In [1]:
```python
#importing required libraires
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score,classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier,RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
```

In [2]:
```python
# Load dataset
data = pd.read_csv("E_commerce dataset1.csv")
```

In [3]:
```python
data
```

Out[3]:

	Timestamp	email	age_group	gender	location	occupation	shop_frequence	brand_pref	trust_factors
0	11-02-2025 17:17	marativijaykumar51@gmail.com	18-24	Male	Hyderabad	Student	Monthly	Sometimes	Customer reviews, Social media presence, Influ...
1	11-02-2025 17:17	rajakimmoji@gmail.com	18-24	Male	Hyderabad	Student	Monthly	Yes, always	Brand name, Customer reviews, Social media pre...
2	11-02-2025 18:23	rushi888888@gmail.com	18-24	Male	Tamilnadu	Salaried Professional	Weekly	Yes, always	Brand name, Customer reviews, Discounts & offers
3	11-02-2025 19:04	palinigajalakshmi@gmail.com	18-24	Male	Vellore	Student	Occasionally	Sometimes	Brand name, Customer reviews, Discounts & offers
4	11-02-2025 19:04	anjali.2020@vitstudent.ac.in	25-34	Female	Vellore, Tamil Nadu	Student	Monthly	Yes, always	Brand name, Customer reviews, Discounts & offers
...	...	...	...	...	...	...	...	...	...
606	16-02-2025 12:04	madhuri352002@gmail.com	25-34	Female	Ahmednagar	Housewife	Weekly	Sometimes	Customer reviews
607	16-02-2025 12:02	yadavaditya1885@gmail.com	18-24	Male	Ahmednagar	Student	Monthly	Yes, always	Brand name

	Timestamp	email	age_group	gender	location	occupation	shop_frequence	brand_pref	trust_factors
608	16-02-2025 12:04	madhuri352002@gmail.com	25-34	Female	Ahmednagar	Housewife	Weekly	Sometimes	Customer reviews
609	16-02-2025 20:50	saibhargavipc@gmail.com	18-24	Female	Hyderabad	Student	Monthly	Sometimes	Brand name, Customer reviews, Influencer recom...
610	17-02-2025 12:06	sushma.murrarishetty@gmail.com	25-34	Female	Hyderabad	House wife	Occasionally	Sometimes	Customer reviews

611 rows × 22 columns

In [4]:
```
data.drop(columns=["Timestamp","email"],inplace=True,errors='ignore')
```

In [5]:
```
data.sample(5)
```

Out[5]:

	age_group	gender	location	occupation	shop_frequence	brand_pref	trust_factors	brand_loyal	brand_discovery	brand_img_im
607	18-24	Male	Ahmednagar	Student	Monthly	Yes, always	Brand name	Yes	Social media ads	
332	25-34	Male	Hanuman	Salaried Professional	Monthly	Sometimes	Discounts & offers	No	Social media ads, Search engines, E-commerce p...	
302	18-24	Female	Bengaluru	Student	Rarely	Yes, always	Brand name, Customer reviews, Discounts & offers	No	E-commerce platforms (Amazon, Flipkart, etc.)	
72	18-24	Male	Karnataka	Startup	Monthly	No, I focus on other factors	Customer reviews, Social media presence, Disco...	Yes	Social media ads, Influencer marketing	
591	25-34	Male	Telangana	Salaried Professional	Monthly	Yes, always	Brand name, Customer reviews, Influencer recom...	Yes	Search engines, E-commerce platforms (Amazon, ...	

In [6]:

```
data.describe()
```

Out[6]:

	brand_img_impact	recommendation
count	611.000000	611.000000
mean	3.175123	3.312602
std	1.176451	1.196507
min	1.000000	1.000000
25%	3.000000	3.000000
50%	3.000000	3.000000
75%	4.000000	4.000000
max	5.000000	5.000000

In [7]:

```
data.info()
```

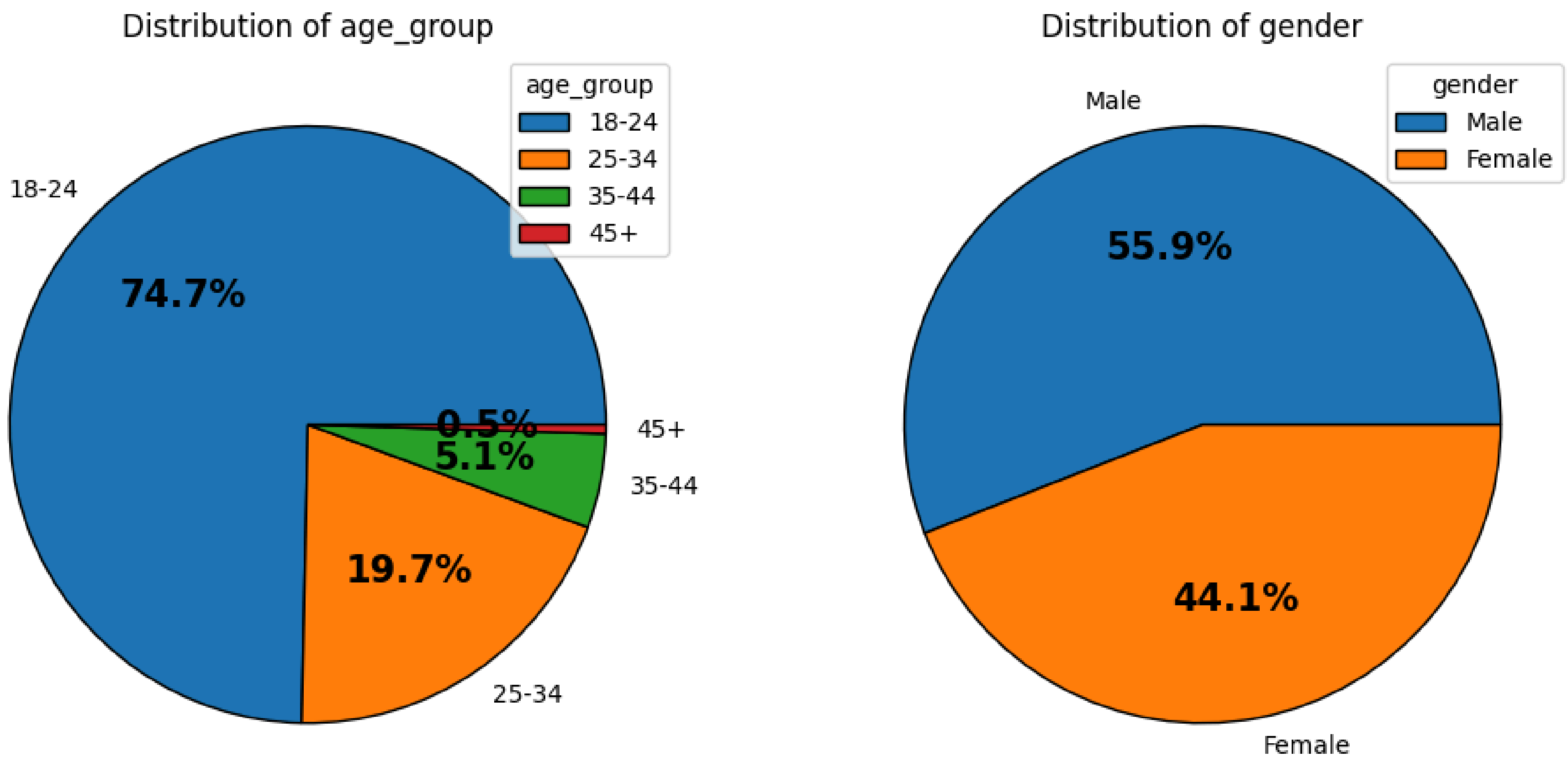
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 611 entries, 0 to 610
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   age_group         611 non-null    object
 1   gender            611 non-null    object
 2   location          611 non-null    object
 3   occupation        611 non-null    object
 4   shop_frequence    611 non-null    object
 5   brand_pref        611 non-null    object
 6   trust_factors     611 non-null    object
 7   brand_loyal       611 non-null    object
 8   brand_discovery   611 non-null    object
 9   brand_img_impact  611 non-null    int64 
 10  inf_platform      611 non-null    object
 11  stop_buying       611 non-null    object
 12  recommendation    611 non-null    int64 
 13  repeat_reasons    611 non-null    object
 14  trusted_platform  611 non-null    object
 15  switch_platforms  611 non-null    object
 16  sati_Amazon       611 non-null    object
 17  sati_flipkart     611 non-null    object
 18  sati_myntra       611 non-null    object
 19  sati_Ajio         611 non-null    object
dtypes: int64(2), object(18)
memory usage: 95.6+ KB
```

In [8]:
```
# Strip spaces from column names
data.columns = data.columns.str.strip()
```

In [9]:
```
data.drop_duplicates(inplace=True)
data.columns, data.shape
```

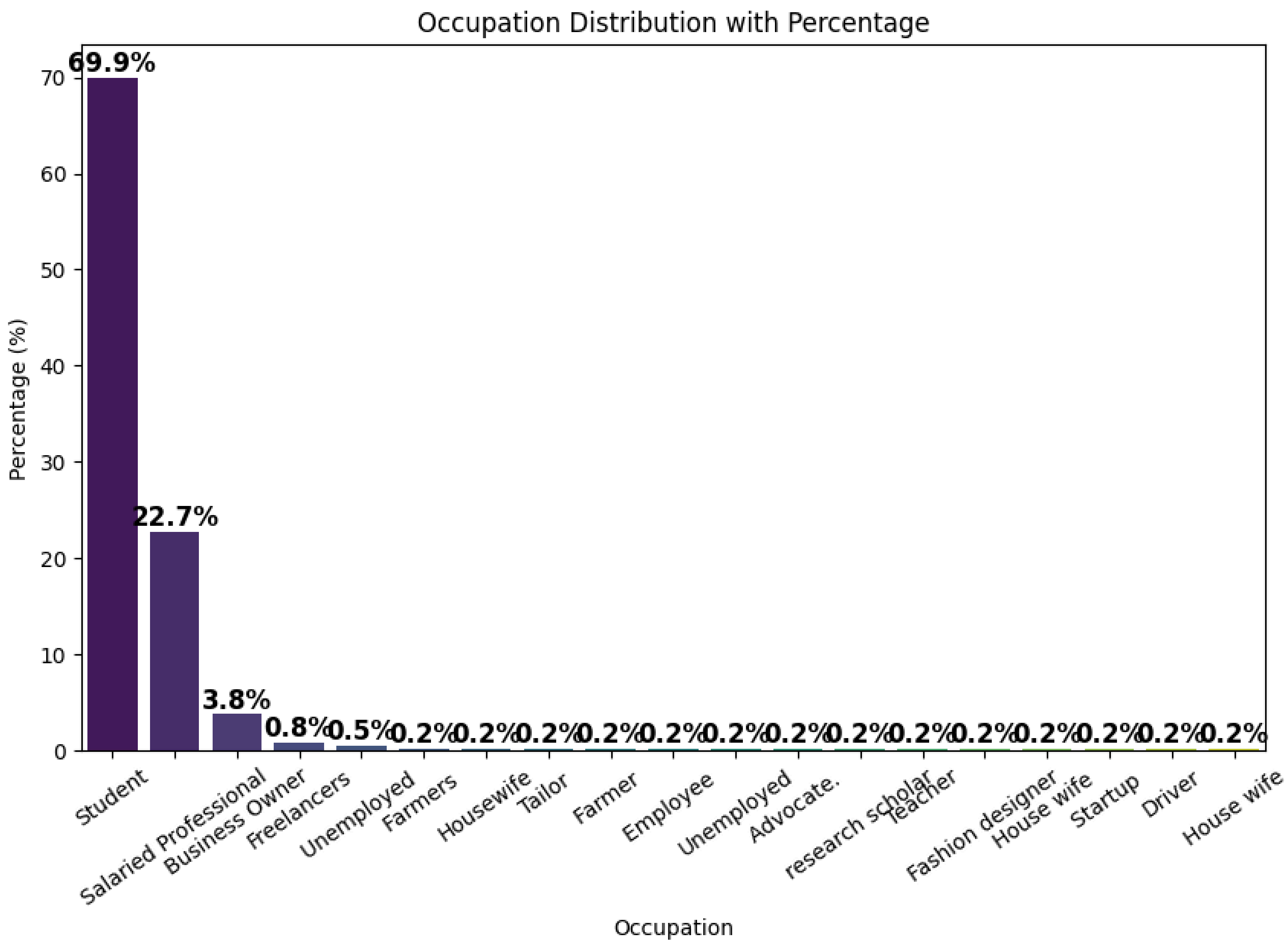
Out[9]:
```
(Index(['age_group', 'gender', 'location', 'occupation', 'shop_frequence',
        'brand_pref', 'trust_factors', 'brand_loyal', 'brand_discovery',
        'brand_img_impact', 'inf_platform', 'stop_buying', 'recommendation',
        'repeat_reasons', 'trusted_platform', 'switch_platforms', 'sati_Amazon',
        'sati_flipkart', 'sati_myntra', 'sati_Ajio'],
       dtype='object'),
 (608, 20))
```

In [10]:
```python
pie_columns = ['age_group', 'gender']

fig, ax = plt.subplots(1,2, figsize=(12,8))

for i, col in enumerate(pie_columns):
    counts = data[col].value_counts()
    labels = counts.index
    sizes = counts.values

    wedges, texts, autotexts = ax[i].pie(sizes, labels=labels, autopct='%1.1f%%',
        wedgeprops={'edgecolor': 'black'})

    for autotext in autotexts:
        autotext.set_color('black')
        autotext.set_fontsize(15)
        autotext.set_fontweight('bold')

    ax[i].legend(wedges, labels, title=col)

    ax[i].set_title(f"Distribution of {col}")

plt.show()
```



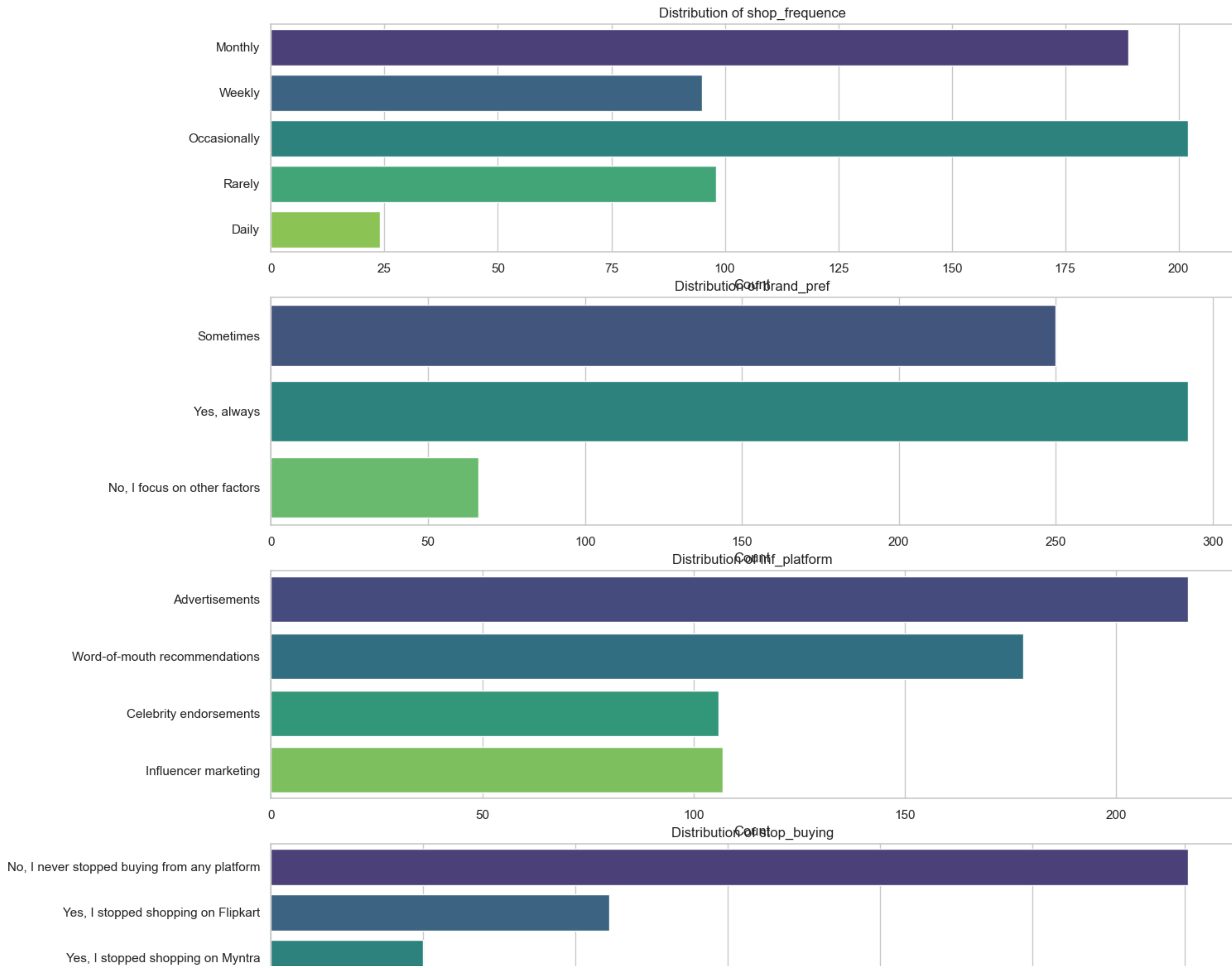
In [11]:
```python
# Calculating percentage values
occupation_counts = data['occupation'].value_counts(normalize=True) * 100

plt.figure(figsize=(10, 6))
ax = sns.barplot(x=occupation_counts.index, y=occupation_counts.values, hue=occupation_counts.index,
                 palette="viridis", legend=False)

# Adding percentage labels on top of bars
for p in ax.patches:
    ax.annotate(f'{p.get_height():.1f}%', (p.get_x() + p.get_width() / 2, p.get_height()),
                ha='center', va='bottom', fontsize=12, fontweight='bold', color='black')

plt.title("Occupation Distribution with Percentage")
```

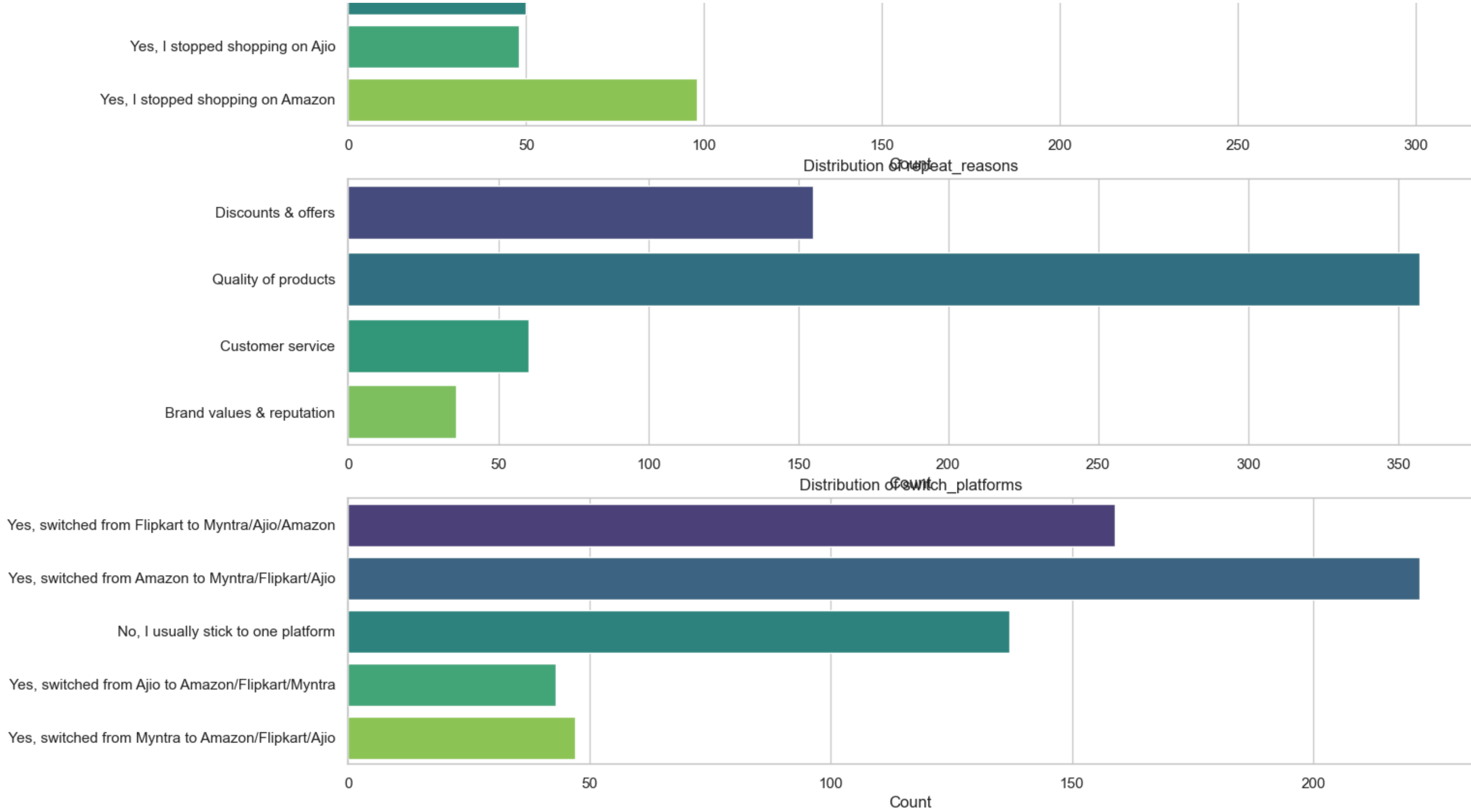
```python
plt.xlabel("Occupation")
plt.ylabel("Percentage (%)")
plt.xticks(rotation=35)
plt.show()
```




```
In [12]: sns.set(style="whitegrid")
```

```python
# Selecting categorical columns for bar chart visualization
categorical_columns = ['shop_frequence','brand_pref','inf_platform', 'stop_buying',
        'repeat_reasons', 'switch_platforms']


fig, axes = plt.subplots(6,1, figsize=(15, 25))
axes = axes.flatten()

for i, col in enumerate(categorical_columns):
    sns.countplot(y=data[col], ax=axes[i], hue=data[col], palette="viridis", legend=False)
    axes[i].set_title(f"Distribution of {col}")
    axes[i].set_xlabel("Count")
    axes[i].set_ylabel("")

plt.show()
```


Distribution of shop_frequence
Monthly
Weekly
Occasionally
Rarely
Daily
0 25 50 75 100 125 150 175 200
Count
Distribution of brand_pref
Sometimes
Yes, always
No, I focus on other factors
0 50 100 150 200 250 300
Count
Distribution of inf_platform
Advertisements
Word-of-mouth recommendations
Celebrity endorsements
Influencer marketing
0 50 100 150 200
Count
Distribution of stop_buying
No, I never stopped buying from any platform
Yes, I stopped shopping on Flipkart
Yes, I stopped shopping on Myntra



In [13]:

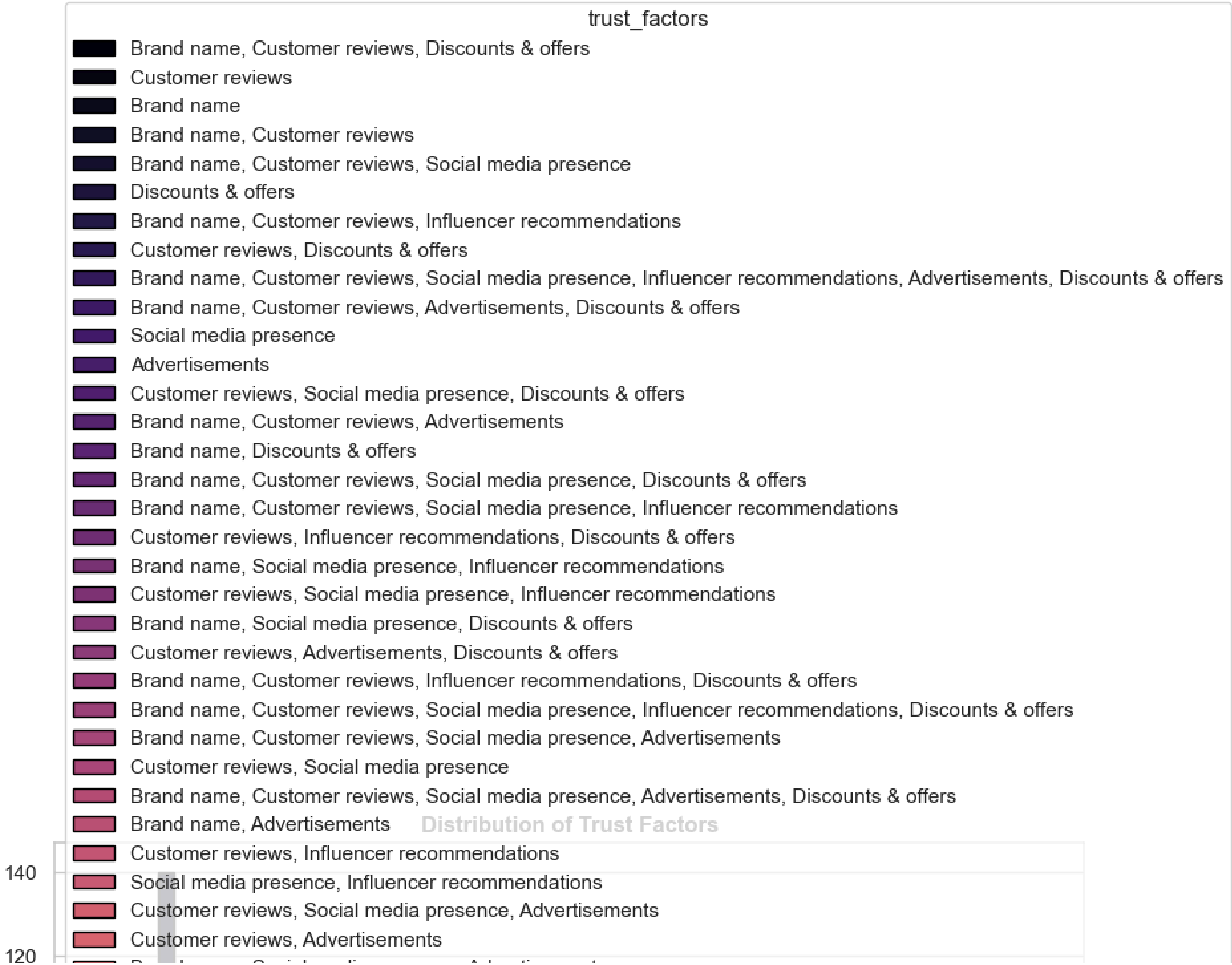
```
# Count of brand discovery sources
plt.figure(figsize=(10, 6))
sns.barplot(hue=data['brand_discovery'].value_counts().index,
            y=data['brand_discovery'].value_counts().values,
            palette="viridis", edgecolor="black")

# Add labels and title
plt.title("Brand Discovery Sources", fontsize=12, fontweight="bold")
plt.xlabel("Brand Discovery Source", fontsize=11)
plt.ylabel("Count", fontsize=11)
```

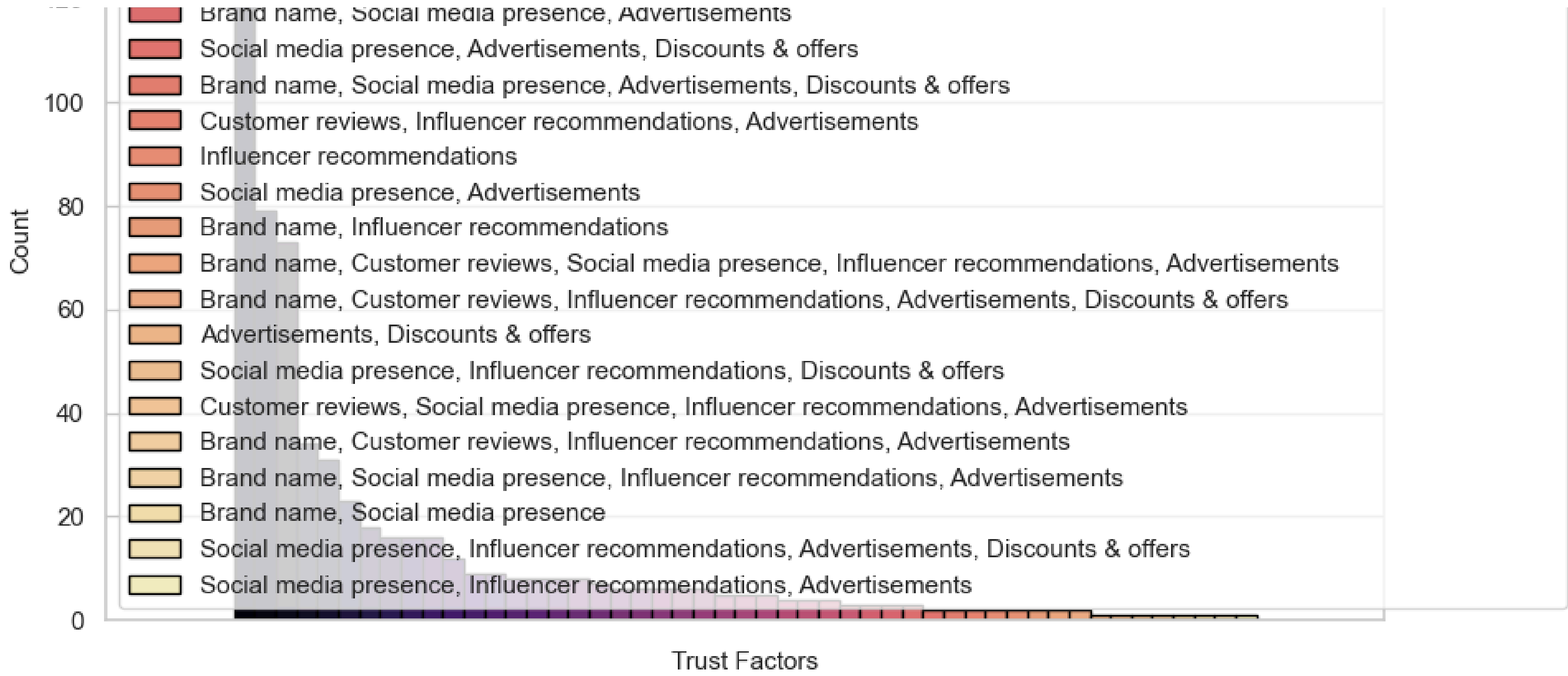
```
plt.xticks(rotation=55)

# Show plot
plt.show()
```


brand_discovery
E-commerce platforms (Amazon, Flipkart, etc.)
Social media ads
Social media ads, E-commerce platforms (Amazon, Flipkart, etc.), Friends & family
Friends & family
Social media ads, Influencer marketing, Search engines
Social media ads, E-commerce platforms (Amazon, Flipkart, etc.)
E-commerce platforms (Amazon, Flipkart, etc.), Friends & family
Social media ads, Friends & family
Influencer marketing
Social media ads, Influencer marketing, Search engines, E-commerce platforms (Amazon, Flipkart, etc.), Friends & family
Social media ads, Influencer marketing
Social media ads, Influencer marketing, E-commerce platforms (Amazon, Flipkart, etc.)
Social media ads, Influencer marketing, E-commerce platforms (Amazon, Flipkart, etc.), Friends & family
Social media ads, Search engines, E-commerce platforms (Amazon, Flipkart, etc.), Friends & family
Search engines, E-commerce platforms (Amazon, Flipkart, etc.)
Search engines
Social media ads, Search engines, E-commerce platforms (Amazon, Flipkart, etc.)
Social media ads, Search engines
Search engines, E-commerce platforms (Amazon, Flipkart, etc.), Friends & family
Social media ads, Influencer marketing, Friends & family
Influencer marketing, Search engines, E-commerce platforms (Amazon, Flipkart, etc.)
Influencer marketing, E-commerce platforms (Amazon, Flipkart, etc.)
Influencer marketing, Friends & family
Social media ads, Search engines, Friends & family
Social media ads, Influencer marketing, Search engines, E-commerce platforms (Amazon, Flipkart, etc.)
Influencer marketing, Search engines
Influencer marketing, Search engines, Friends & family
Search engines, Friends & family
Influencer marketing, E-commerce platforms (Amazon, Flipkart, etc.), Friends & family
Influencer marketing, Search engines, E-commerce platforms (Amazon, Flipkart, etc.), Friends & family
Social media ads, Influencer marketing, Search engines, Friends & family
Count
100
80
60
40
20
0

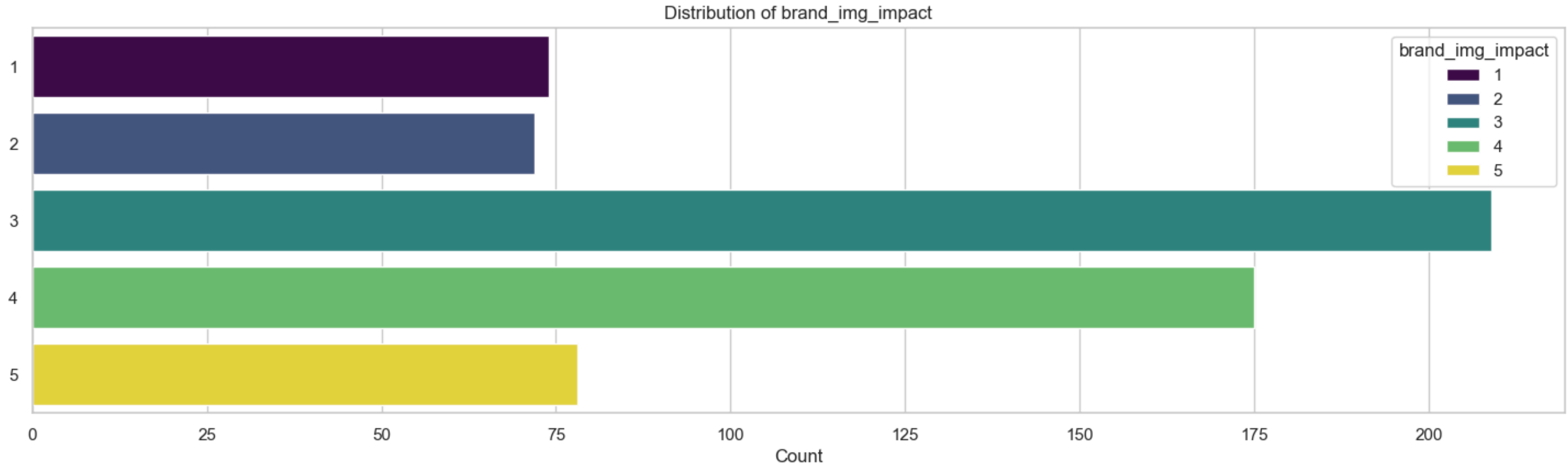
Brand Discovery Source

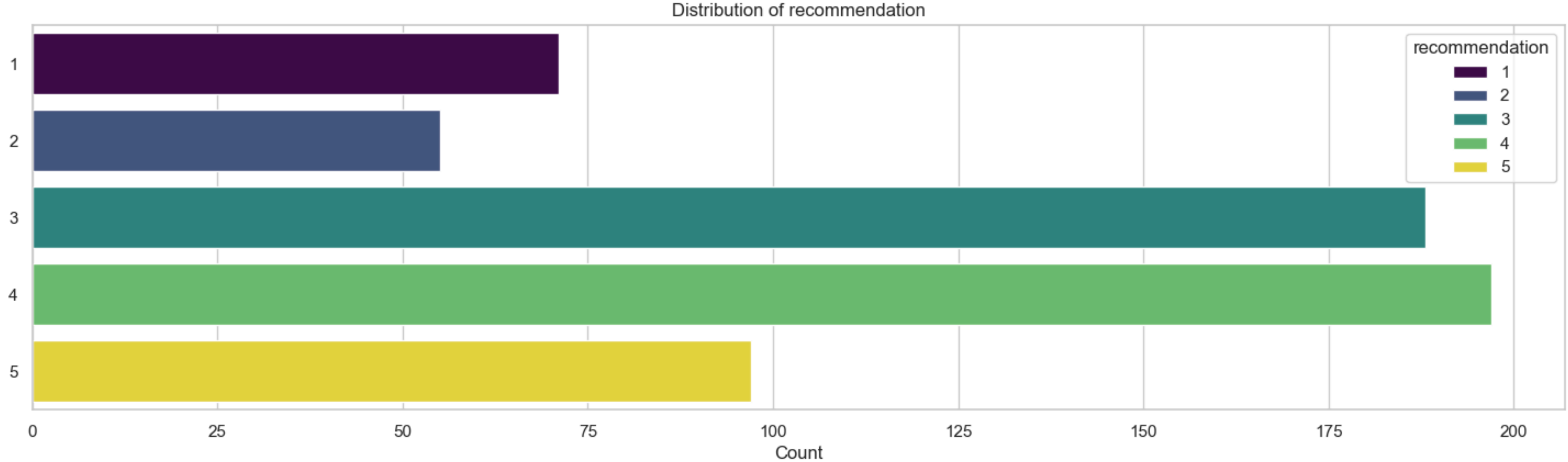
In [14]:
```
# Count of trust factors
plt.figure(figsize=(10, 6))
sns.barplot(hue=data['trust_factors'].value_counts().index,
            y=data['trust_factors'].value_counts().values,
            palette="magma", edgecolor="black")

# Add labels and title
plt.title("Distribution of Trust Factors", fontsize=12, fontweight="bold")
plt.xlabel("Trust Factors", fontsize=11)
plt.ylabel("Count", fontsize=11)
plt.xticks(rotation=15)

# Show plot
plt.show()
```


trust_factors
Brand name, Customer reviews, Discounts & offers
Customer reviews
Brand name
Brand name, Customer reviews
Brand name, Customer reviews, Social media presence
Discounts & offers
Brand name, Customer reviews, Influencer recommendations
Customer reviews, Discounts & offers
Brand name, Customer reviews, Social media presence, Influencer recommendations, Advertisements, Discounts & offers
Brand name, Customer reviews, Advertisements, Discounts & offers
Social media presence
Advertisements
Customer reviews, Social media presence, Discounts & offers
Brand name, Customer reviews, Advertisements
Brand name, Discounts & offers
Brand name, Customer reviews, Social media presence, Discounts & offers
Brand name, Customer reviews, Social media presence, Influencer recommendations
Customer reviews, Influencer recommendations, Discounts & offers
Brand name, Social media presence, Influencer recommendations
Customer reviews, Social media presence, Influencer recommendations
Brand name, Social media presence, Discounts & offers
Customer reviews, Advertisements, Discounts & offers
Brand name, Customer reviews, Influencer recommendations, Discounts & offers
Brand name, Customer reviews, Social media presence, Influencer recommendations, Discounts & offers
Brand name, Customer reviews, Social media presence, Advertisements
Customer reviews, Social media presence
Brand name, Customer reviews, Social media presence, Advertisements, Discounts & offers
Brand name, Advertisements
Distribution of Trust Factors
Customer reviews, Influencer recommendations
140
Social media presence, Influencer recommendations
Customer reviews, Social media presence, Advertisements
Customer reviews, Advertisements
120



In [15]:
```
sns.set(style="whitegrid")

# Selecting numerical columns for bar chart visualization
numerical_columns = ['brand_img_impact','recommendation']

fig, axes = plt.subplots(2,1, figsize=(18, 10))
axes = axes.flatten()

for i, col in enumerate(numerical_columns):
    sns.countplot(y=data[col], ax=axes[i],hue=data[col], palette="viridis")
    axes[i].set_title(f"Distribution of {col}")
    axes[i].set_xlabel("Count")
    axes[i].set_ylabel("")
plt.show()
```

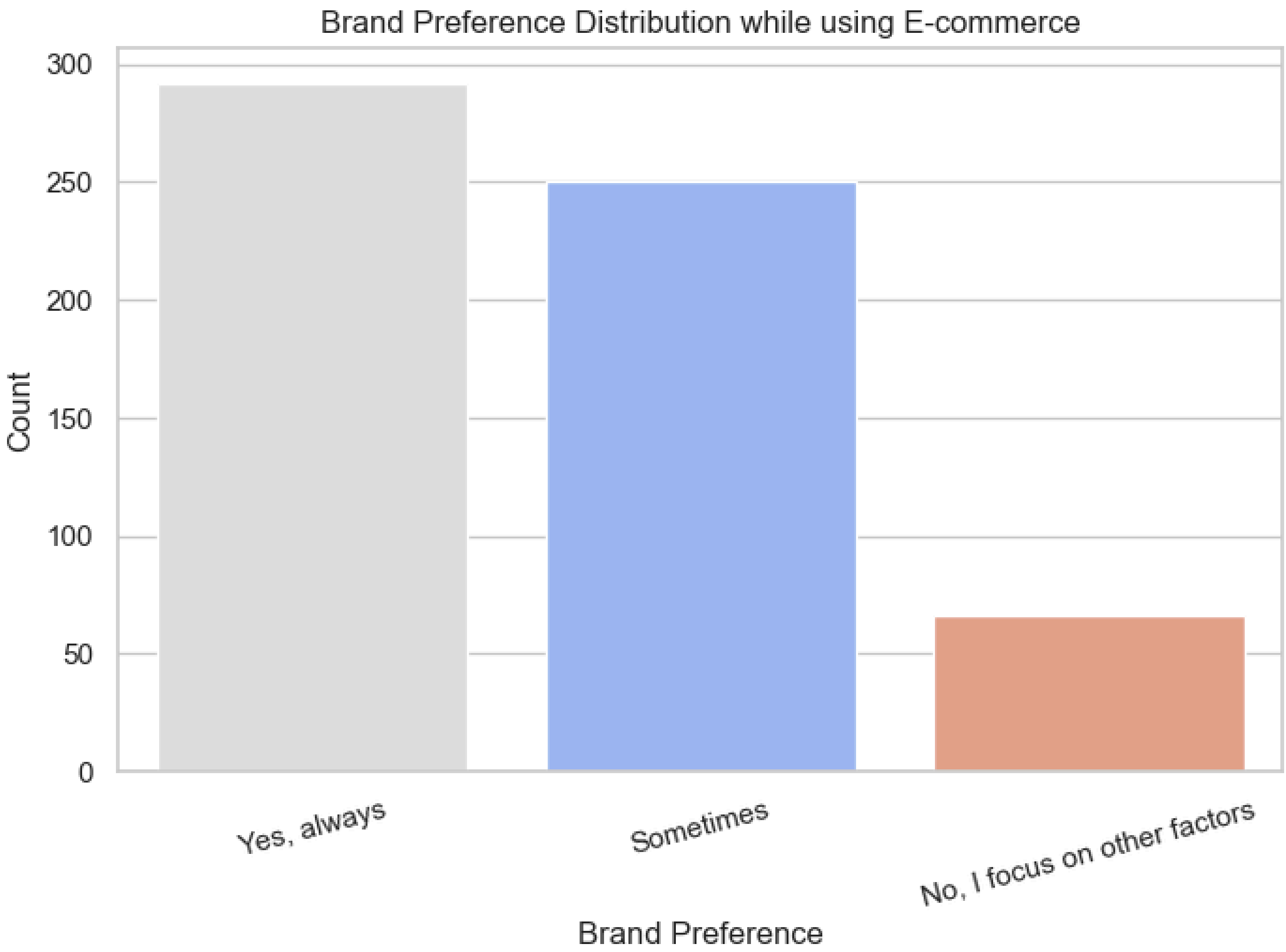




In [16]:

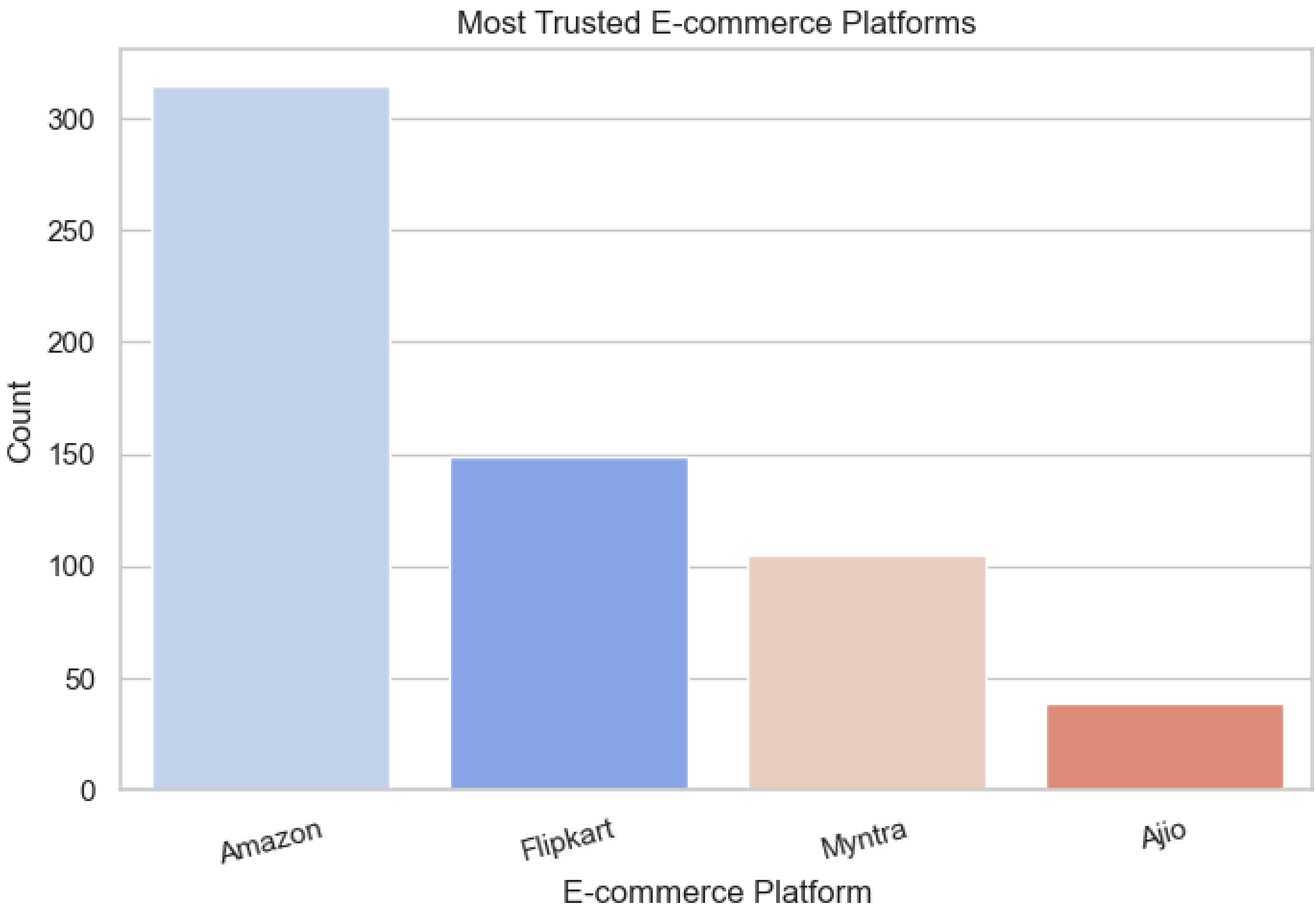
```
sns.set(style="whitegrid")

# Ploting Brand Preference Distribution
plt.figure(figsize=(8, 5))
sns.countplot(data=data, x='brand_pref', hue='brand_pref', palette='coolwarm', legend=False,
              order=data['brand_pref'].value_counts().index)
plt.title("Brand Preference Distribution while using E-commerce")
```

```
plt.xlabel("Brand Preference")
plt.ylabel("Count")
plt.xticks(rotation=15)
plt.show()
```



In [17]:
```
# Ploting Trusted E-commerce Platform
plt.figure(figsize=(8, 5))
sns.countplot(data=data, x='trusted_platform', hue='trusted_platform', palette='coolwarm', legend=False,
              order=data['trusted_platform'].value_counts().index)
```

```python
plt.title("Most Trusted E-commerce Platforms")
plt.xlabel("E-commerce Platform")
plt.ylabel("Count")
plt.xticks(rotation=15)
plt.show()
```



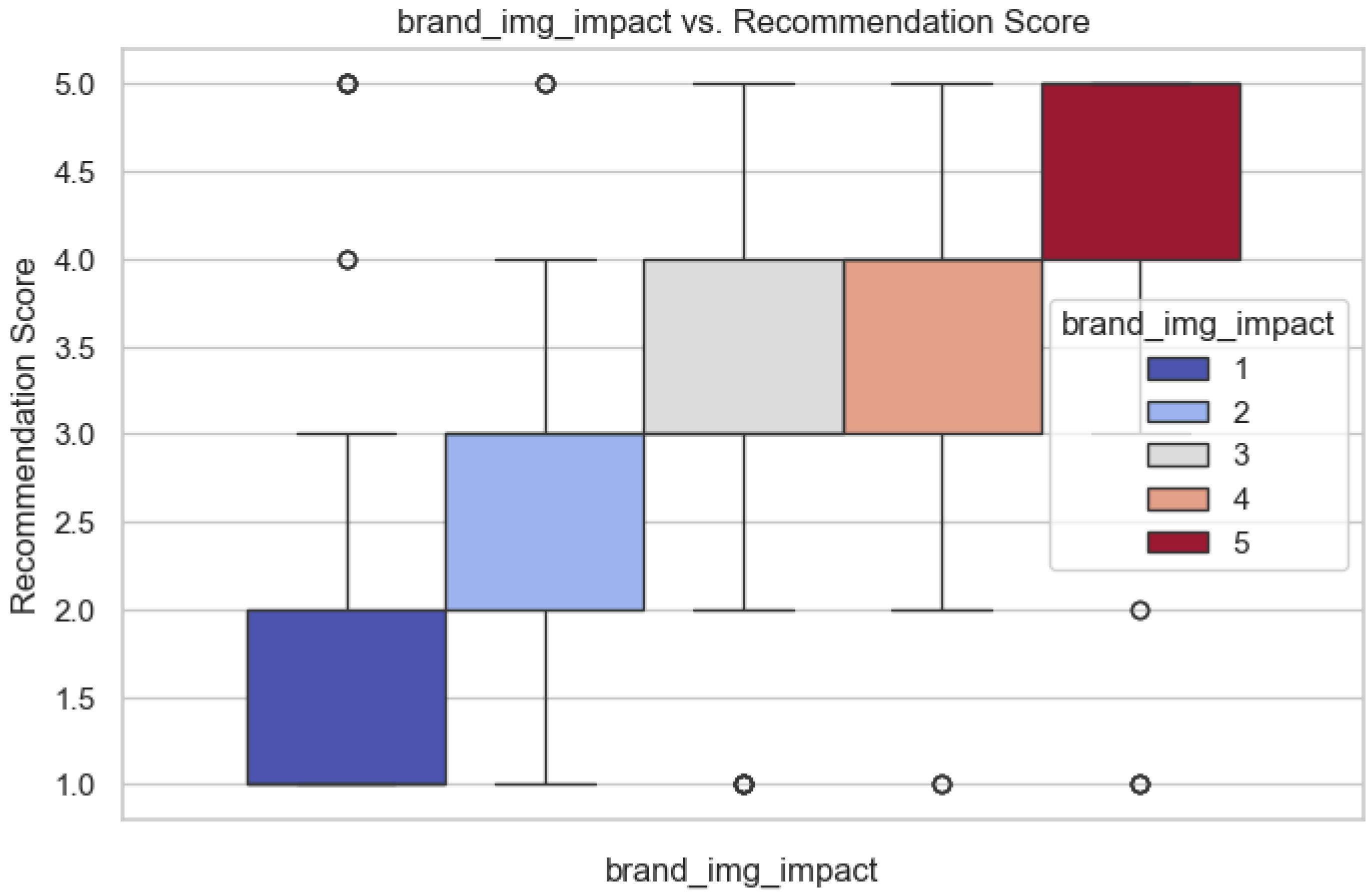
In [18]:

```python
# Counting the number of people using each trusted platform
df_trusted_counts = data['trusted_platform'].value_counts()
print(df_trusted_counts)
```

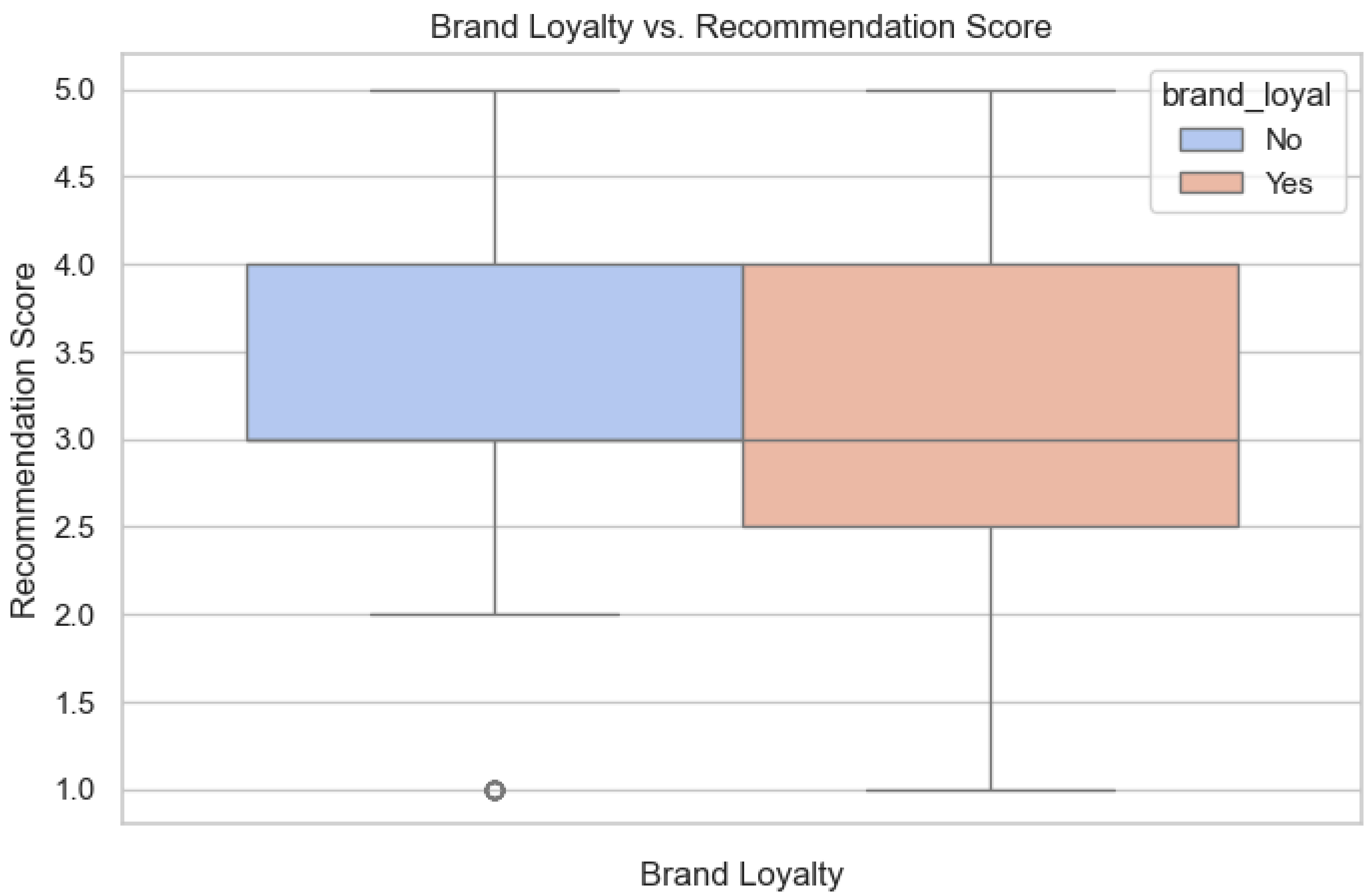
```
trusted_platform
Amazon      315
Flipkart    149
Myntra      105
Ajio         39
Name: count, dtype: int64
```

In [19]:

```python
# Analyze correlation between brand loyalty and recommendation score
plt.figure(figsize=(8, 5))
sns.boxplot(data=data, hue="brand_img_impact", y="recommendation", palette="coolwarm")
plt.title("brand_img_impact vs. Recommendation Score")
plt.xlabel("brand_img_impact")
plt.ylabel("Recommendation Score")
plt.show()
```



In [20]:
```
# Analyze correlation between brand loyalty and recommendation score
plt.figure(figsize=(8, 5))
sns.boxplot(data=data, hue="brand_loyal", y="recommendation", palette="coolwarm")
plt.title("Brand Loyalty vs. Recommendation Score")
plt.xlabel("Brand Loyalty")
plt.ylabel("Recommendation Score")
plt.show()
```



In [21]:
```
from sklearn.preprocessing import LabelEncoder

# Drop unnecessary columns
df = data.drop(columns=['age_group', 'gender', 'occupation', 'location'])

# Encode categorical satisfaction levels
satisfaction_mapping = {
    '• Very satisfied': 5,
    '• Satisfied': 4,
    '• Neutral': 3,
    '• Dissatisfied': 2,
    '• Very dissatisfied': 1
```

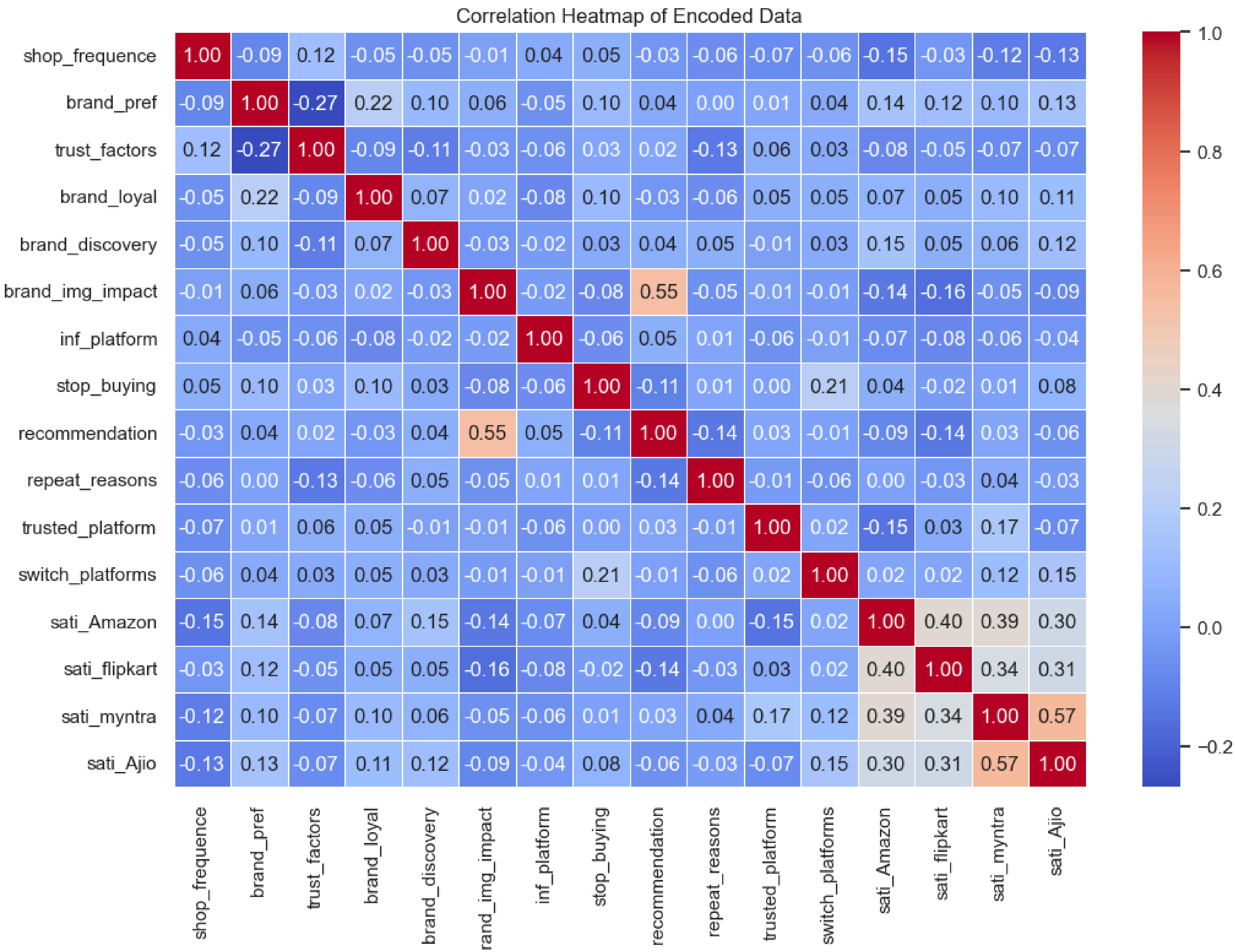
```python
}

# Apply mapping to satisfaction columns
satisfaction_columns = ['sati_Amazon', 'sati_flipkart', 'sati_myntra', 'sati_Ajio']
for col in satisfaction_columns:
    df[col] = df[col].map(satisfaction_mapping)

# Convert categorical variables using Label Encoding
categorical_cols = df.select_dtypes(include=['object']).columns
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le  # Store encoder for inverse transformation if needed

# Display updated dataset info
df.head()
```

Out[21]:

	shop_frequence	brand_pref	trust_factors	brand_loyal	brand_discovery	brand_img_impact	inf_platform	stop_buying	recommendatior
0	1	1	38	0	23	4	0	0	4
1	1	2	19	1	21	4	0	3	4
2	4	2	7	1	0	5	3	4	5
3	2	1	7	0	17	3	0	0	3
4	1	2	7	1	0	5	3	0	4

In [22]:

```python
correlation_matrix_encoded = df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix_encoded, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap of Encoded Data")
plt.show()
df.describe()
```

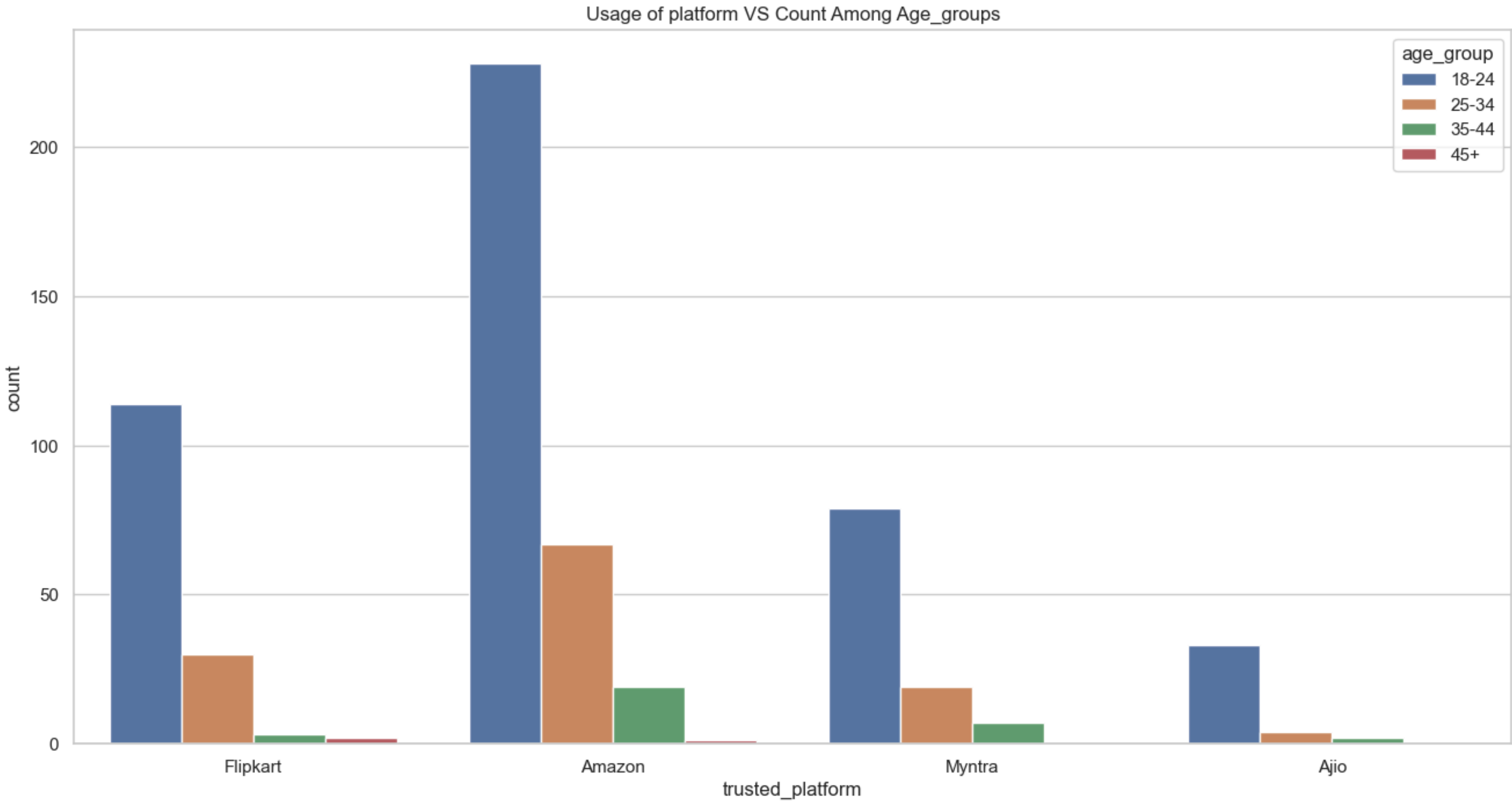

Correlation Heatmap of Encoded Data
shop_frequence 1.00 -0.09 0.12 -0.05 -0.05 -0.01 0.04 0.05 -0.03 -0.06 -0.07 -0.06 -0.15 -0.03 -0.12 -0.13
brand_pref -0.09 1.00 -0.27 0.22 0.10 0.06 -0.05 0.10 0.04 0.00 0.01 0.04 0.14 0.12 0.10 0.13
trust_factors 0.12 -0.27 1.00 -0.09 -0.11 -0.03 -0.06 0.03 0.02 -0.13 0.06 0.03 -0.08 -0.05 -0.07 -0.07
brand_loyal -0.05 0.22 -0.09 1.00 0.07 0.02 -0.08 0.10 -0.03 -0.06 0.05 0.05 0.07 0.05 0.10 0.11
brand_discovery -0.05 0.10 -0.11 0.07 1.00 -0.03 -0.02 0.03 0.04 0.05 -0.01 0.03 0.15 0.05 0.06 0.12
brand_img_impact -0.01 0.06 -0.03 0.02 -0.03 1.00 -0.02 -0.08 0.55 -0.05 -0.01 -0.01 -0.14 -0.16 -0.05 -0.09
inf_platform 0.04 -0.05 -0.06 -0.08 -0.02 -0.02 1.00 -0.06 0.05 0.01 -0.06 -0.01 -0.07 -0.08 -0.06 -0.04
stop_buying 0.05 0.10 0.03 0.10 0.03 -0.08 -0.06 1.00 -0.11 0.01 0.00 0.21 0.04 -0.02 0.01 0.08
recommendation -0.03 0.04 0.02 -0.03 0.04 0.55 0.05 -0.11 1.00 -0.14 0.03 -0.01 -0.09 -0.14 0.03 -0.06
repeat_reasons -0.06 0.00 -0.13 -0.06 0.05 -0.05 0.01 0.01 -0.14 1.00 -0.01 -0.06 0.00 -0.03 0.04 -0.03
trusted_platform -0.07 0.01 0.06 0.05 -0.01 -0.01 -0.06 0.00 0.03 -0.01 1.00 0.02 -0.15 0.03 0.17 -0.07
switch_platforms -0.06 0.04 0.03 0.05 0.03 -0.01 -0.01 0.21 -0.01 -0.06 0.02 1.00 0.02 0.02 0.12 0.15
sati_Amazon -0.15 0.14 -0.08 0.07 0.15 -0.14 -0.07 0.04 -0.09 0.00 -0.15 0.02 1.00 0.40 0.39 0.30
sati_flipkart -0.03 0.12 -0.05 0.05 0.05 -0.16 -0.08 -0.02 -0.14 -0.03 0.03 0.02 0.40 1.00 0.34 0.31
sati_myntra -0.12 0.10 -0.07 0.10 0.06 -0.05 -0.06 0.01 0.03 0.04 0.17 0.12 0.39 0.34 1.00 0.57
sati_Ajio -0.13 0.13 -0.07 0.11 0.12 -0.09 -0.04 0.08 -0.06 -0.03 -0.07 0.15 0.30 0.31 0.57 1.00
shop_frequence
brand_pref
trust_factors
brand_loyal
brand_discovery
rand_img_impact
inf_platform
stop_buying
recommendation
repeat_reasons
trusted_platform
switch_platforms
sati_Amazon
sati_flipkart
sati_myntra
sati_Ajio
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Out[22]:

	shop_frequence	brand_pref	trust_factors	brand_loyal	brand_discovery	brand_img_impact	inf_platform	stop_buying	recommend
count	608.000000	608.000000	608.000000	608.000000	608.000000	608.000000	608.000000	608.000000	608.00
mean	2.083882	1.371711	16.000000	0.610197	12.712171	3.182566	1.404605	1.277961	3.31
std	1.118013	0.671855	12.968851	0.488107	9.166646	1.172681	1.242087	1.432745	1.19
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.00
25%	1.000000	1.000000	7.000000	0.000000	2.000000	3.000000	0.000000	0.000000	3.00
50%	2.000000	1.000000	9.500000	1.000000	15.000000	3.000000	1.000000	1.000000	3.00
75%	3.000000	2.000000	28.000000	1.000000	19.000000	4.000000	3.000000	3.000000	4.00
max	4.000000	2.000000	48.000000	1.000000	30.000000	5.000000	3.000000	4.000000	5.00

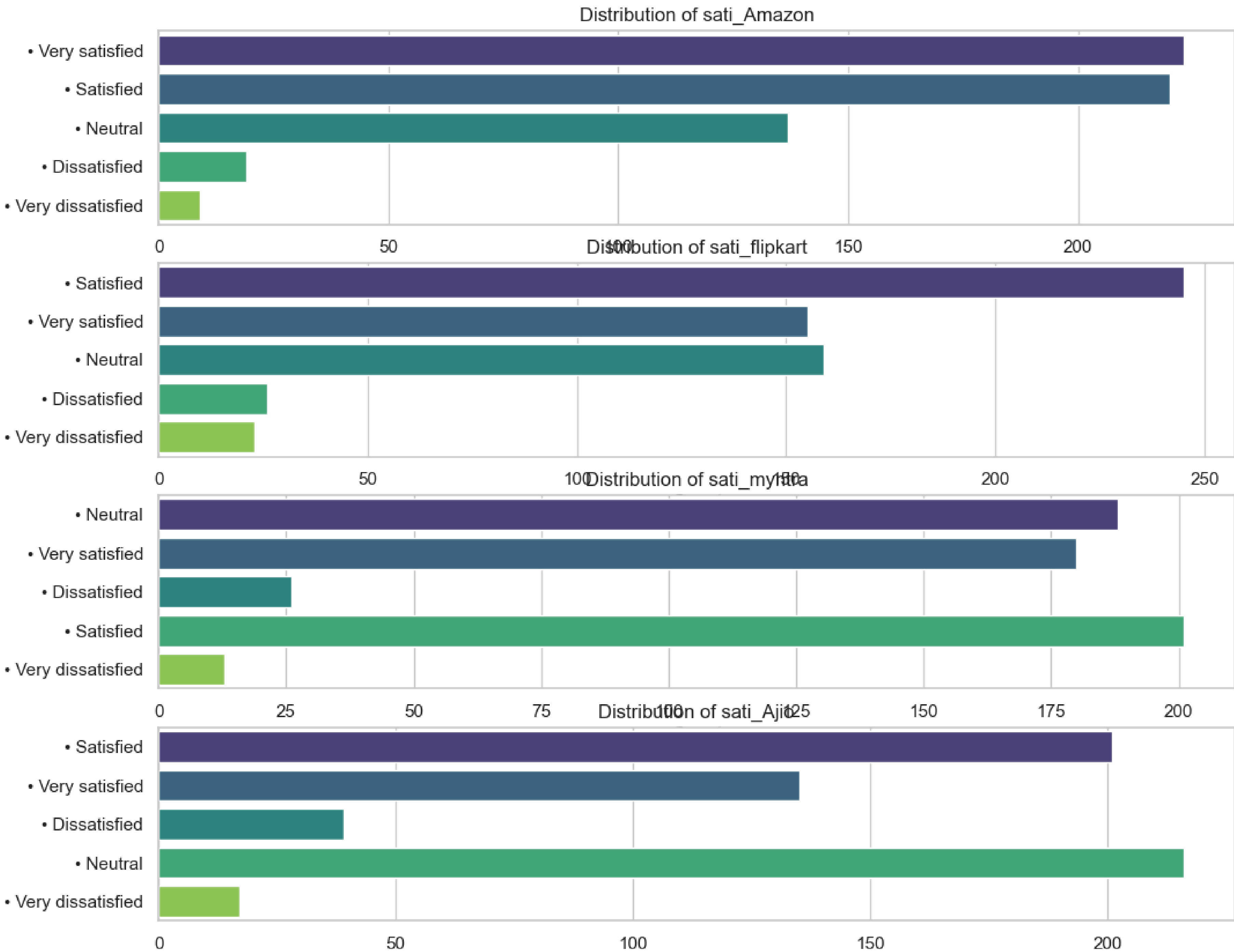
In [23]:

```
fig = plt.figure(figsize=(16, 8))
#polting how different age_group people using different platform
sns.countplot(data=data, x="trusted_platform", hue="age_group")
plt.title("Usage of platform VS Count Among Age_groups")
plt.show()
```



In [24]:
```
# ploting how people satisfaction with the platforms by bar chart visualization
categorical_columns = [
        'sati_Amazon', 'sati_flipkart', 'sati_myntra',
        'sati_Ajio']

fig, axes = plt.subplots(4,1, figsize=(12,10))
axes = axes.flatten()

for i, col in enumerate(categorical_columns):
    sns.countplot(y=data[col], ax=axes[i], hue=data[col], palette="viridis", legend=False)
    axes[i].set_title(f"Distribution of {col}")
```

```
    axes[i].set_xlabel("Count")
    axes[i].set_ylabel("")

plt.show()
```


Distribution of sati_Amazon
• Very satisfied
• Satisfied
• Neutral
• Dissatisfied
• Very dissatisfied
0 50 100 150 200
Distribution of sati_flipkart
• Satisfied
• Very satisfied
• Neutral
• Dissatisfied
• Very dissatisfied
0 50 100 150 200 250
Distribution of sati_myntra
• Neutral
• Very satisfied
• Dissatisfied
• Satisfied
• Very dissatisfied
0 25 50 75 100 125 150 175 200
Distribution of sati_Ajio
• Satisfied
• Very satisfied
• Dissatisfied
• Neutral
• Very dissatisfied
0 50 100 150 200

Count

```
In [25]: from scipy.stats import skew
         skew(df['trusted_platform'])
```

Out[25]: 0.4475631175172476

```
In [26]: df.head()
```

Out[26]:

	shop_frequence	brand_pref	trust_factors	brand_loyal	brand_discovery	brand_img_impact	inf_platform	stop_buying	recommendatior
0	1	1	38	0	23	4	0	0	4
1	1	2	19	1	21	4	0	3	4
2	4	2	7	1	0	5	3	4	5
3	2	1	7	0	17	3	0	0	3
4	1	2	7	1	0	5	3	0	4

Amazon

```
In [278…
# Convert satisfaction scores into categorical labels (1 = satisfied, 0 = not satisfied)
X=df.drop(columns=['sati_Amazon','sati_flipkart','sati_myntra','sati_Ajio'])
y_class = (df['sati_Amazon'] >= 4).astype(int)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y_class, test_size=0.2, random_state=0)

# Train logistic regression model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred_class = log_reg.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred_class)
report = classification_report(y_test, y_pred_class)

print("Accuracy :",accuracy)
print("Report : " ,report)
y_pred_class
```

```
Accuracy : 0.6639344262295082
Report :               precision    recall  f1-score   support

           0       0.11      0.03      0.05        34
           1       0.71      0.91      0.80        88

    accuracy                           0.66       122
   macro avg       0.41      0.47      0.42       122
weighted avg       0.54      0.66      0.59       122

C:\Users\marat\AppData\Roaming\Python\Python312\site-packages\sklearn\linear_model\_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[278…

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [279…

```
#Decision tree Accuracy
model=DecisionTreeClassifier()
model.fit(X_train,y_train)
predictions1=model.predict(X_test)
print("Decision tree Accuracy:",accuracy_score(y_test,model.predict(X_test)))
```

```
Decision tree Accuracy: 0.6147540983606558
```

In [280…
```
#Bagging Classifier Accuracy
BC=BaggingClassifier(n_estimators=100,random_state=0)
BC.fit(X_train,y_train)
BC.predict(X_test)
print("BaggingClassifier Accuracy:",accuracy_score(BC.predict(X_test),y_test))
```
```
BaggingClassifier Accuracy: 0.6065573770491803
```
In [281…
```
#parameter Bagging Classifier Accuracy
pBC=BaggingClassifier(n_estimators=100,random_state=0,bootstrap=False)
pBC.fit(X_train,y_train)
pBC.predict(X_test)
print("pBC Accuracy:",accuracy_score(pBC.predict(X_test),y_test))
```
```
pBC Accuracy: 0.6065573770491803
```
In [282…
```
#Random forest classifier Accuracy
RFC=RandomForestClassifier(n_estimators=100,max_features="sqrt",random_state=0)
RFC.fit(X_train,y_train)
RFC.predict(X_test)
print("RFC Accuracy:",accuracy_score(RFC.predict(X_test),y_test))
```
```
RFC Accuracy: 0.6475409836065574
```
In [283…
```
print(RFC.feature_importances_)
```
```
[0.08989467 0.05911068 0.13147889 0.03669039 0.14856011 0.07627122
 0.0683958  0.07316655 0.07962799 0.06162288 0.09506049 0.08012033]
```
In [284…
```
#Adaboost Accuracy
base_model=DecisionTreeClassifier(max_depth=1)
ABC=AdaBoostClassifier(base_model,n_estimators=500,random_state=0)
ABC.fit(X_train,y_train)
pred_ABC=ABC.predict(X_test)
print("AdaBoost Accuracy:",accuracy_score(y_test,pred_ABC))
```
```
AdaBoost Accuracy: 0.6639344262295082
```
In [285…
```
#cat boost Accuracy
cat_model = CatBoostClassifier(iterations=500, random_seed=0, verbose=0)
cat_model.fit(X_train, y_train)
```

```python
pred_cat = cat_model.predict(X_test)
print("CatBoost Accuracy:", accuracy_score(y_test, pred_cat))
```

```
CatBoost Accuracy: 0.6475409836065574
```

In [286…

```python
#Gradient boosting Accuracy
gb_model = GradientBoostingClassifier(n_estimators=500, random_state=42)
gb_model.fit(X_train, y_train)
pred_gb = gb_model.predict(X_test)
print("Gradient Boosting Accuracy:", accuracy_score(y_test, pred_gb))
```

```
Gradient Boosting Accuracy: 0.6557377049180327
```

In [287…

```python
#XGBoost Accuracy
xgb_model = XGBClassifier(n_estimators=500, random_state=0, eval_metric='logloss')
xgb_model.fit(X_train, y_train)
pred_xgb = xgb_model.predict(X_test)
print("XGBoost Accuracy:", accuracy_score(y_test, pred_xgb))
```

```
XGBoost Accuracy: 0.6721311475409836
```

In [289…

```python
param_grid = {
    'C': [0.1, 1, 10,100],
    'kernel': [ 'linear','rbf'],
    'gamma': [0.001, 0.01, 0.1,1]
}

grid_search = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Best Parameters: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
Best Score: 0.7325057858194824
```

In [293…

```python
param_dist = {
    'C': uniform(0.1, 100),
    'kernel': ['linear', 'rbf'],
    'gamma': uniform(0.001, 1)
}

# Perform Random Search
```

```
random_search = RandomizedSearchCV(SVC(), param_dist, n_iter=10, cv=5, scoring='accuracy', n_jobs=-1, random_state=42)
random_search.fit(X_train, y_train)

# Best parameters and best score
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)
```

```
Best Parameters: {'C': 37.55401188473625, 'gamma': 0.9517143064099162, 'kernel': 'linear'}
Best Score: 0.7304649694929519
```

Flipkart

In [294…

```
# Convert satisfaction scores into categorical labels (1 = satisfied, 0 = not satisfied)
X=df.drop(columns=['sati_Amazon','sati_flipkart','sati_myntra','sati_Ajio'])
y_class = (df['sati_flipkart'] >= 4).astype(int)

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

# Train logistic regression model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred_class = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_class)
report = classification_report(y_test, y_pred_class)

print("Accuracy:",accuracy)
print("Report :",report)
y_pred_class
```

```
Accuracy: 0.6147540983606558
Report :               precision    recall  f1-score   support

           0       0.43      0.07      0.11        46
           1       0.63      0.95      0.75        76

    accuracy                           0.61       122
   macro avg       0.53      0.51      0.43       122
weighted avg       0.55      0.61      0.51       122
```

Out[294…

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [295…

```
#Decision tree Accuracy
model=DecisionTreeClassifier()
model.fit(X_train,y_train)
predictions1=model.predict(X_test)
print("Decision tree Accuracy:",accuracy_score(y_test,model.predict(X_test)))
```

```
Decision tree Accuracy: 0.5491803278688525
```

In [296…

```
#Bagging Classifier Accuracy
BC=BaggingClassifier(n_estimators=100,random_state=0)
BC.fit(X_train,y_train)
#print(BC.predict(X_test))
print("Bagging Classifier Accuracy :",accuracy_score(BC.predict(X_test),y_test))
```

```
Bagging Classifier Accuracy : 0.6229508196721312
```

In [297…

```
#parameter Bagging classifier Accuracy
pBC=BaggingClassifier(n_estimators=100,random_state=0,bootstrap=False)
pBC.fit(X_train,y_train)
pBC.predict(X_test)
print("pBC Accuracy :",accuracy_score(pBC.predict(X_test),y_test))
```

```
pBC Accuracy : 0.5655737704918032
```

In [298…

```
#Randon forest classifier Accuracy
RFC=RandomForestClassifier(n_estimators=100,max_features="sqrt",random_state=0)
RFC.fit(X_train,y_train)
RFC.predict(X_test)
print("RFC Accuracy :",accuracy_score(RFC.predict(X_test),y_test))
```

```
RFC Accuracy : 0.6721311475409836
```

In [299…

```
print(RFC.feature_importances_)
```

```
[0.0868249  0.05179324 0.1439369  0.03816644 0.1476762  0.08749123
 0.06971191 0.07046556 0.08110149 0.06268083 0.07398889 0.0861624 ]
```

In [300…

```
#ADAboost Accuracy
base_model=DecisionTreeClassifier(max_depth=1)
ABC=AdaBoostClassifier(base_model,n_estimators=500,random_state=0)
ABC.fit(X_train,y_train)
pred_ABC=ABC.predict(X_test)
print("AdaBoost Accuracy:",accuracy_score(y_test,pred_ABC))
```

```
AdaBoost Accuracy: 0.6065573770491803
```

In [301…

```
#CATboost Accuracy
cat_model = CatBoostClassifier(iterations=500, random_seed=0, verbose=0)
cat_model.fit(X_train, y_train)
pred_cat = cat_model.predict(X_test)
print("CatBoost Accuracy:", accuracy_score(y_test, pred_cat))
```

```
CatBoost Accuracy: 0.6639344262295082
```

In [302…

```
#Gradient boosting Accuracy
gb_model = GradientBoostingClassifier(n_estimators=500, random_state=0)
gb_model.fit(X_train, y_train)
pred_gb = gb_model.predict(X_test)
print("Gradient Boosting Accuracy:", accuracy_score(y_test, pred_gb))
```

```
Gradient Boosting Accuracy: 0.6557377049180327
```

In [303…

```
#XGBoost Accuracy
xgb_model = XGBClassifier(n_estimators=500, random_state=0, eval_metric='logloss')
xgb_model.fit(X_train, y_train)
```

```
pred_xgb = xgb_model.predict(X_test)
print("XGBoost Accuracy:", accuracy_score(y_test, pred_xgb))
```

```
XGBoost Accuracy: 0.639344262295082
```

In [304…

```
param_grid = {
    'C': [0.1, 1, 10,100],
    'kernel': [ 'linear','rbf'],
    'gamma': [0.001, 0.01, 0.1,1]
}

grid_search = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Best Parameters: {'C': 0.1, 'gamma': 0.001, 'kernel': 'linear'}
Best Score: 0.6666736797811907
```

In [309…

```
param_dist = {
    'C': uniform(0.1, 100),
    'kernel': [ 'rbf'],
    'gamma': uniform(0.001, 1)
}

# Perform Random Search
random_search = RandomizedSearchCV(SVC(), param_dist, n_iter=10, cv=5, scoring='accuracy', n_jobs=-1, random_state=42)
random_search.fit(X_train, y_train)

# Best parameters and best score
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)
```

```
Best Parameters: {'C': 37.55401188473625, 'gamma': 0.9517143064099162, 'kernel': 'rbf'}
Best Score: 0.6646118241110877
```

Myntra

In [310…

```
# Convert satisfaction scores into categorical labels (1 = satisfied, 0 = not satisfied)
X=df.drop(columns=['sati_Amazon','sati_flipkart','sati_myntra','sati_Ajio'])
```

```
y_class = (df['sati_myntra'] >= 4).astype(int)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y_class, test_size=0.2, random_state=42)

# Train logistic regression model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred_class = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_class)
report = classification_report(y_test, y_pred_class)

print("Accuracy:",accuracy)
print("Report :",report)
y_pred_class
```

```
Accuracy: 0.5655737704918032
Report :               precision    recall  f1-score   support

           0       0.47      0.17      0.25        52
           1       0.58      0.86      0.69        70

    accuracy                           0.57       122
   macro avg       0.53      0.52      0.47       122
weighted avg       0.54      0.57      0.51       122
```

Out[310…

```
array([1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [311…

```
#Decision tree Accuracy
model=DecisionTreeClassifier()
model.fit(X_train,y_train)
predictions1=model.predict(X_test)
print("Accuracy of decision tree:",accuracy_score(y_test,model.predict(X_test)))
```

```
Accuracy of decision tree: 0.6721311475409836
```

In [312…

```
#Bagging Classifier Accuracy
BC=BaggingClassifier(n_estimators=100,random_state=0)
BC.fit(X_train,y_train)
BC.predict(X_test)
print("Bagging Classifier Accuracy :",accuracy_score(BC.predict(X_test),y_test))
```

```
Bagging Classifier Accuracy : 0.7049180327868853
```

In [313…

```
#parameter Bagging classifier Accuracy
pBC=BaggingClassifier(n_estimators=100,random_state=0,bootstrap=False)
pBC.fit(X_train,y_train)
pBC.predict(X_test)
print("pBC Accuracy :",accuracy_score(pBC.predict(X_test),y_test))
```

```
pBC Accuracy : 0.6639344262295082
```

In [314…

```
#Random forest classifier accuracy
RFC=RandomForestClassifier(n_estimators=100,max_features="sqrt",random_state=0)
RFC.fit(X_train,y_train)
#print(RFC.predict(X_test))
print("RFC Accuracy :",accuracy_score(RFC.predict(X_test),y_test))
```

```
RFC Accuracy : 0.7213114754098361
```

In [315…

```
print(RFC.feature_importances_)
```

```
[0.08629652 0.0495828  0.12655862 0.03632001 0.15080459 0.08246541
 0.06707184 0.07303466 0.084289   0.05909908 0.08798822 0.09648924]
```

In [316…

```
#ADAboost Accuracy
base_model=DecisionTreeClassifier(max_depth=1)
ABC=AdaBoostClassifier(base_model,n_estimators=500,random_state=0)
ABC.fit(X_train,y_train)
pred_ABC=ABC.predict(X_test)
print("AdaBoost Accuracy:",accuracy_score(y_test,pred_ABC))
```

```
AdaBoost Accuracy: 0.6885245901639344
```

In [317…

```
#CATboost Accuracy
cat_model = CatBoostClassifier(iterations=500, random_seed=0, verbose=0)
cat_model.fit(X_train, y_train)
```

```python
pred_cat = cat_model.predict(X_test)
print("CatBoost Accuracy:", accuracy_score(y_test, pred_cat))
```

```
CatBoost Accuracy: 0.6885245901639344
```

In [318…

```python
#Gradient boost Accuracy
gb_model = GradientBoostingClassifier(n_estimators=500, random_state=0)
gb_model.fit(X_train, y_train)
pred_gb = gb_model.predict(X_test)
print("Gradient Boosting Accuracy:", accuracy_score(y_test, pred_gb))
```

```
Gradient Boosting Accuracy: 0.6967213114754098
```

In [319…

```python
#XG boost Accuracy
xgb_model = XGBClassifier(n_estimators=500, random_state=0,  eval_metric='logloss')
xgb_model.fit(X_train, y_train)
pred_xgb = xgb_model.predict(X_test)
print("XGBoost Accuracy:", accuracy_score(y_test, pred_xgb))
```

```
XGBoost Accuracy: 0.6557377049180327
```

In [321…

```python
param_grid = {
    'C': [0.1, 1, 10,100],
    'kernel': [ 'linear','rbf'],
    'gamma': [0.001, 0.01, 0.1,1]
}

grid_search = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Best Parameters: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
Best Score: 0.6399957921312854
```

In [324…

```python
param_dist = {
    'C': uniform(0.1, 100),
    'kernel': [ 'linear','rbf'],
    'gamma': uniform(0.001, 1)
}

# Perform Random Search
```

```
random_search = RandomizedSearchCV(SVC(), param_dist, n_iter=10, cv=5, scoring='accuracy', n_jobs=-1, random_state=42)
random_search.fit(X_train, y_train)

# Best parameters and best score
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)
```

```
Best Parameters: {'C': 37.55401188473625, 'gamma': 0.9517143064099162, 'kernel': 'linear'}
Best Score: 0.6399116347569955
```

Ajio

In [325…

```
# Convert satisfaction scores into categorical labels (1 = satisfied, 0 = not satisfied)
X=df.drop(columns=['sati_Amazon','sati_flipkart','sati_myntra','sati_Ajio'])
y_class = (df['sati_Ajio'] >= 4).astype(int)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y_class, test_size=0.2, random_state=42)

# Train logistic regression model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred_class = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_class)
report = classification_report(y_test, y_pred_class)

print("Accuracy:",accuracy)
print("Report :",report)
y_pred_class
```

```
Accuracy: 0.6065573770491803
Report :               precision    recall  f1-score   support

           0       0.58      0.47      0.52        55
           1       0.62      0.72      0.67        67

    accuracy                           0.61       122
   macro avg       0.60      0.59      0.59       122
weighted avg       0.60      0.61      0.60       122
```

Out[325…

```
array([1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1])
```

In [326…

```
#Decision tree Accuracy
model=DecisionTreeClassifier()
model.fit(X_train,y_train)
predictions1=model.predict(X_test)
print("Decision tree Accuracy :",accuracy_score(y_test,model.predict(X_test)))
```

```
Decision tree Accuracy : 0.5901639344262295
```

In [327…

```
#Bagging Classifier Accuracy
BC=BaggingClassifier(n_estimators=100,random_state=0)
BC.fit(X_train,y_train)
BC.predict(X_test)
print("Bagging Classifier Accuracy :",accuracy_score(BC.predict(X_test),y_test))
```

```
Bagging Classifier Accuracy : 0.6147540983606558
```

In [328…

```
#parameter Bagging classifier Accuracy
pBC=BaggingClassifier(n_estimators=100,random_state=0,bootstrap=False)
pBC.fit(X_train,y_train)
pBC.predict(X_test)
print("pBC Accuracy :",accuracy_score(pBC.predict(X_test),y_test))
```

```
pBC Accuracy : 0.5737704918032787
```

In [329…
```
#Random forest classifier Accuracy
RFC=RandomForestClassifier(n_estimators=100,max_features="sqrt",random_state=0)
RFC.fit(X_train,y_train)
RFC.predict(X_test)
print("RFC Accuracy :",accuracy_score(RFC.predict(X_test),y_test))
```

```
RFC Accuracy : 0.6475409836065574
```

In [330…
```
print(RFC.feature_importances_)
```

```
[0.0803623  0.04698539 0.14496338 0.03384012 0.16937552 0.08256615
 0.06374133 0.07515822 0.0829073  0.05452587 0.08227994 0.08329448]
```

In [331…
```
#ADA boost Accuracy
base_model=DecisionTreeClassifier(max_depth=1)
ABC=AdaBoostClassifier(base_model,n_estimators=500,random_state=0)
ABC.fit(X_train,y_train)
pred_ABC=ABC.predict(X_test)
print("AdaBoost Accuracy:",accuracy_score(y_test,pred_ABC))
```

```
AdaBoost Accuracy: 0.5901639344262295
```

In [332…
```
#CAT boost Accurcay
cat_model = CatBoostClassifier(iterations=500, random_seed=0, verbose=0)
cat_model.fit(X_train, y_train)
pred_cat = cat_model.predict(X_test)
print("CatBoost Accuracy:", accuracy_score(y_test, pred_cat))
```

```
CatBoost Accuracy: 0.6311475409836066
```

In [333…
```
#Gradient boost Accuracy
gb_model = GradientBoostingClassifier(n_estimators=500, random_state=0)
gb_model.fit(X_train, y_train)
pred_gb = gb_model.predict(X_test)
print("Gradient Boosting Accuracy:", accuracy_score(y_test, pred_gb))
```

```
Gradient Boosting Accuracy: 0.5573770491803278
```

In [334…
```
#XG boost Accuracy
xgb_model = XGBClassifier(n_estimators=500, random_state=0, eval_metric='logloss')
xgb_model.fit(X_train, y_train)
```

```python
pred_xgb = xgb_model.predict(X_test)
print("XGBoost Accuracy:", accuracy_score(y_test, pred_xgb))
```

```
XGBoost Accuracy: 0.5409836065573771
```

In [335…

```python
param_grid = {
    'C': [0.1, 1, 10,100],
    'kernel': [ 'linear','rbf'],
    'gamma': [0.001, 0.01, 0.1,1]
}

grid_search = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Best Parameters: {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
Best Score: 0.584283610351357
```

In [336…

```python
param_dist = {
    'C': uniform(0.1, 100),
    'kernel': [ 'linear','rbf'],
    'gamma': uniform(0.001, 1)
}

# Perform Random Search
random_search = RandomizedSearchCV(SVC(), param_dist, n_iter=10, cv=5, scoring='accuracy', n_jobs=-1, random_state=42)
random_search.fit(X_train, y_train)

# Best parameters and best score
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)
```

```
Best Parameters: {'C': 15.699452033620265, 'gamma': 0.05908361216819946, 'kernel': 'rbf'}
Best Score: 0.5823690300862612
```

In [ ]: