E-Commerce Analytics: Swiggy, Zomato, Blinkit

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

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
## Load the Dataset
df = pd.read_csv(r"C:\Users\chitt\Downloads\Ecommerce_Delivery_Analytics_New.csv
df
```

Out[2]:

	Order ID	Customer ID	Platform	Order Date & Time	Delivery Time (Minutes)	Product Category	Order Value (INR)	Custom Feedbac
0	ORD000001	CUST2824	JioMart	19:29.5	30	Fruits & Vegetables	382	Fa deliver gre servic
1	ORD000002	CUST1409	Blinkit	54:29.5	16	Dairy	279	Quick ar reliabl
2	ORD000003	CUST5506	JioMart	21:29.5	25	Beverages	599	Iten missir fro orde
3	ORD000004	CUST5012	JioMart	19:29.5	42	Beverages	946	Iten missir fro orde
4	ORD000005	CUST4657	Blinkit	49:29.5	30	Beverages	334	Fa deliver gre servic
...	...	...	...	...	...	...	...	
99995	ORD099996	CUST5324	JioMart	49:29.5	24	Dairy	289	Packagir could b bette
99996	ORD099997	CUST1677	JioMart	18:29.5	19	Snacks	322	Goo quali product
99997	ORD099998	CUST8198	JioMart	27:29.5	41	Dairy	135	Fa deliver gre servic
99998	ORD099999	CUST9975	JioMart	14:29.5	31	Grocery	973	Quick ar reliabl
99999	ORD100000	CUST3748	JioMart	41:29.5	34	Fruits & Vegetables	453	Packagir could b bette

100000 rows × 11 columns

In [3]:

```
## Display basis info
df.head()
```

Out[3]:

	Order ID	Customer ID	Platform	Order Date & Time	Delivery Time (Minutes)	Product Category	Order Value (INR)	Customer Feedback	S I
0	ORD000001	CUST2824	JioMart	19:29.5	30	Fruits & Vegetables	382	Fast delivery, great service!	
1	ORD000002	CUST1409	Blinkit	54:29.5	16	Dairy	279	Quick and reliable!	
2	ORD000003	CUST5506	JioMart	21:29.5	25	Beverages	599	Items missing from order.	
3	ORD000004	CUST5012	JioMart	19:29.5	42	Beverages	946	Items missing from order.	
4	ORD000005	CUST4657	Blinkit	49:29.5	30	Beverages	334	Fast delivery, great service!	

In [4]:
```
df.tail()
```

Out[4]:

	Order ID	Customer ID	Platform	Order Date & Time	Delivery Time (Minutes)	Product Category	Order Value (INR)	Custome Feedbac
99995	ORD099996	CUST5324	JioMart	49:29.5	24	Dairy	289	Packagir could b bette
99996	ORD099997	CUST1677	JioMart	18:29.5	19	Snacks	322	Goc quali product
99997	ORD099998	CUST8198	JioMart	27:29.5	41	Dairy	135	Fa deliver gre servic
99998	ORD099999	CUST9975	JioMart	14:29.5	31	Grocery	973	Quick ar reliabl
99999	ORD100000	CUST3748	JioMart	41:29.5	34	Fruits & Vegetables	453	Packagir could b bette

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 11 columns):
 #   Column                   Non-Null Count   Dtype 
---  ------                   --------------   ----- 
 0   Order ID                 100000 non-null  object
 1   Customer ID              100000 non-null  object
 2   Platform                 100000 non-null  object
 3   Order Date & Time        100000 non-null  object
 4   Delivery Time (Minutes)  100000 non-null  int64 
 5   Product Category         100000 non-null  object
 6   Order Value (INR)        100000 non-null  int64 
 7   Customer Feedback        100000 non-null  object
 8   Service Rating           100000 non-null  int64 
 9   Delivery Delay           100000 non-null  object
 10  Refund Requested         100000 non-null  object
dtypes: int64(3), object(8)
memory usage: 8.4+ MB
```

In [6]:
```python
df.describe()
```

Out[6]:

	Delivery Time (Minutes)	Order Value (INR)	Service Rating
count	100000.000000	100000.000000	100000.000000
mean	29.536140	590.994400	3.240790
std	9.958933	417.409058	1.575962
min	5.000000	50.000000	1.000000
25%	23.000000	283.000000	2.000000
50%	30.000000	481.000000	3.000000
75%	36.000000	770.000000	5.000000
max	76.000000	2000.000000	5.000000

In [7]:
```python
## Check for missing values
df.isnull().sum()
```

Out[7]:
```
Order ID                   0
Customer ID                0
Platform                   0
Order Date & Time          0
Delivery Time (Minutes)    0
Product Category           0
Order Value (INR)          0
Customer Feedback          0
Service Rating             0
Delivery Delay             0
Refund Requested           0
dtype: int64
```

In [8]:
```python
## Check for duplicate rows
df.duplicated().sum()
```

Out[8]: 0

In [9]:
```
## Handle missing vallues
df.dropna(inplace=True)
```

In [10]:
```
df.dtypes
```

Out[10]:
```
Order ID                   object
Customer ID                object
Platform                   object
Order Date & Time          object
Delivery Time (Minutes)     int64
Product Category           object
Order Value (INR)           int64
Customer Feedback          object
Service Rating              int64
Delivery Delay             object
Refund Requested           object
dtype: object
```

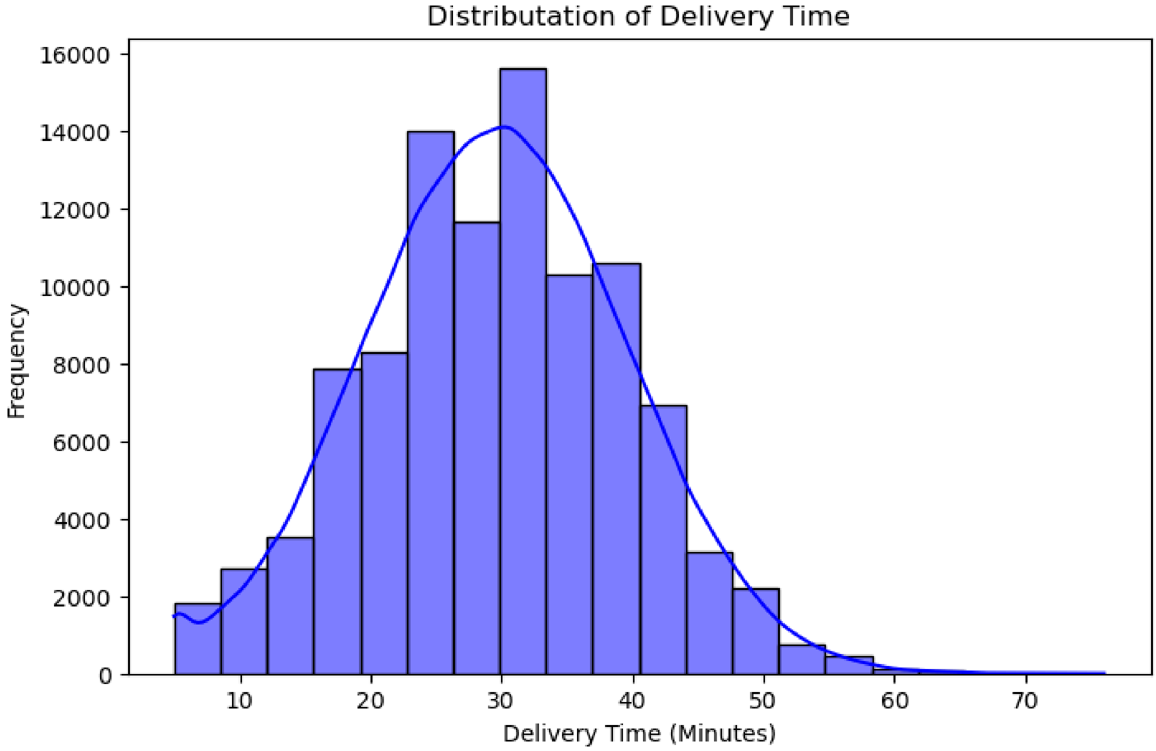
In [11]:
```
df.columns
```

Out[11]:
```
Index(['Order ID', 'Customer ID', 'Platform', 'Order Date & Time',
       'Delivery Time (Minutes)', 'Product Category', 'Order Value (INR)',
       'Customer Feedback', 'Service Rating', 'Delivery Delay',
       'Refund Requested'],
      dtype='object')
```

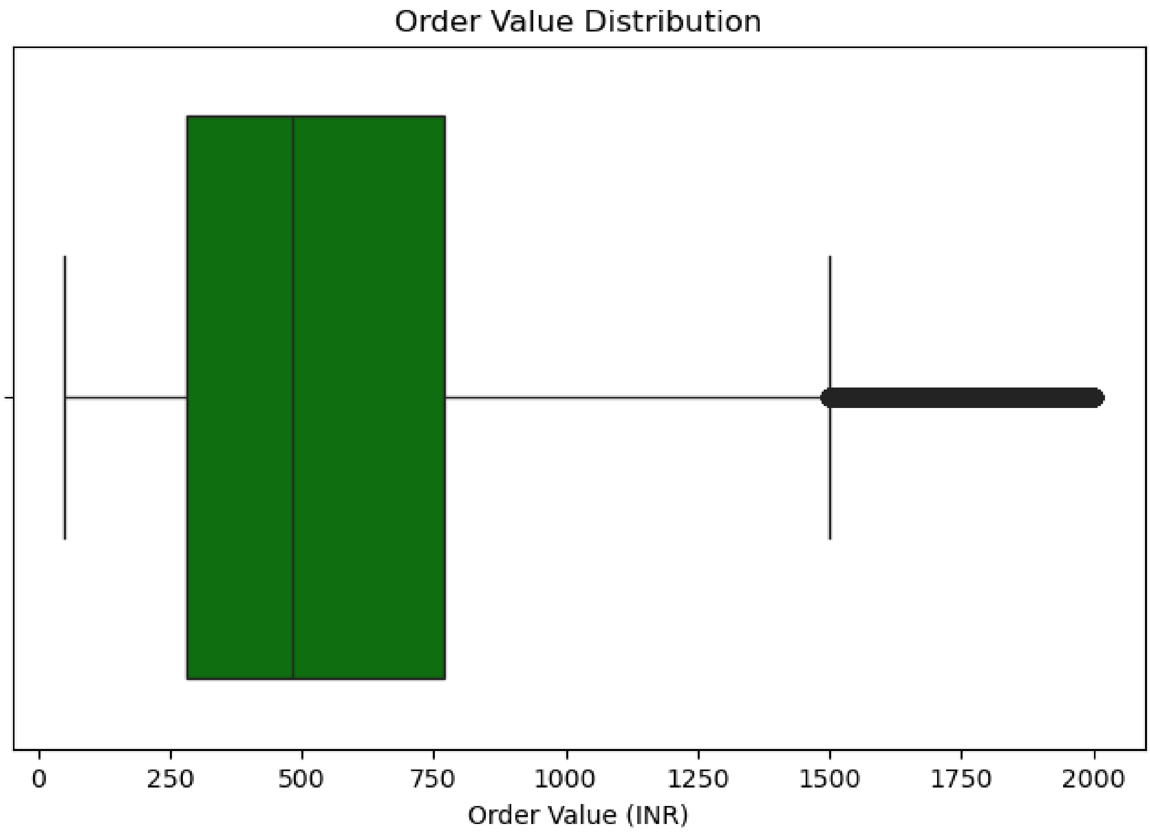
In [12]:
```
df.shape
```

Out[12]:
```
(100000, 11)
```

In [13]:
```
## EDA
## Histplot
plt.figure(figsize=(8,5))
sns.histplot(df['Delivery Time (Minutes)'], bins=20, kde=True, color='blue')
plt.title('Distributation of Delivery Time')
plt.xlabel('Delivery Time (Minutes)')
plt.ylabel('Frequency')
plt.show()
```



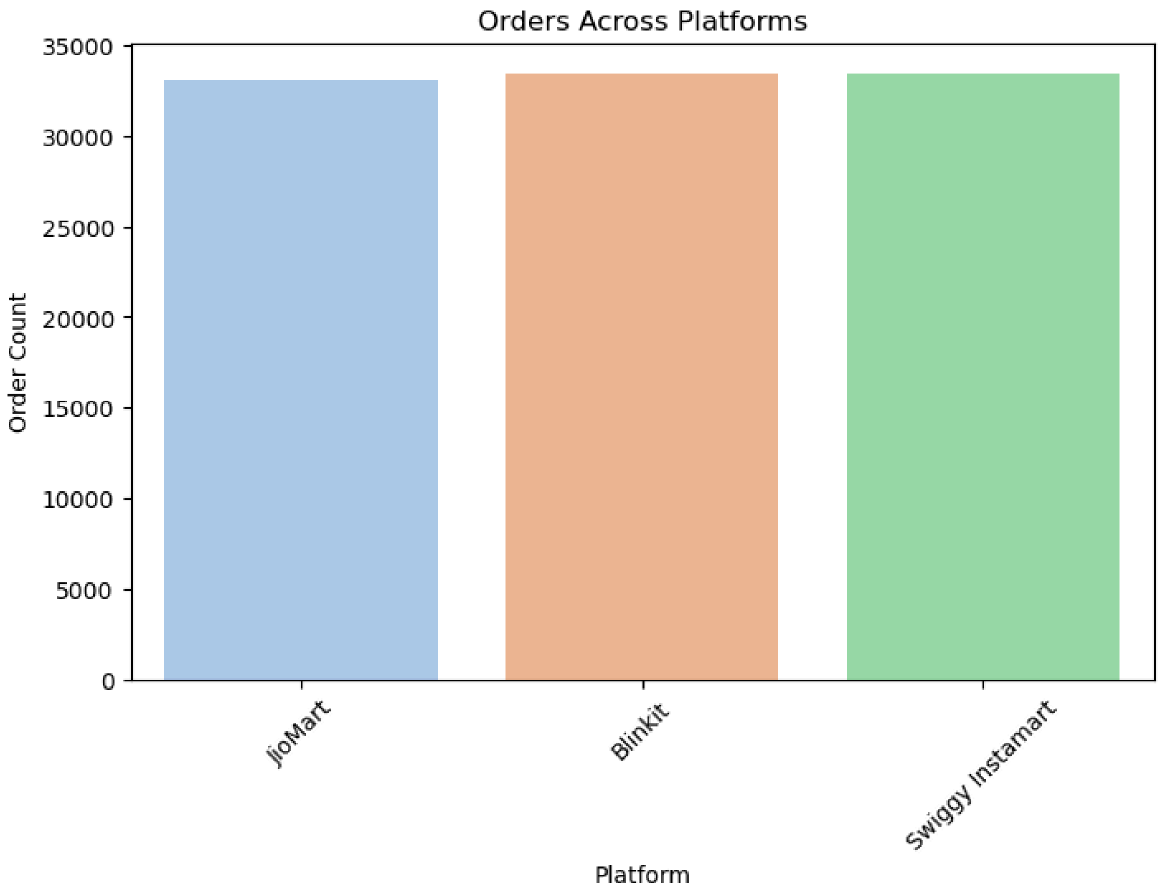
In [14]:
```
## Boxplot
plt.figure(figsize=(8,5))
sns.boxplot(x=df['Order Value (INR)'], color='green')
plt.title('Order Value Distribution')
plt.xlabel('Order Value (INR)')
plt.show()
```



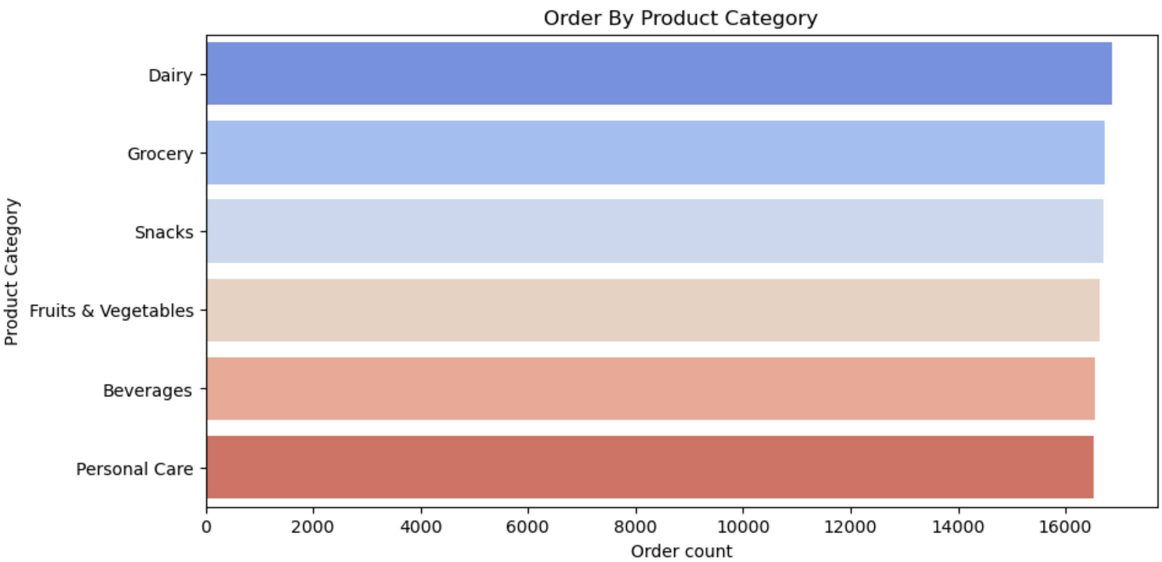
In [15]:
```
plt.figure(figsize=(8, 5))
sns.boxplot(x=df['Refund Requested'], y=df['Delivery Time (Minutes)'], palette='
plt.title('Impact of Delivery Time on Refund Requests')
```

```
plt.xlabel('Refund Requested')
plt.ylabel('Delivery Time (Minutes)')
plt.show()
```

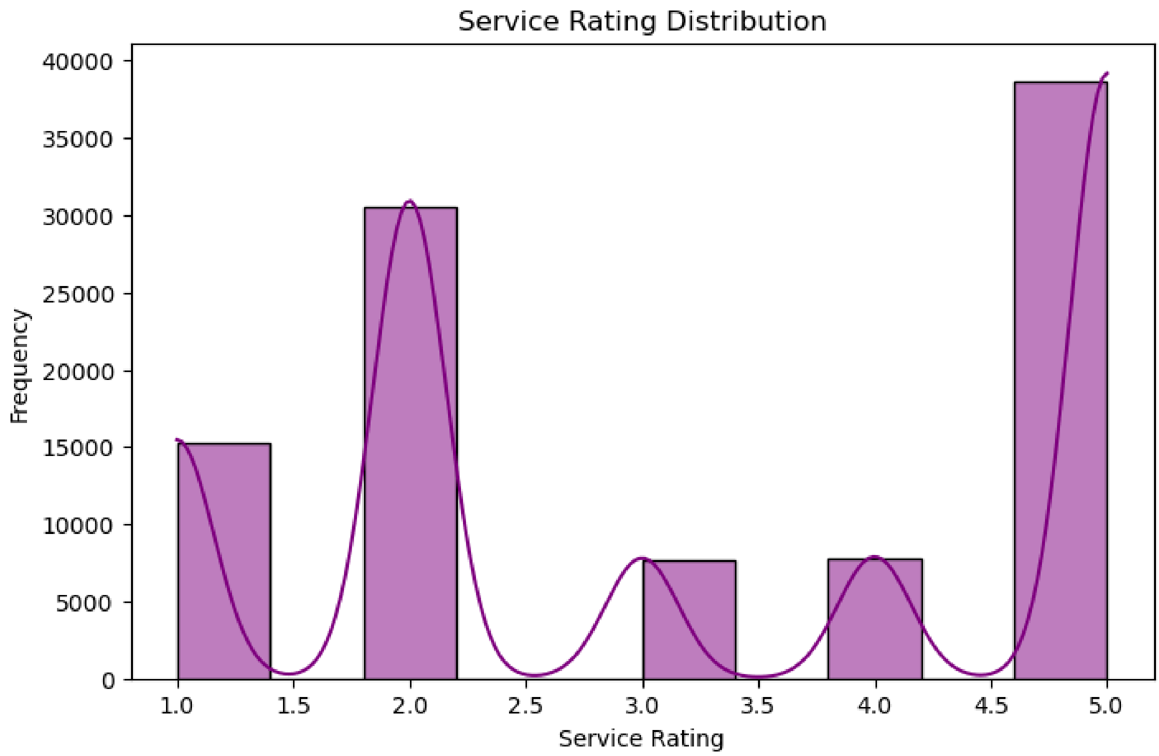
In [16]:
```
## Countplot
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='Platform', palette='pastel')
plt.title('Orders Across Platforms')
plt.xlabel('Platform')
plt.ylabel('Order Count')
plt.xticks(rotation=45)
plt.show()
```



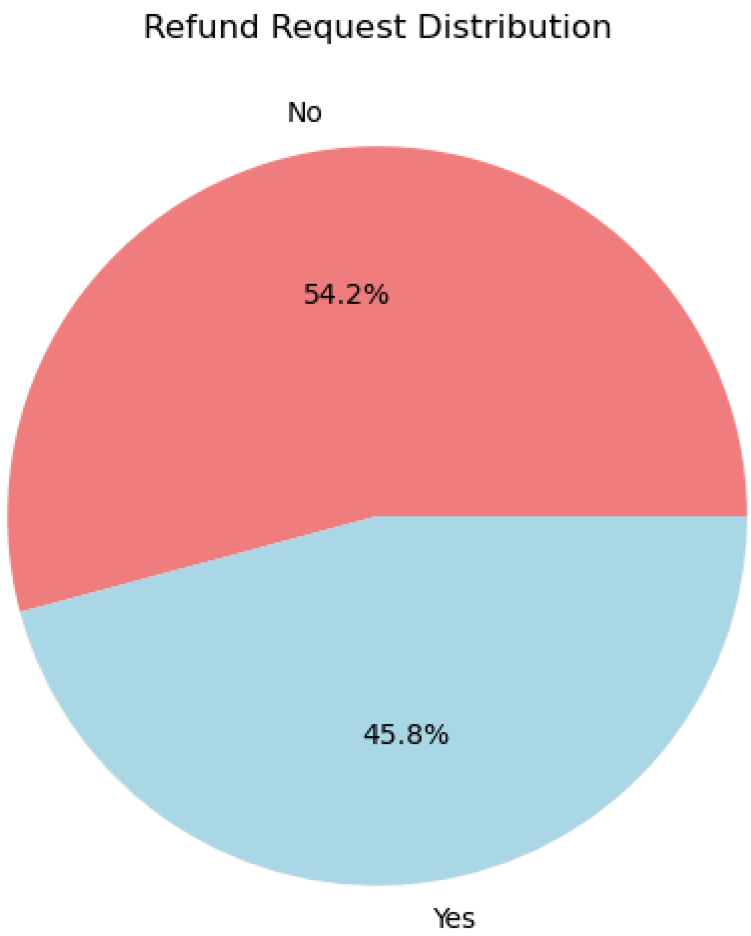
In [17]:
```
plt.figure(figsize=(10,5))
sns.countplot(data=df, y='Product Category', order=df['Product Category'].value_
plt.title('Order By Product Category')
plt.xlabel('Order count')
plt.ylabel('Product Category')
plt.show()
```



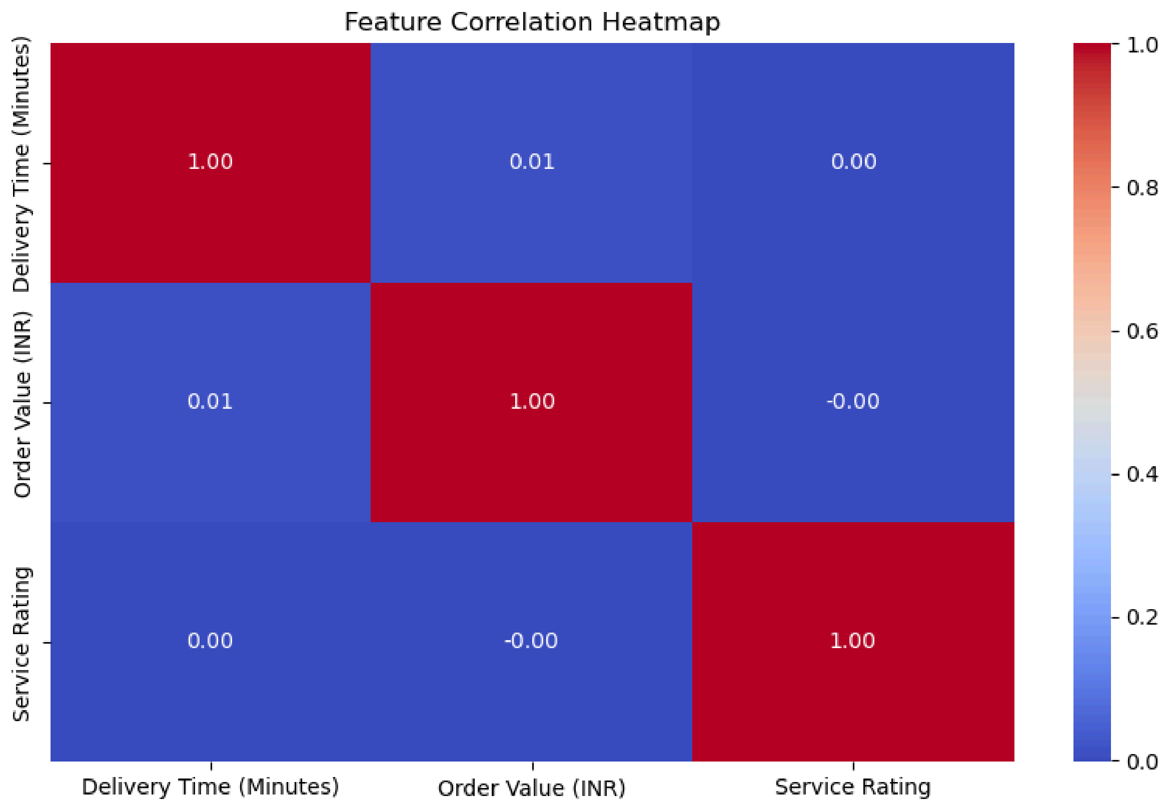
In [18]:
```
plt.figure(figsize=(8,5))
sns.histplot(df['Service Rating'], bins=10, kde=True, color='purple')
plt.title('Service Rating Distribution')
plt.xlabel('Service Rating')
plt.ylabel('Frequency')
plt.show()
```



In [19]:
```
## Pie Plot
plt.figure(figsize=(6, 6))
df['Refund Requested'].value_counts().plot(kind='pie', autopct='%1.1f%%', colors
plt.title('Refund Request Distribution')
plt.ylabel('')
plt.show()
```



In [20]:
```
plt.figure(figsize=(10, 6))
sns.heatmap(df.select_dtypes(include=['number']).corr(), annot=True, cmap='coolw
plt.title('Feature Correlation Heatmap')
plt.show()
```



In [21]:
```
## Predictive Modeling
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
```

In [22]:
```
df['Refund Requested']=df['Refund Requested'].map({'No':0,'Yes':1})
df['Delivery Delay']=df['Delivery Delay'].map({'No':0,'Yes':1})
```

In [23]:
```
df.drop(['Order ID', 'Customer ID', 'Order Date & Time'], axis=1, inplace=True)
```

In [24]:
```
# Encoding categorical features
categorical_cols = ['Platform', 'Product Category', 'Customer Feedback']
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

In [25]:
```
# Splitting Features and Target
X = df.drop(columns=['Refund Requested'])
y = df['Refund Requested']
```

In [26]:
```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [53]:
```
# Standardization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

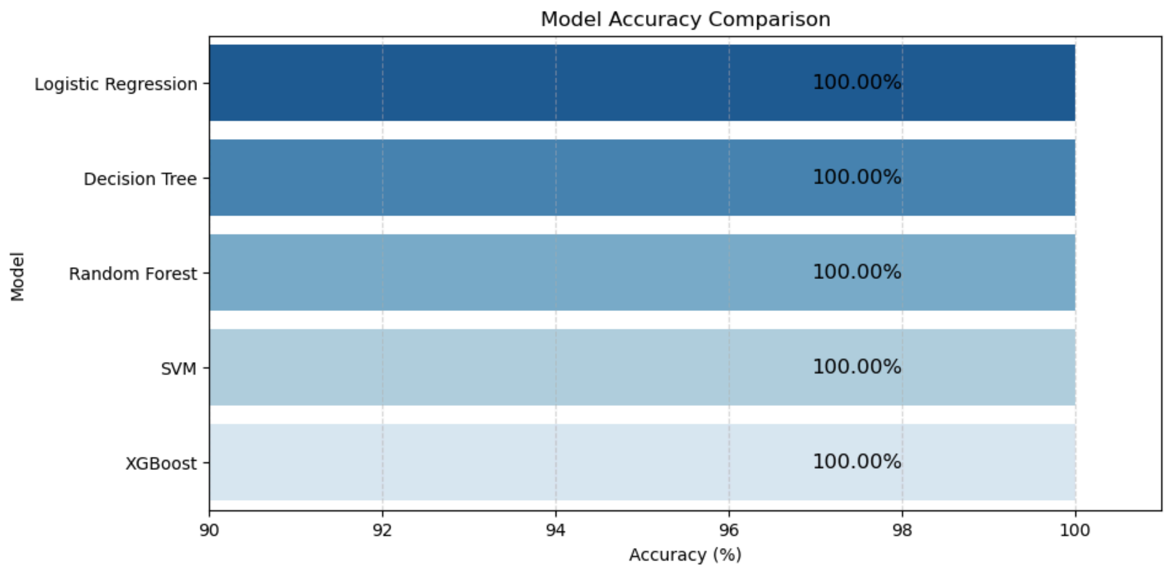
In [55]:
```
# Dictionary to store models and results
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}
```

In [57]:
```
# Training and Evaluating Models
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    print(f"◆ {name}: Accuracy = {acc:.4f}")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    print("-" * 50)
```

```
🔹 Logistic Regression: Accuracy = 1.0000
[[10836     0]
 [    0  9164]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10836
           1       1.00      1.00      1.00      9164

    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000

--------------------------------------------------
🔹 Decision Tree: Accuracy = 1.0000
[[10836     0]
 [    0  9164]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10836
           1       1.00      1.00      1.00      9164

    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000

--------------------------------------------------
🔹 Random Forest: Accuracy = 1.0000
[[10836     0]
 [    0  9164]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10836
           1       1.00      1.00      1.00      9164

    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000

--------------------------------------------------
🔹 SVM: Accuracy = 1.0000
[[10836     0]
 [    0  9164]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10836
           1       1.00      1.00      1.00      9164

    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000

--------------------------------------------------
🔹 XGBoost: Accuracy = 1.0000
[[10836     0]
 [    0  9164]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10836
           1       1.00      1.00      1.00      9164
```

```
    accuracy                           1.00     20000
   macro avg       1.00      1.00      1.00     20000
weighted avg       1.00      1.00      1.00     20000

--------------------------------------------------
```

In [59]:
```
results = {
    "Logistic Regression": 1.0,
    "Decision Tree": 1.0,
    "Random Forest": 1.0,
    "SVM": 1.0,
    "XGBoost": 1.0
}

# Convert to percentage
model_names = list(results.keys())
accuracies = [acc * 100 for acc in results.values()]

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(x=accuracies, y=model_names, palette="Blues_r")
# Add value labels
for index, value in enumerate(accuracies):
    plt.text(value - 2, index, f"{value:.2f}%", va='center', ha='right', fontsiz

# Titles and labels
plt.xlabel("Accuracy (%)")
plt.ylabel("Model")
plt.title("Model Accuracy Comparison")
plt.xlim(90, 101)  # Focus on high accuracy range
plt.grid(axis='x', linestyle="--", alpha=0.5)
plt.show()
```



Completed