In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve, precision_recall_fscore_support
from datetime import datetime
import os
```

In [2]:
```
INPUT_CSV = "C:\\datasets\\ecommerce_returns_synthetic_data.csv"
OUT_DIR = "ecomm_outputs"
RISK_THRESHOLD = 0.50
RANDOM_STATE = 42
```

In [3]:
```
os.makedirs(OUT_DIR, exist_ok=True)
```

In [4]:
```
df = pd.read_csv(INPUT_CSV)
```

In [5]:
```
df
```

Out[5]:

	Order_ID	Product_ID	User_ID	Order_Date	Return_Date	Product_Category	Product_Price	Order_Quantity	Return_Reason	Return_Status	Days_to_Return	User_Age	User_Gender	User_Location	Payment_Method	Shipping_Method	Discount_Applied
0	ORD00000000	PROD00000000	USER00000000	2023-08-05	2024-08-26	Clothing	411.59	3	Changed mind	Returned	387.0	58	Male	City54	Debit Card	Next-Day	45.27
1	ORD00000001	PROD00000001	USER00000001	2023-10-09	2023-11-09	Books	288.88	3	Wrong item	Returned	31.0	68	Female	City85	Credit Card	Express	47.79
2	ORD00000002	PROD00000002	USER00000002	2023-05-06	NaN	Toys	390.03	5	NaN	Not Returned	NaN	22	Female	City30	Debit Card	Next-Day	26.64
3	ORD00000003	PROD00000003	USER00000003	2024-08-29	NaN	Toys	401.09	3	NaN	Not Returned	NaN	40	Male	City95	PayPal	Next-Day	15.37
4	ORD00000004	PROD00000004	USER00000004	2023-01-16	NaN	Books	110.09	4	NaN	Not Returned	NaN	34	Female	City80	Gift Card	Standard	16.37
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	ORD00009995	PROD00009995	USER00009995	2023-10-20	NaN	Home	142.50	4	NaN	Not Returned	NaN	37	Male	City40	PayPal	Standard	34.27
9996	ORD00009996	PROD00009996	USER00009996	2023-02-25	NaN	Electronics	484.63	3	NaN	Not Returned	NaN	69	Male	City62	Debit Card	Express	25.44
9997	ORD00009997	PROD00009997	USER00009997	2024-05-10	NaN	Toys	386.57	5	NaN	Not Returned	NaN	46	Male	City74	Credit Card	Next-Day	12.67
9998	ORD00009998	PROD00009998	USER00009998	2024-02-13	NaN	Toys	129.22	1	NaN	Not Returned	NaN	34	Female	City34	Gift Card	Express	49.97
9999	ORD00009999	PROD00009999	USER00009999	2024-12-08	2024-09-09	Toys	459.42	5	Not as described	Returned	-90.0	24	Female	City51	Debit Card	Express	16.05

10000 rows × 17 columns

In [6]:
```
print("Rows,cols:", df.shape)
print(df.columns.tolist())
print(df.head(1).T)
```

```
Rows,cols: (10000, 17)
['Order_ID', 'Product_ID', 'User_ID', 'Order_Date', 'Return_Date', 'Product_Category', 'Product_Price', 'Order_Quantity', 'Return_Reason', 'Return_Status', 'Days_to_Return', 'User_Age', 'User_Gender', 'User_Location', 'Payment_Method', 'Shipping_Method', 'Discount_Applied']
                          0
Order_ID          ORD00000000
Product_ID        PROD00000000
User_ID           USER00000000
Order_Date          2023-08-05
Return_Date         2024-08-26
Product_Category      Clothing
Product_Price           411.59
Order_Quantity               3
Return_Reason     Changed mind
Return_Status         Returned
Days_to_Return           387.0
User_Age                    58
User_Gender               Male
User_Location           City54
Payment_Method      Debit Card
Shipping_Method       Next-Day
Discount_Applied         45.27
```

In [7]:
```
df.columns = [c.strip() for c in df.columns]
```

In [8]:
```
for col in ["Order_Date", "Return_Date"]:
    if col in df.columns:
        df[col] = pd.to_datetime(df[col], dayfirst=False, errors='coerce')
```

In [9]:
```
if "Days_to_Return" not in df.columns or df["Days_to_Return"].isnull().any():
    df["Days_to_Return"] = (df["Return_Date"] - df["Order_Date"]).dt.days
    df["Days_to_Return"] = df["Days_to_Return"].fillna(-1).astype(int)
```

In [10]:
```
if "Return_Status" in df.columns:
    df["Return_Flag"] = df["Return_Status"].str.lower().map(lambda x: 1 if str(x).strip().startswith("ret") else 0)
else:
    df["Return_Flag"] = df["Return_Date"].notna().astype(int)
```

In [11]:
```
if "Product_Price" in df.columns:
    df["Product_Price"] = pd.to_numeric(df["Product_Price"], errors='coerce')
else:
    df["Product_Price"] = pd.to_numeric(df.get("Price", None), errors='coerce')
```

In [12]:
```
if "Order_Quantity" in df.columns:
    df["Order_Quantity"] = pd.to_numeric(df["Order_Quantity"], errors='coerce').fillna(1).astype(int)
```

```
else:
    df["Order_Quantity"] = 1
```

In [13]:
```
if "Discount_Applied" in df.columns:
    df["Discount_Applied"] = pd.to_numeric(df["Discount_Applied"], errors='coerce').fillna(0.0)
```

In [14]:
```
df["Days_to_Return_clean"] = df["Days_to_Return"].replace(-1, np.nan)
```

In [15]:
```
if "Product_Category" in df.columns:
    cat_summary = df.groupby("Product_Category").agg(
        orders_count = ("Order_ID","count"),
        return_rate = ("Return_Flag","mean")
    ).reset_index().sort_values("return_rate", ascending=False)
    print("\nReturn rate by Category:\n", cat_summary)
    cat_summary.to_csv(os.path.join(OUT_DIR,"return_rate_by_category.csv"), index=False)
```

```
Return rate by Category:
   Product_Category  orders_count  return_rate
1         Clothing          2000     0.524500
2      Electronics          1985     0.509320
0            Books          2041     0.506614
4             Toys          1944     0.495370
3             Home          2030     0.490148
```

In [16]:
```
if "User_Location" in df.columns:
    loc_summary = df.groupby("User_Location").agg(
        orders_count = ("Order_ID","count"),
        return_rate = ("Return_Flag","mean")
    ).reset_index().sort_values("return_rate", ascending=False)
    loc_summary.to_csv(os.path.join(OUT_DIR,"return_rate_by_location.csv"), index=False)
    print("\nSaved return_rate_by_location.csv")
```

```
Saved return_rate_by_location.csv
```

In [17]:
```
for col in ["Payment_Method","Shipping_Method","User_Gender","Payment_Channel","Marketing_Channel","marketing_channel"]:
    if col in df.columns:
        s = df.groupby(col).agg(orders_count=("Order_ID","count"), return_rate=("Return_Flag","mean")).reset_index().sort_values("return_rate", ascending=False)
        s.to_csv(os.path.join(OUT_DIR,f"return_rate_by_{col}.csv"), index=False)
        print(f"Saved return_rate_by_{col}.csv")
```

```
Saved return_rate_by_Payment_Method.csv
Saved return_rate_by_Shipping_Method.csv
Saved return_rate_by_User_Gender.csv
```

In [18]:
```
if "User_Age" in df.columns:
    df["age_bucket"] = pd.cut(df["User_Age"].fillna(0), bins=[0,18,25,35,45,55,65,120], labels=["<=18","19-25","26-35","36-45","46-55","56-65","65+"])
    age_summary = df.groupby("age_bucket").agg(orders_count=("Order_ID","count"), return_rate=("Return_Flag","mean")).reset_index()
    age_summary.to_csv(os.path.join(OUT_DIR,"return_rate_by_age_bucket.csv"), index=False)
```

```
C:\Users\harsh\AppData\Local\Temp\ipykernel_20828\3676866813.py:3: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  age_summary = df.groupby("age_bucket").agg(orders_count=("Order_ID","count"), return_rate=("Return_Flag","mean")).reset_index()
```

In [19]:
```
try:
    sns.set(style="whitegrid")
    if "Product_Category" in df.columns:
        plt.figure(figsize=(8,5))
        order = cat_summary.sort_values("return_rate", ascending=False)
        sns.barplot(y="Product_Category", x="return_rate", data=order)
        plt.xlabel("Return Rate")
        plt.title("Return Rate by Product Category")
        plt.tight_layout()
        plt.savefig(os.path.join(OUT_DIR,"return_rate_by_category.png"))
        plt.close()
except Exception as e:
    print("Plotting failed:", e)
```

In [20]:
```
features = []
if "Product_Category" in df.columns: features.append("Product_Category")
if "Product_Price" in df.columns: features.append("Product_Price")
if "Order_Quantity" in df.columns: features.append("Order_Quantity")
if "Days_to_Return_clean" in df.columns: features.append("Days_to_Return_clean")
if "User_Age" in df.columns: features.append("User_Age")
if "User_Gender" in df.columns: features.append("User_Gender")
if "User_Location" in df.columns: features.append("User_Location")
if "Payment_Method" in df.columns: features.append("Payment_Method")
if "Shipping_Method" in df.columns: features.append("Shipping_Method")
if "Discount_Applied" in df.columns: features.append("Discount_Applied")

print("Using features:", features)

if "User_Location" in features:
    top_loc = df["User_Location"].value_counts().nlargest(30).index.tolist()
    df["User_Location_reduced"] = df["User_Location"].where(df["User_Location"].isin(top_loc), other="Other")
    features = [("User_Location_reduced" if f=="User_Location" else f) for f in features]
```

```
Using features: ['Product_Category', 'Product_Price', 'Order_Quantity', 'Days_to_Return_clean', 'User_Age', 'User_Gender', 'User_Location', 'Payment_Method', 'Shipping_Method', 'Discount_Applied']
```

In [21]:
```
y = df["Return_Flag"]
```

In [22]:
```
mask_valid = y.notna()
X = df.loc[mask_valid, features].copy()
y = y.loc[mask_valid].astype(int)
```

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=RANDOM_STATE, stratify=y)
```

In [26]:
```
# Assuming you already have your full feature list
features = ["Product_Category","Product_Price","Order_Quantity","Days_to_Return_clean",
            "User_Age","User_Gender","User_Location_reduced","Payment_Method","Shipping_Method","Discount_Applied"]

# Identify numeric and categorical features
```

```
num_features = [c for c in features if df[c].dtype.kind in 'fi']  # floats or ints
cat_features = [c for c in features if c not in num_features]

print("Numeric features:", num_features)
print("Categorical features:", cat_features)
```

```
Numeric features: ['Product_Price', 'Order_Quantity', 'Days_to_Return_clean', 'User_Age', 'Discount_Applied']
Categorical features: ['Product_Category', 'User_Gender', 'User_Location_reduced', 'Payment_Method', 'Shipping_Method']
```

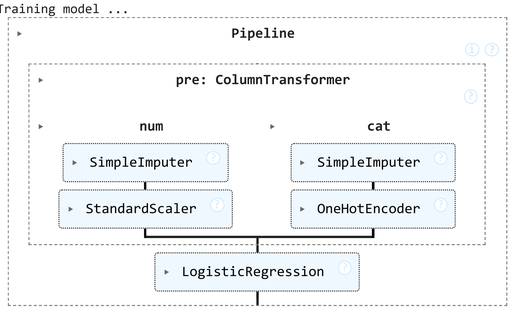
In [27]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler

# Numeric features
num_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),   # fills NaN with median
    ('scaler', StandardScaler())
])

# Categorical features
cat_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='Missing')),  # fills NaN
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False)) # use sparse_output
])

# Column transformer
preprocessor = ColumnTransformer(transformers=[
    ('num', num_transformer, num_features),
    ('cat', cat_transformer, cat_features)
])
```

In [28]:
```
clf = Pipeline(steps=[
    ("pre", preprocessor),
    ("model", LogisticRegression(max_iter=1000, class_weight="balanced", solver="liblinear", random_state=RANDOM_STATE))
])
```

In [29]:
```
print("Training model ...")
clf.fit(X_train, y_train)
```

```
Training model ...
```

Out[29]:



In [30]:
```
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)[:,1]
print("\nClassification report (test):\n", classification_report(y_test, y_pred, digits=4))
print("ROC AUC:", roc_auc_score(y_test, y_proba))
```

```
Classification report (test):
              precision    recall  f1-score   support

           0     0.4943    0.4683    0.4810      1484
           1     0.5050    0.5310    0.5177      1516

    accuracy                         0.5000      3000
   macro avg     0.4997    0.4997    0.4993      3000
weighted avg     0.4997    0.5000    0.4995      3000

ROC AUC: 0.5080591391731681
```

In [31]:
```
joblib.dump(clf, os.path.join(OUT_DIR,"logistic_pipeline.joblib"))
print("Saved pipeline to", os.path.join(OUT_DIR,"logistic_pipeline.joblib"))
```

```
Saved pipeline to ecomm_outputs\logistic_pipeline.joblib
```

In [32]:
```
try:
    ohe = clf.named_steps['pre'].named_transformers_['cat']
    ohe_feature_names = []
    if hasattr(ohe, "get_feature_names_out"):
        ohe_feature_names = list(ohe.get_feature_names_out(cat_features))
    num_names = num_features
    all_feature_names = num_names + ohe_feature_names
    coefs = clf.named_steps['model'].coef_[0]
    feat_imp = pd.DataFrame({"feature": all_feature_names, "coef": coefs})
    feat_imp = feat_imp.reindex(feat_imp.coef.abs().sort_values(ascending=False).index)
    feat_imp.head(30).to_csv(os.path.join(OUT_DIR,"feature_coefficients.csv"), index=False)
except Exception as e:
    print("Could not extract feature names:", e)
```

In [33]:
```
full_proba = clf.predict_proba(X)[:,1]
df.loc[mask_valid, "pred_return_prob"] = full_proba
# For rows that were invalid, fill 0
df["pred_return_prob"] = df["pred_return_prob"].fillna(0.0)
```

In [34]:
```
if "Product_ID" in df.columns:
    product_risk = df.groupby(["Product_ID","Product_Category"]).agg(
        avg_pred_return_prob = ("pred_return_prob","mean"),
        actual_return_rate = ("Return_Flag","mean"),
        orders_count = ("Order_ID","count")
    ).reset_index().sort_values("avg_pred_return_prob", ascending=False)
else:
    # fallback: product name
    product_risk = df.groupby(["Product_Category"]).agg(
        avg_pred_return_prob = ("pred_return_prob","mean"),
        actual_return_rate = ("Return_Flag","mean"),
        orders_count = ("Order_ID","count")
    ).reset_index().sort_values("avg_pred_return_prob", ascending=False)

product_risk.to_csv(os.path.join(OUT_DIR,"product_return_risk.csv"), index=False)
print("Saved product_return_risk.csv")
```

```
Saved product_return_risk.csv
```

In [35]:
```
high_risk = product_risk[product_risk["avg_pred_return_prob"] >= RISK_THRESHOLD].sort_values("avg_pred_return_prob", ascending=False)
high_risk.to_csv(os.path.join(OUT_DIR,"high_risk_products.csv"), index=False)
print("Saved high_risk_products.csv (threshold {})".format(RISK_THRESHOLD))
```

```
Saved high_risk_products.csv (threshold 0.5)
```

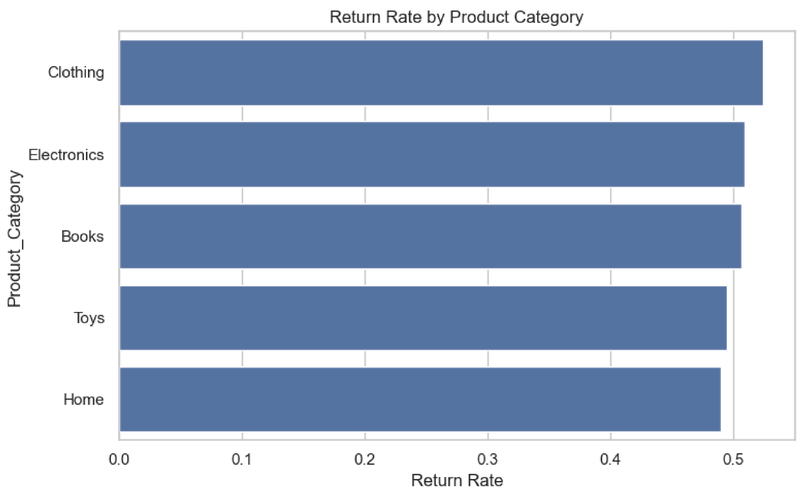
In [36]:
```
df.to_csv(os.path.join(OUT_DIR,"ecomm_clean.csv"), index=False)
print("Saved cleaned dataset ecomm_clean.csv")
```

```
Saved cleaned dataset ecomm_clean.csv
```

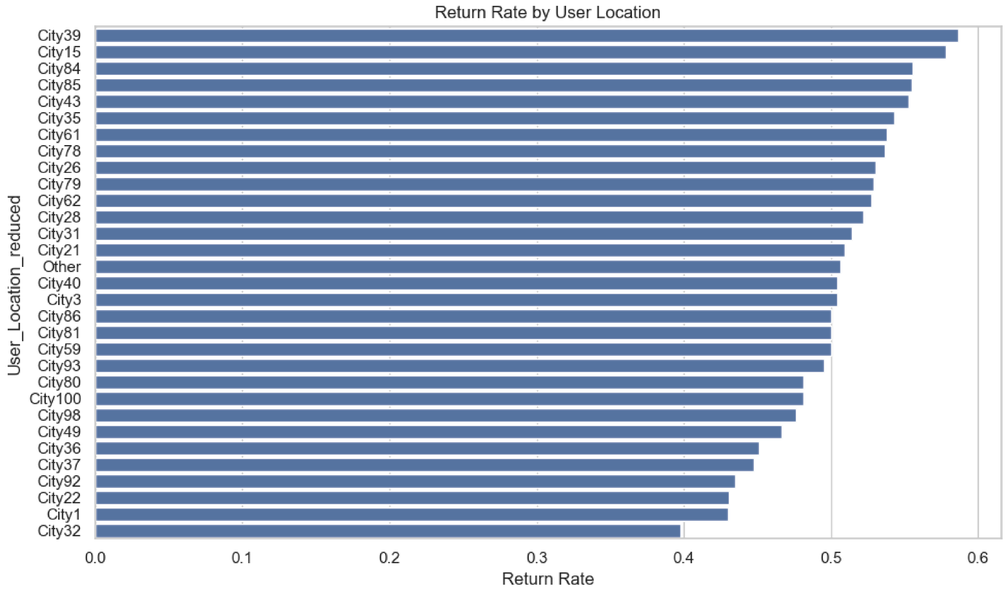
In [37]:
```
metrics = {
    "roc_auc": roc_auc_score(y_test, y_proba),
}
pd.DataFrame.from_dict(metrics, orient="index", columns=["value"]).to_excel(os.path.join(OUT_DIR,"model_metrics.xlsx"))

print("All outputs saved to", OUT_DIR)
```

```
All outputs saved to ecomm_outputs
```

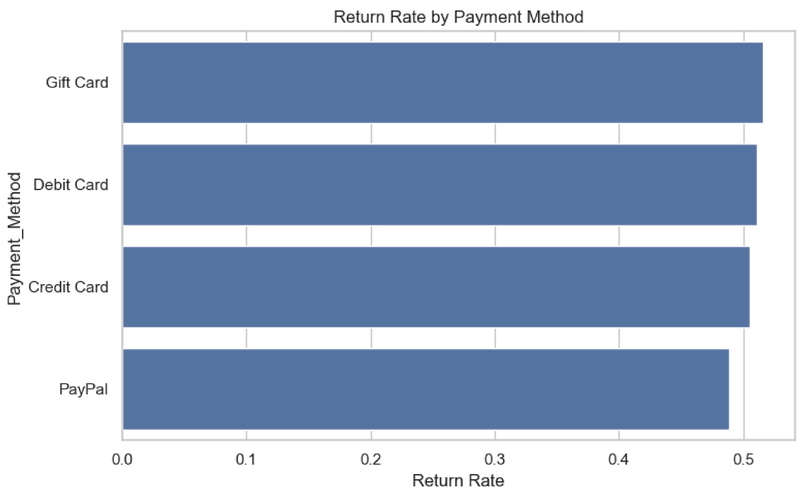
In [41]:
```
plt.figure(figsize=(8,5))
cat_summary = df.groupby("Product_Category")["Return_Flag"].mean().sort_values(ascending=False)
sns.barplot(x=cat_summary.values, y=cat_summary.index)
plt.xlabel("Return Rate")
plt.title("Return Rate by Product Category")
plt.tight_layout()
plt.show()
```



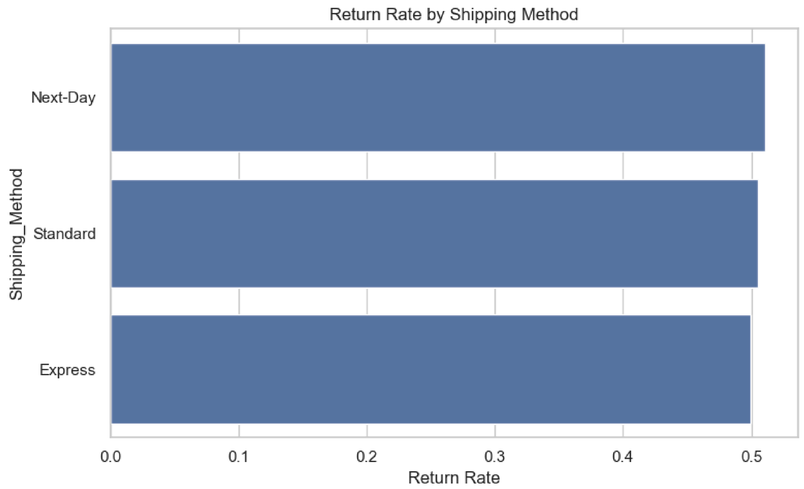
In [42]:
```
plt.figure(figsize=(10,6))
loc_summary = df.groupby("User_Location_reduced")["Return_Flag"].mean().sort_values(ascending=False)
sns.barplot(x=loc_summary.values, y=loc_summary.index)
plt.xlabel("Return Rate")
plt.title("Return Rate by User Location")
plt.tight_layout()
plt.show()
```



In [43]:
```
if "Payment_Method" in df.columns:
    plt.figure(figsize=(8,5))
    pay_summary = df.groupby("Payment_Method")["Return_Flag"].mean().sort_values(ascending=False)
    sns.barplot(x=pay_summary.values, y=pay_summary.index)
    plt.xlabel("Return Rate")
    plt.title("Return Rate by Payment Method")
    plt.tight_layout()
    plt.show()
```

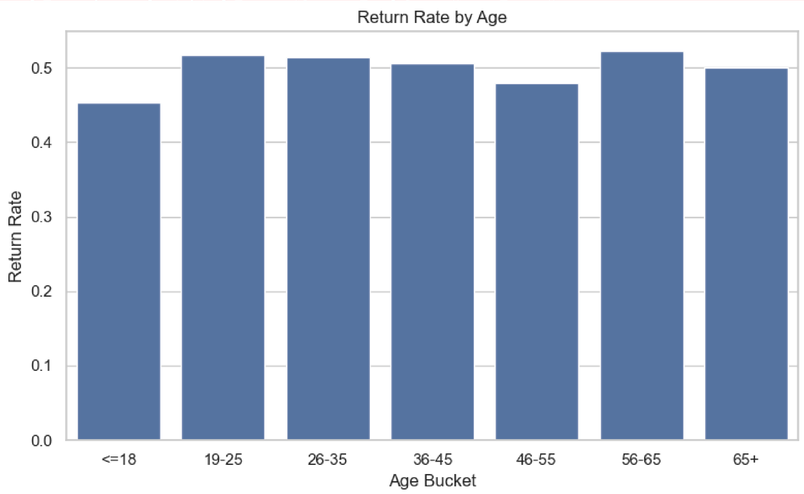


In [44]:
```
if "Shipping_Method" in df.columns:
    plt.figure(figsize=(8,5))
    ship_summary = df.groupby("Shipping_Method")["Return_Flag"].mean().sort_values(ascending=False)
    sns.barplot(x=ship_summary.values, y=ship_summary.index)
    plt.xlabel("Return Rate")
    plt.title("Return Rate by Shipping Method")
    plt.tight_layout()
    plt.show()
```

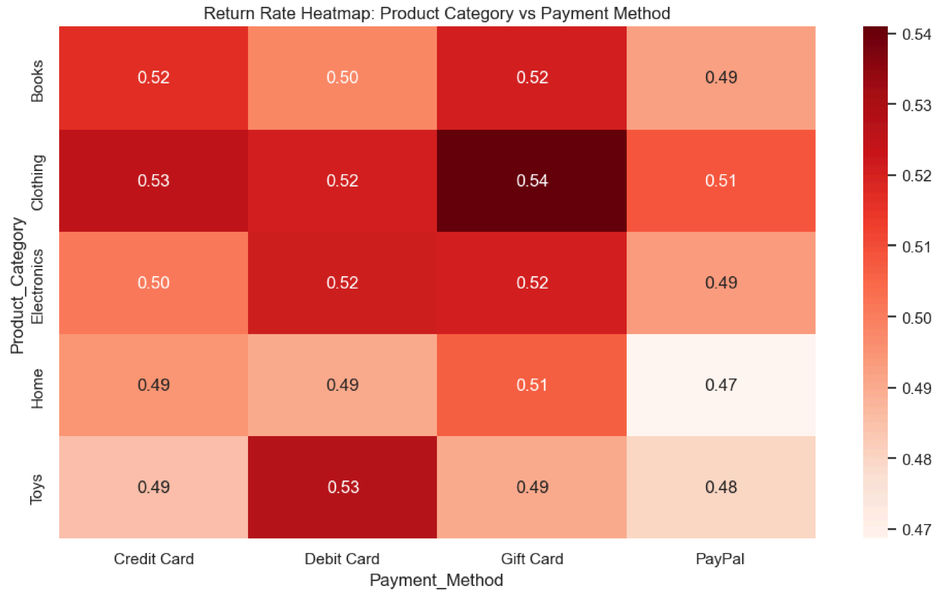


In [45]:
```
plt.figure(figsize=(8,5))
age_summary = df.groupby("age_bucket")["Return_Flag"].mean().sort_index()
sns.barplot(x=age_summary.index, y=age_summary.values)
plt.xlabel("Age Bucket")
plt.ylabel("Return Rate")
plt.title("Return Rate by Age")
plt.tight_layout()
plt.show()
```

```
C:\Users\harsh\AppData\Local\Temp\ipykernel_20828\4011546959.py:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  age_summary = df.groupby("age_bucket")["Return_Flag"].mean().sort_index()
```



In [46]:
```
heat_df = df.pivot_table(index="Product_Category", columns="Payment_Method",
                         values="Return_Flag", aggfunc="mean").fillna(0)
plt.figure(figsize=(10,6))
sns.heatmap(heat_df, annot=True, fmt=".2f", cmap="Reds")
plt.title("Return Rate Heatmap: Product Category vs Payment Method")
plt.tight_layout()
plt.show()
```


Return Rate Heatmap: Product Category vs Payment Method
Product_Category
Books
Clothing
Electronics
Home
Toys
0.52 0.50 0.52 0.49
0.53 0.52 0.54 0.51
0.50 0.52 0.52 0.49
0.49 0.49 0.51 0.47
0.49 0.53 0.49 0.48
Credit Card
Debit Card
Gift Card
PayPal
Payment_Method
0.54
0.53
0.52
0.51
0.50
0.49
0.48
0.47

In [ ]: