In [1]:
```python
import pandas as pd
```

In [2]:
```python
data =pd.read_csv('C:/Users/HP/Desktop/Groceries_dataset.csv')
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_21180\1070506844.py:1: DtypeWarning: Column
s (1,2,3,4,5,6,7,9,10) have mixed types. Specify dtype option on import or set low_m
emory=False.
  data =pd.read_csv('C:/Users/HP/Desktop/Groceries_dataset.csv')
```

In [3]:
```python
data = pd.read_csv('C:/Users/HP/Desktop/Groceries_dataset.csv', low_memory=False)
```

In [4]:
```python
data.head(5)
```

Out[4]:

	Customer_ID_number	OrderDate	Items	ItemCategory	Gender	Geographic location	Purchas Histor
0	1808.0	2025-03-04	Indomie Instant Noodles	Food	F	Ikeja	2023-0 3
1	2552.0	2025-01-01	Golden Morn	Food	M	Iyana ipaja	2022-0 (
2	2300.0	2025-03-02	Spaghetti	Food	M	Mafoluku	2024-0 1
3	1187.0	2025-03-30	Maggi Cubes	Food Seasoning	M	Ikeja	2025-0 1
4	3037.0	2025-02-20	Golden Morn	Food	F	Festac town	2023-0 1

In [5]:
```python
data.head(7)
```

Out[5]:

	Customer_ID_number	OrderDate	Items	ItemCategory	Gender	Geographic location	Purchas Histor
0	1808.0	2025-03-04	Indomie Instant Noodles	Food	F	Ikeja	2023-04 3
1	2552.0	2025-01-01	Golden Morn	Food	M	Iyana ipaja	2022-05 0
2	2300.0	2025-03-02	Spaghetti	Food	M	Mafoluku	2024-04 1
3	1187.0	2025-03-30	Maggi Cubes	Food Seasoning	M	Ikeja	2025-02 1
4	3037.0	2025-02-20	Golden Morn	Food	F	Festac town	2023-06 1
5	4941.0	2025-02-20	Semovita	Food	M	Agege	2022-03 1
6	4501.0	2025-05-13	Maggi Cubes	Food Seasoning	F	Iyana ipaja	2023-10 1

In [6]:
```
data.dtypes
```

Out[6]:
```
Customer_ID_number     float64
OrderDate               object
Items                   object
ItemCategory            object
Gender                  object
Geographic location     object
Purchase History        object
ProductPrice            object
OrderQuantity          float64
OrderNumber             object
PaymentMethod           object
Age                    float64
dtype: object
```

In [7]:
```
def clean_price_column(df, column_name):
    df[column_name] = (
        df[column_name]
        .astype(str)  # Make sure it's all text first
        .str.replace(r'[^\d.]', '', regex=True)  # Remove anything that isn't a dig
    )
    df[column_name] = pd.to_numeric(df[column_name], errors='coerce')  # Turn into
    return df
```

In [8]:
```
import pandas as pd
data = pd.read_csv('C:/Users/HP/Desktop/Groceries_dataset.csv', encoding='utf-8', l
```

In [9]:
```
data = clean_price_column(data, 'ProductPrice')
```

In [10]:
```
data.head(7)
```

Out[10]:

	Customer_ID_number	OrderDate	Items	ItemCategory	Gender	Geographic location	Purchas Histor
0	1808.0	2025-03-04	Indomie Instant Noodles	Food	F	Ikeja	2023-0 3
1	2552.0	2025-01-01	Golden Morn	Food	M	Iyana ipaja	2022-0 (
2	2300.0	2025-03-02	Spaghetti	Food	M	Mafoluku	2024-0 1
3	1187.0	2025-03-30	Maggi Cubes	Food Seasoning	M	Ikeja	2025-0 1
4	3037.0	2025-02-20	Golden Morn	Food	F	Festac town	2023-0 1
5	4941.0	2025-02-20	Semovita	Food	M	Agege	2022-0 1
6	4501.0	2025-05-13	Maggi Cubes	Food Seasoning	F	Iyana ipaja	2023-1 1

In [11]:
```
# Check the number of missing values per column
data.isnull().sum()
```

Out[11]:
```
Customer_ID_number     1009810
OrderDate              1009810
Items                  1009810
ItemCategory           1009810
Gender                 1009810
Geographic location    1009810
Purchase History       1009810
ProductPrice           1009810
OrderQuantity          1009810
OrderNumber            1009810
PaymentMethod          1009810
Age                    1009810
dtype: int64
```

In [12]:
```
# Drop rows that contain any missing values
data_cleaned = data.dropna()

# Check shape to confirm
print("New shape after dropping missing values:", data_cleaned.shape)
```

```
New shape after dropping missing values: (38765, 12)
```

In [13]:
```
# Explicitly use .loc to modify the DataFrame
data_cleaned.loc[:, 'OrderDate'] = pd.to_datetime(data_cleaned['OrderDate'], errors
data_cleaned.loc[:, 'Purchase History'] = pd.to_datetime(data_cleaned['Purchase His
```

In [14]:
```
data_cleaned.loc[:, 'Age'] = data_cleaned['Age'].astype(int)
data_cleaned.loc[:, 'OrderQuantity'] = data_cleaned['OrderQuantity'].astype(int)
```

In [15]:
```
# Display summary info
data_cleaned.info()

# Preview cleaned data
data_cleaned.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 38765 entries, 0 to 38764
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Customer_ID_number  38765 non-null  float64
 1   OrderDate           38765 non-null  object 
 2   Items               38765 non-null  object 
 3   ItemCategory        38765 non-null  object 
 4   Gender              38765 non-null  object 
 5   Geographic location 38765 non-null  object 
 6   Purchase History    38765 non-null  object 
 7   ProductPrice        38765 non-null  float64
 8   OrderQuantity       38765 non-null  float64
 9   OrderNumber         38765 non-null  object 
 10  PaymentMethod       38765 non-null  object 
 11  Age                 38765 non-null  float64
dtypes: float64(4), object(8)
memory usage: 3.8+ MB
```

Out[15]:

	Customer_ID_number	OrderDate	Items	ItemCategory	Gender	Geographic location	Purchas Histor
0	1808.0	2025-03-04 00:00:00	Indomie Instant Noodles	Food	F	Ikeja	2023-0 3 00:00:0
1	2552.0	2025-01-01 00:00:00	Golden Morn	Food	M	Iyana ipaja	2022-0 0 00:00:0
2	2300.0	2025-03-02 00:00:00	Spaghetti	Food	M	Mafoluku	2024-0 1 00:00:0
3	1187.0	2025-03-30 00:00:00	Maggi Cubes	Food Seasoning	M	Ikeja	2025-0 1 00:00:0
4	3037.0	2025-02-20 00:00:00	Golden Morn	Food	F	Festac town	2023-0 1 00:00:0

In [16]:

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set Seaborn theme
sns.set(style="whitegrid")
```

In [17]:

```python
# Summary statistics
data_cleaned.describe()
```

Out[17]:

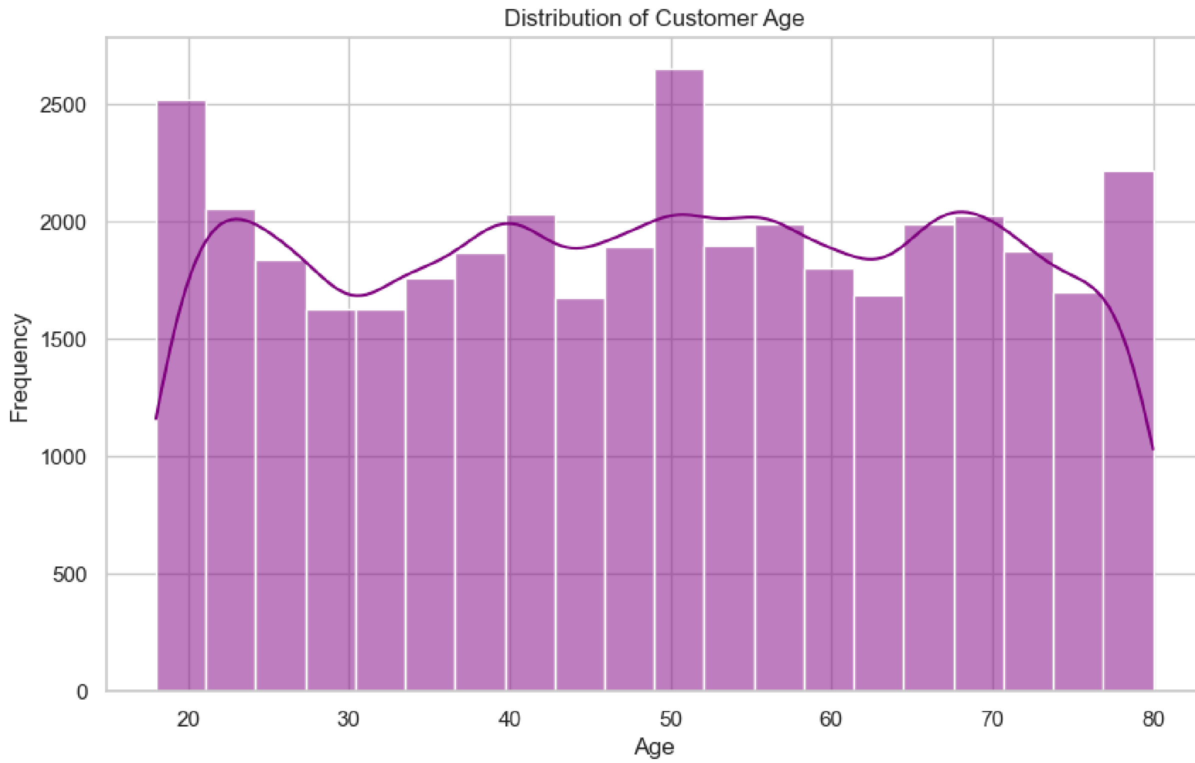
	Customer_ID_number	ProductPrice	OrderQuantity	Age
count	38765.000000	38765.000000	38765.000000	38765.000000
mean	3003.641868	2233.051025	5.504347	48.891268
std	1153.611031	3134.565943	2.871390	18.058219
min	1000.000000	120.000000	1.000000	18.000000
25%	2002.000000	700.000000	3.000000	34.000000
50%	3005.000000	1200.000000	5.000000	49.000000
75%	4007.000000	1855.000000	8.000000	65.000000
max	5000.000000	25000.000000	10.000000	80.000000

In [18]:

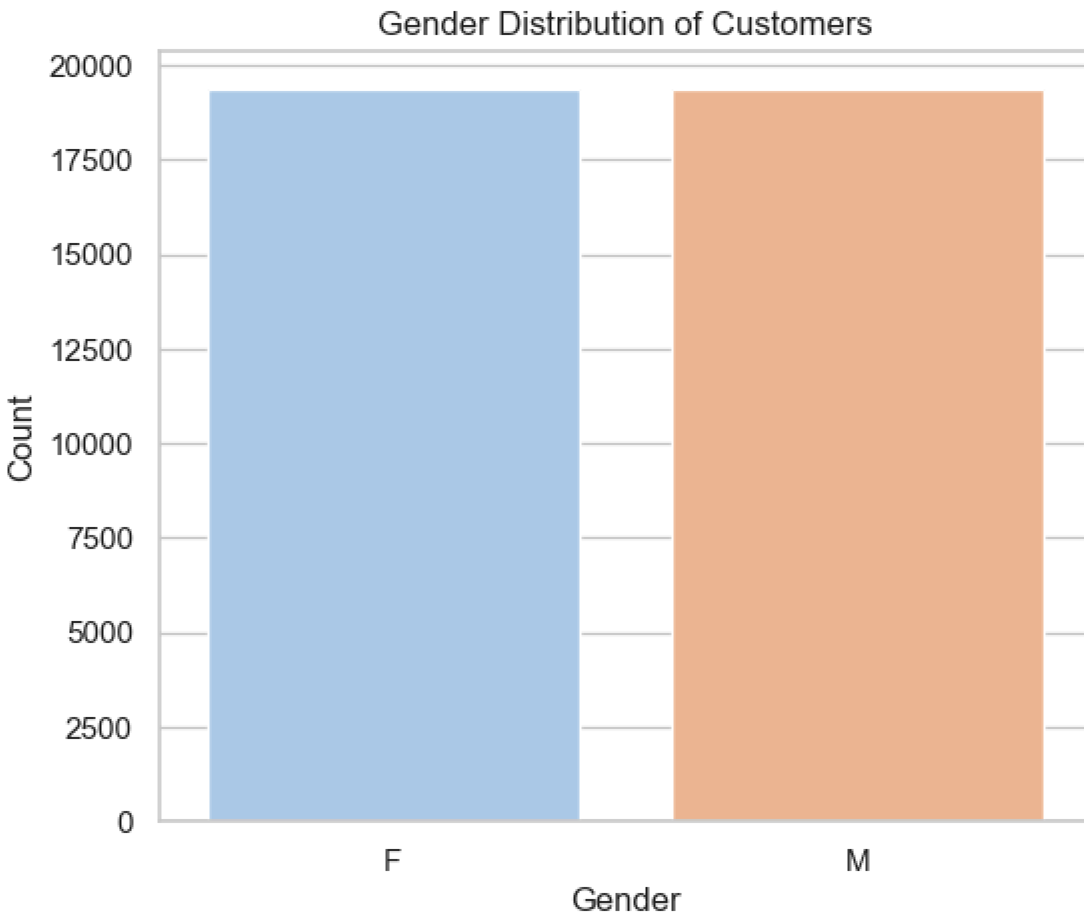
```python
plt.figure(figsize=(10, 6))
sns.histplot(data_cleaned['Age'], bins=20, kde=True, color='Purple')
plt.title("Distribution of Customer Age")
plt.xlabel("Age")
```

```
plt.ylabel("Frequency")
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

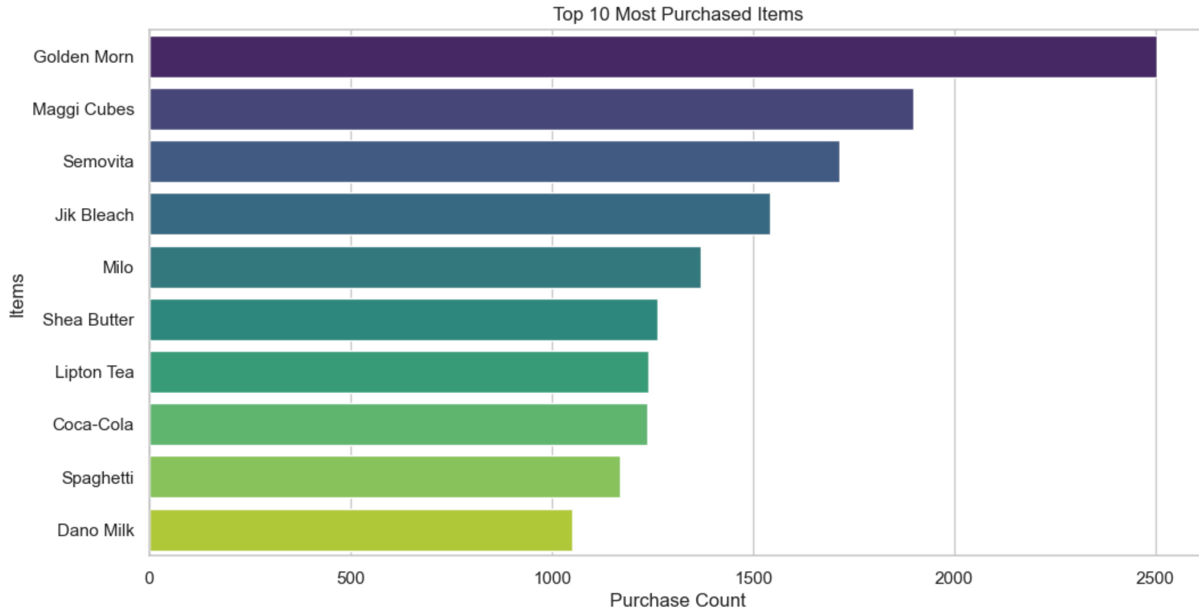


In [19]:
```
plt.figure(figsize=(6, 5))
sns.countplot(data=data_cleaned, x='Gender', palette='pastel')
plt.title("Gender Distribution of Customers")
plt.xlabel("Gender")
plt.ylabel("Count")
plt.show()
```



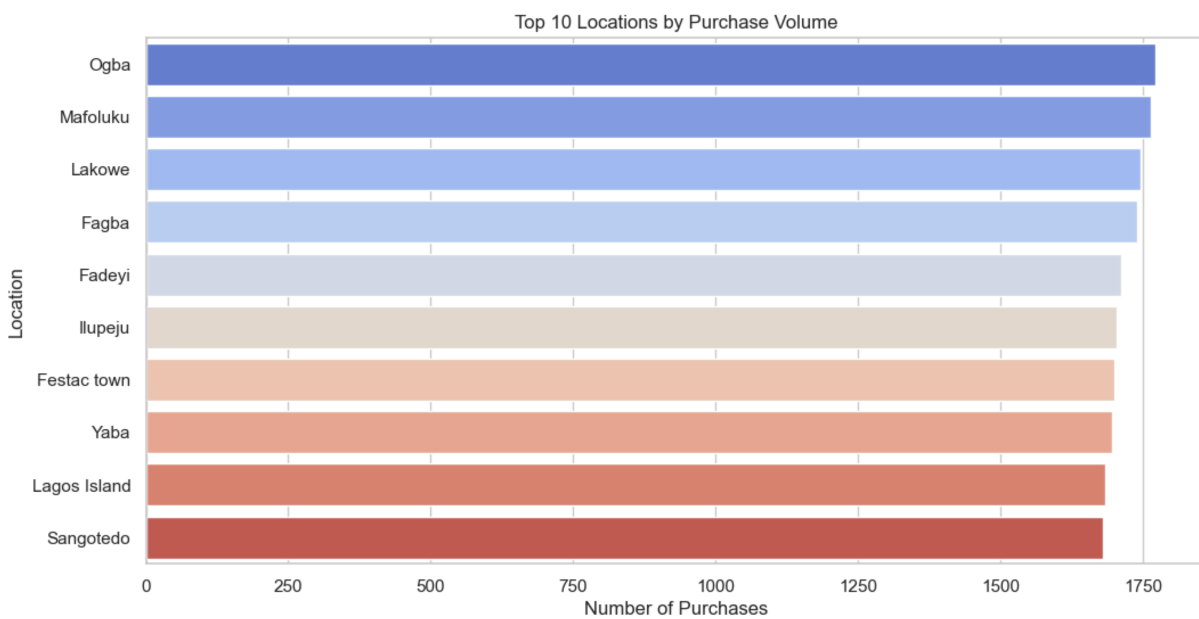
In [20]:
```
top_items = data_cleaned['Items'].value_counts().head(10)

plt.figure(figsize=(12, 6))
sns.barplot(x=top_items.values, y=top_items.index, palette='viridis')
plt.title("Top 10 Most Purchased Items")
plt.xlabel("Purchase Count")
plt.ylabel("Items")
plt.show()
```



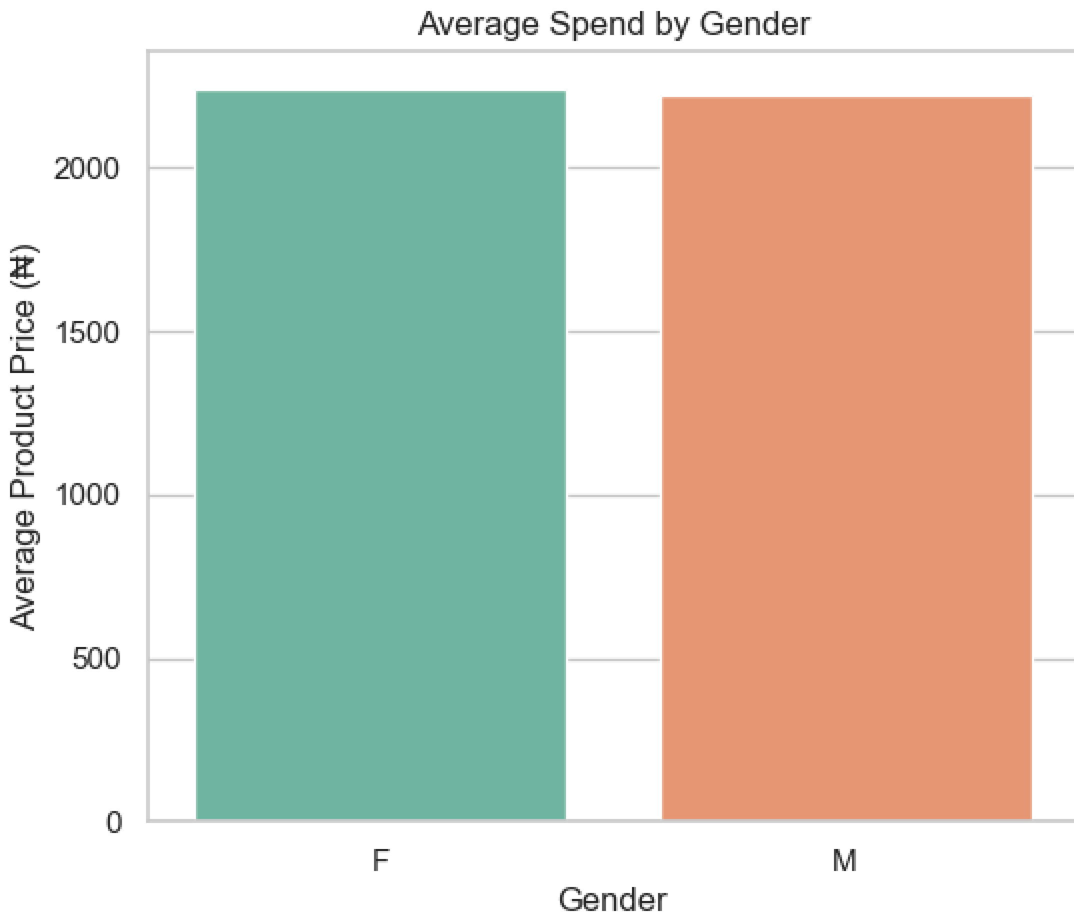
In [21]:
```
top_locations = data_cleaned['Geographic location'].value_counts().head(10)

plt.figure(figsize=(12, 6))
sns.barplot(x=top_locations.values, y=top_locations.index, palette='coolwarm')
plt.title("Top 10 Locations by Purchase Volume")
plt.xlabel("Number of Purchases")
plt.ylabel("Location")
plt.show()
```



In [22]:
```
avg_price_gender = data_cleaned.groupby('Gender')['ProductPrice'].mean().reset_inde

plt.figure(figsize=(6, 5))
sns.barplot(x='Gender', y='ProductPrice', data=avg_price_gender, palette='Set2', co
plt.title("Average Spend by Gender")
plt.ylabel("Average Product Price (₦)")
plt.xlabel("Gender")
plt.show()
```



In [25]:
```
#MODEL PREDICTION

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

In [28]:
```
data = pd.read_csv("C:/Users/HP/Desktop/Cleaned_Groceries_dataset.csv")
data.head()
```

Out[28]:

	Customer_ID_number	OrderDate	Items	ItemCategory	Gender	Geographic location	Purcha Histo
0	1808	3/4/2025	Indomie Instant Noodles	Food	F	Ikeja	4/30/20
1	2552	1/1/2025	Golden Morn	Food	M	Iyana ipaja	5/4/20
2	2300	3/2/2025	Spaghetti	Food	M	Mafoluku	4/12/20
3	1187	3/30/2025	Maggi Cubes	Food Seasoning	M	Ikeja	2/12/20
4	3037	2/20/2025	Golden Morn	Food	F	Festac town	6/18/20

In [29]:

```
# Define target (what we want to predict)
target = 'Items'

# Define features
features = ['Age', 'Gender', 'Geographic location', 'ProductPrice', 'OrderQuantity'
```

In [30]:

```
#Encode Categorical Features

data_encoded = data[features + [target]].copy()

# Encode Gender
le_gender = LabelEncoder()
data_encoded['Gender'] = le_gender.fit_transform(data_encoded['Gender'])

# Encode Location
le_location = LabelEncoder()
data_encoded['Geographic location'] = le_location.fit_transform(data_encoded['Geogr

# Encode target column (Items)
le_items = LabelEncoder()
data_encoded['Items'] = le_items.fit_transform(data_encoded['Items'])
```

In [31]:

```
#Train/Testsplit
X = data_encoded.drop(columns=[target])
y = data_encoded[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [32]:

```
#Train a random forest classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Out[32]:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [33]:

```
#Make predictions and evaluate the model
y_pred = model.predict(X_test)

# Evaluation
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy Score: 0.8062685412098543

Classification Report:
              precision    recall  f1-score   support

           0       0.23      0.14      0.17       172
           1       0.00      0.00      0.00         8
           2       0.97      0.62      0.76        56
           3       0.69      0.38      0.49        24
           4       0.50      0.18      0.27        22
           5       0.92      1.00      0.96       159
           6       0.43      0.41      0.42        92
           7       1.00      1.00      1.00       207
           8       0.86      1.00      0.92        74
           9       0.79      0.86      0.82        79
          10       0.90      1.00      0.95       170
          11       0.83      0.96      0.89       106
          12       0.31      0.38      0.34       231
          13       1.00      1.00      1.00       207
          14       0.83      0.87      0.85        63
          15       0.90      0.73      0.81        77
          16       0.80      0.99      0.88       138
          17       0.89      0.67      0.76        51
          18       0.88      0.88      0.88        34
          19       0.99      0.98      0.99       163
          20       1.00      0.57      0.73        21
          21       0.21      0.22      0.22       147
          22       0.76      0.98      0.86       172
          23       0.20      0.15      0.17       141
          24       0.74      0.84      0.78       495
          25       0.99      0.94      0.96        93
          26       0.00      0.00      0.00        11
          27       0.65      0.78      0.70       183
          28       0.84      0.76      0.80        80
          29       0.88      0.64      0.74        47
          30       1.00      1.00      1.00       213
          31       1.00      1.00      1.00       307
          32       0.83      0.82      0.83        73
          33       0.58      0.73      0.65        99
          34       1.00      1.00      1.00       223
          35       0.86      0.96      0.91       123
          36       1.00      0.98      0.99        66
          37       0.15      0.14      0.15       133
          38       0.85      0.97      0.90       375
          39       1.00      0.17      0.29        12
          40       1.00      1.00      1.00       259
          41       1.00      1.00      1.00       183
          42       0.21      0.04      0.07        69
          43       0.98      0.92      0.95        59
          44       0.89      0.82      0.85        68
          45       0.95      0.99      0.97        92
          46       1.00      1.00      1.00       100
          47       0.94      0.98      0.96        97
          48       0.39      0.25      0.30        68
          49       0.92      0.94      0.93        72
          50       0.96      0.98      0.97       139
```

```
             51       0.43      0.32      0.37       102
             52       0.93      0.82      0.87        50
             53       0.63      0.36      0.46        33
             54       0.78      0.82      0.79        38
             55       0.82      0.56      0.67        16
             56       0.10      0.07      0.09        69
             57       0.71      0.17      0.27        30
             58       0.99      1.00      0.99       335
             59       0.96      1.00      0.98       258
             60       1.00      1.00      1.00       219
             61       0.78      0.29      0.42        24
             62       0.47      0.49      0.48        97
             63       1.00      0.98      0.99        98
             64       0.72      0.68      0.70        31

       accuracy                           0.81      7753
      macro avg       0.75      0.70      0.71      7753
   weighted avg       0.79      0.81      0.79      7753


Confusion Matrix:
 [[24  0  0 ...  0  0  0]
 [ 0  0  0 ...  0  0  0]
 [ 0  0 35 ...  0  0  0]
 ...
 [ 0  0  0 ... 48  0  0]
 [ 0  0  0 ...  0 96  0]
 [ 0  0  0 ...  0  0 21]]
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [34]:

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(class_weight='balanced')
model.fit(X_train, y_train)
```

Out[34]:

```
▾              RandomForestClassifier

RandomForestClassifier(class_weight='balanced')
```

In [35]:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Make predictions
y_pred = model.predict(X_test)

# Evaluation
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred, zero_divi
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy Score: 0.8097510641042177

Classification Report:
              precision    recall  f1-score   support

           0       0.23      0.15      0.18       172
           1       1.00      0.25      0.40         8
           2       1.00      0.93      0.96        56
           3       1.00      0.71      0.83        24
           4       1.00      0.91      0.95        22
           5       1.00      1.00      1.00       159
           6       0.39      0.33      0.36        92
           7       0.93      0.95      0.94       207
           8       0.89      0.89      0.89        74
           9       0.98      1.00      0.99        79
          10       0.85      0.99      0.92       170
          11       0.88      0.98      0.93       106
          12       0.31      0.35      0.33       231
          13       1.00      1.00      1.00       207
          14       0.92      0.95      0.94        63
          15       1.00      0.82      0.90        77
          16       0.92      1.00      0.96       138
          17       0.97      0.67      0.79        51
          18       1.00      1.00      1.00        34
          19       1.00      0.99      1.00       163
          20       1.00      0.33      0.50        21
          21       0.25      0.26      0.25       147
          22       0.87      1.00      0.93       172
          23       0.25      0.18      0.21       141
          24       0.70      0.83      0.76       495
          25       0.99      0.82      0.89        93
          26       0.86      0.55      0.67        11
          27       0.62      0.72      0.66       183
          28       0.95      0.96      0.96        80
          29       1.00      0.83      0.91        47
          30       1.00      1.00      1.00       213
          31       0.91      1.00      0.95       307
          32       1.00      0.99      0.99        73
          33       0.58      0.68      0.63        99
          34       0.97      0.98      0.98       223
          35       0.96      0.96      0.96       123
          36       0.98      0.82      0.89        66
          37       0.14      0.15      0.15       133
          38       0.84      0.97      0.90       375
          39       1.00      0.25      0.40        12
          40       0.86      0.98      0.92       259
          41       0.93      0.92      0.93       183
          42       0.28      0.07      0.11        69
          43       0.98      0.76      0.86        59
          44       0.98      0.85      0.91        68
          45       0.91      0.93      0.92        92
          46       0.99      0.89      0.94       100
          47       0.99      0.94      0.96        97
          48       0.40      0.31      0.35        68
          49       0.99      0.97      0.98        72
          50       0.99      0.99      0.99       139
```

```
          51       0.36      0.25      0.30       102
          52       1.00      1.00      1.00        50
          53       0.96      0.70      0.81        33
          54       1.00      0.84      0.91        38
          55       1.00      1.00      1.00        16
          56       0.12      0.09      0.10        69
          57       1.00      0.60      0.75        30
          58       0.94      1.00      0.97       335
          59       0.91      0.99      0.95       258
          60       0.98      1.00      0.99       219
          61       1.00      0.46      0.63        24
          62       0.45      0.42      0.44        97
          63       0.93      0.79      0.85        98
          64       0.91      0.97      0.94        31

    accuracy                           0.81      7753
   macro avg       0.83      0.75      0.77      7753
weighted avg       0.80      0.81      0.80      7753


Confusion Matrix:
 [[25  0  0 ...  0  0  0]
 [ 0  2  0 ...  0  0  0]
 [ 0  0 52 ...  0  0  0]
 ...
 [ 1  0  0 ... 41  0  0]
 [ 1  0  0 ...  0 77  0]
 [ 0  0  0 ...  0  0 30]]
```

In [36]:

```
#Predict Item for a New Customer
# Encode new customer input
#Predict what a 35-year-old female from Ikeja who buys 4 items priced at ₦800 might
new_gender = le_gender.transform(['F'])[0]
new_location = le_location.transform(['Ikeja'])[0]

# Example input
new_customer = pd.DataFrame([{
    'Age': 35,
    'Gender': new_gender,
    'Geographic location': new_location,
    'ProductPrice': 800,
    'OrderQuantity': 4
}])

# Predict and decode the item
predicted_item_encoded = model.predict(new_customer)[0]
predicted_item = le_items.inverse_transform([predicted_item_encoded])[0]

print("Predicted Product:", predicted_item)
```

```
Predicted Product: Lipton Tea
```

In [39]:

```
#MODEL COMPARISON: Logistic Regression vs. Decision Tree vs. Random Forest
#Goal: Predict Items using demographic and transaction data.

from sklearn.linear_model import LogisticRegression
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

In [40]:
```
#define Feature and target

X = data_encoded.drop(columns=['Items'])
y = data_encoded['Items']
```

In [41]:
```
#split the dataset for final evaluation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [42]:
```
#Initialize the model

# Initialize models
log_reg = LogisticRegression(max_iter=1000)
decision_tree = DecisionTreeClassifier(random_state=42)
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [49]:
```
#Cross-validation for reliability

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform 5-fold cross-validation with scaled data
cv_log_reg = cross_val_score(log_reg, X_scaled, y, cv=5)
cv_tree = cross_val_score(decision_tree, X, y, cv=5)
cv_rf = cross_val_score(random_forest, X, y, cv=5)

print("Logistic Regression Accuracy (CV):", round(cv_log_reg.mean()*100, 2), "%")
print("Decision Tree Accuracy (CV):", round(cv_tree.mean()*100, 2), "%")
print("Random Forest Accuracy (CV):", round(cv_rf.mean()*100, 2), "%")
```

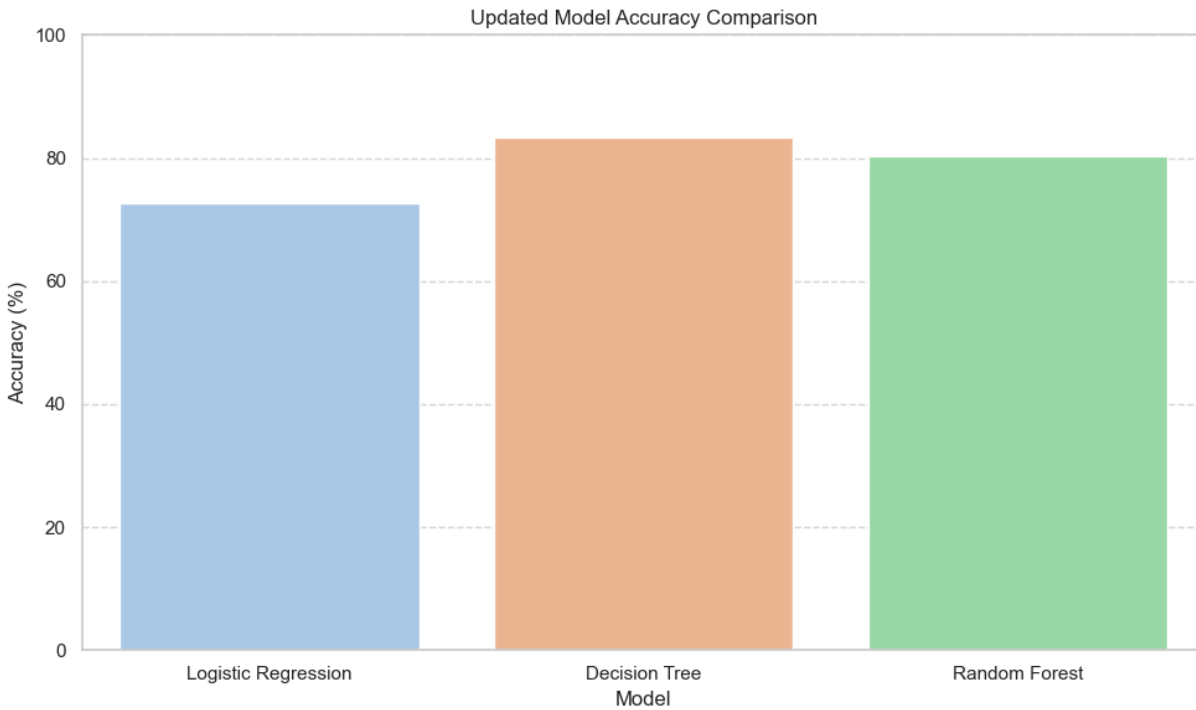
```
Logistic Regression Accuracy (CV): 30.46 %
Decision Tree Accuracy (CV): 83.44 %
Random Forest Accuracy (CV): 80.29 %
```

In [50]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Updated model accuracy values
updated_results = {
    'Logistic Regression': 72.60,
    'Decision Tree': 83.44,
    'Random Forest': 80.29
}

# Create DataFrame
updated_results_df = pd.DataFrame(list(updated_results.items()), columns=['Model',
```

```
# Plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Accuracy', data=updated_results_df, palette='pastel')
plt.title('Updated Model Accuracy Comparison')
plt.ylabel('Accuracy (%)')
plt.ylim(0, 100)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```




```
In [ ]:
```