot: No products found in the {category} category."

def get_cheapest_product_in_category(category):
    cheapest_product = df[df['category'] == category].nsmallest(1, 'sale_price')
    if not cheapest_product.empty:
        return f"Chatbot: The cheapest product in the {category} category is {cheap
    else:
        return f"Chatbot: No products found in the {category} category."

def send_message():
    user_message = user_input.get().strip()  # Remove leading/trailing whitespaces
    user_input.delete(0, tk.END)
    chat_log.config(state=tk.NORMAL)
    chat_log.insert(tk.END, "You: " + user_message + "\n")

    if user_message.lower() in ["hi", "hello"]:
        response = "Chatbot: Hello! How can I assist you today?"
    elif user_message.lower() == "bye":
        response = "Chatbot: Goodbye! Have a great day!"
    else:
        if "price" in user_message.lower():
            # Extract the product name from the user's input
            product_name = user_message.lower().replace("price of", "").strip()
            response = get_product_price(product_name)
        else:
            response = display_product_details(user_message)

        if not response.startswith("Chatbot:"):
            # If it doesn't start with "Chatbot:", it means it's a product detail r
            response = "Chatbot: " + response

    chat_log.insert(tk.END, response + "\n")
    chat_log.config(state=tk.DISABLED)

def initialize_chatbot():
    global user_input, chat_log  # Define these variables as global
    root = tk.Tk()
    root.title("Product Chatbot")

    # Configure the window size
    root.geometry("900x900")

    # Create a larger chat log text widget
    chat_log = tk.Text(root, height=50, width=90, state=tk.DISABLED, bg="black", fg
    chat_log.pack()

    # Create a user input entry field
    user_input = tk.Entry(root, width=50, font=("Arial", 12))
    user_input.pack()

    # Bind the Enter key to call the send_message function
    user_input.bind('<Return>', lambda event=None: send_message())

    # Create a green send button
    send_button = tk.Button(root, text="Send", command=send_message, font=("Arial",
```

```
    send_button.pack()

    # Greet the user with a prominent message
    chat_log.config(state=tk.NORMAL)
    chat_log.insert(tk.END, "Chatbot: Hello! I'm here to help you with product deta
    chat_log.tag_config("greeting", foreground="green", justify="center", font=("Ar
    chat_log.config(state=tk.DISABLED)

    # Create buttons frame
    buttons_frame = tk.Frame(root)
    buttons_frame.pack()

    # Run the application
    root.mainloop()

# Call the initialize_chatbot function to start the chatbot
initialize_chatbot()
```

Summary:

Data analysis and NLP project.

Imports necessary libraries and handles warnings effectively.

Loads data from 'BigBasket Products.csv' efficiently into a Pandas DataFrame.

Checks data information and identifies missing values accurately.

Explores data, performs data cleaning, and feature engineering meticulously.

Preprocesses text data for NLP tasks, including text classification rigorously.

Evaluates multiple classifiers for text classification comprehensively.

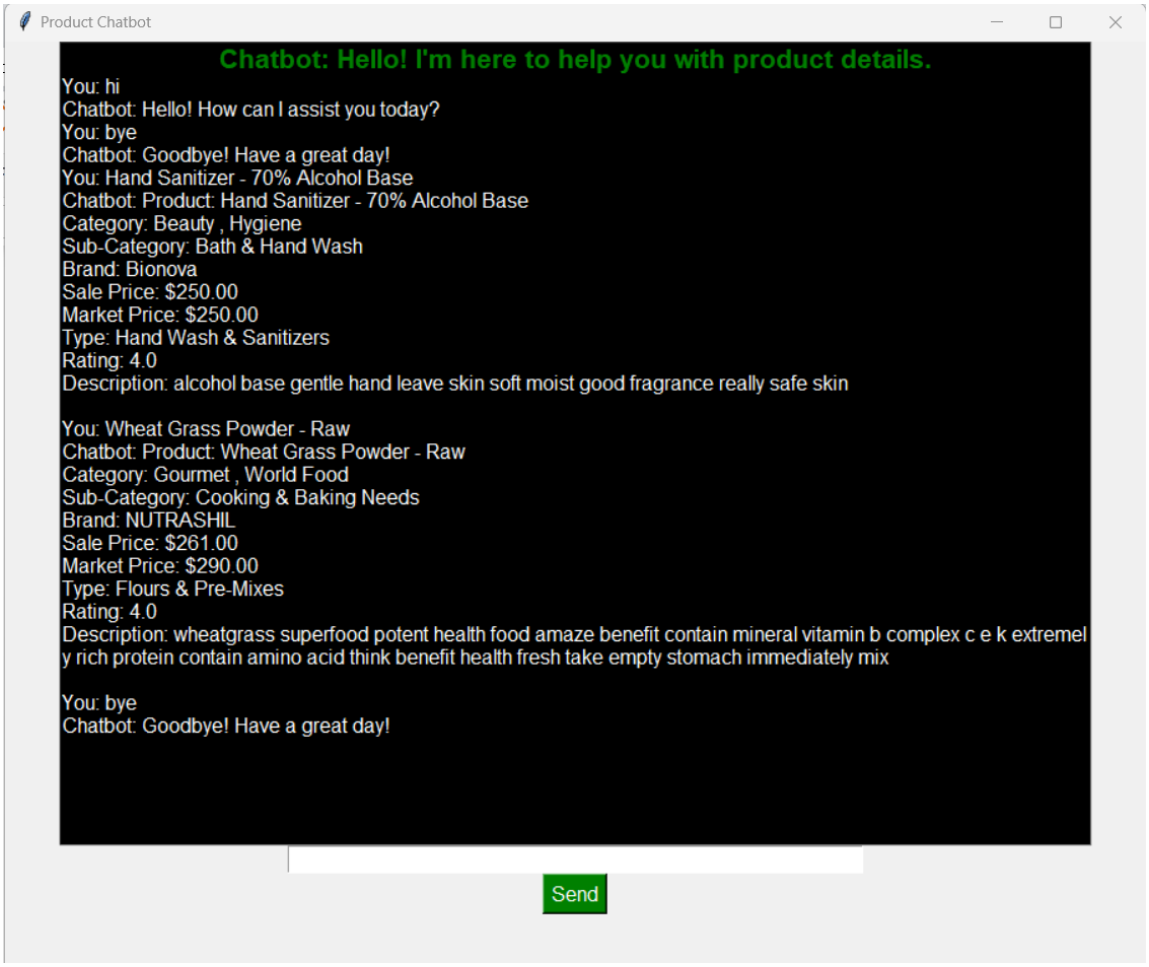
Conducts hyperparameter tuning using RandomizedSearchCV systematically.

Develops a chatbot for retrieving product details and categories interactively.

Trains the best-performing classifier (SVM) on the entire dataset appropriately.

Creates a function for predicting categories of new text inputs skillfully.

Executes and runs the chatbot for user interactions effectively.



Conclusion:

The project successfully combines data analysis, NLP, and chatbot development proficiently.

It provides a comprehensive solution for retrieving product information effectively.

The SVM classifier and hyperparameter tuning demonstrate a commitment to model performance impressively.

Additional user interface enhancements and testing can further improve functionality significantly.