```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('Train.csv')
```

```
pd.read_csv('Train.csv')
```

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Custome
0	1	D	Flight	4	
1	2	F	Flight	4	
2	3	A	Flight	2	
3	4	B	Flight	3	
4	5	C	Flight	2	
...	...	...	...	...	
10994	10995	A	Ship	4	
10995	10996	B	Ship	4	
10996	10997	C	Ship	5	
10997	10998	F	Ship	5	
10998	10999	D	Ship	2	

10999 rows × 12 columns

```
df.shape
```

```
(10999, 12)
```

```
df.dtypes
```

```
                              0
                   ID     int64
      Warehouse_block    object
     Mode_of_Shipment    object
  Customer_care_calls     int64
      Customer_rating     int64
  Cost_of_the_Product     int64
      Prior_purchases     int64
   Product_importance    object
               Gender    object
     Discount_offered     int64
        Weight_in_gms     int64
   Reached.on.Time_Y.N    int64

dtype: object
```

```
#Drop column
df.drop(['ID'], axis=1, inplace=True)
```

```
#Checking for null/missing values
df.isnull().sum()
```

```
                          0
   Warehouse_block        0
   Mode_of_Shipment       0
   Customer_care_calls    0
   Customer_rating        0
   Cost_of_the_Product    0
   Prior_purchases        0
   Product_importance     0
   Gender                 0
   Discount_offered       0
   Weight_in_gms          0
   Reached.on.Time_Y.N    0

dtype: int64
```

```
df.duplicated().sum()
```

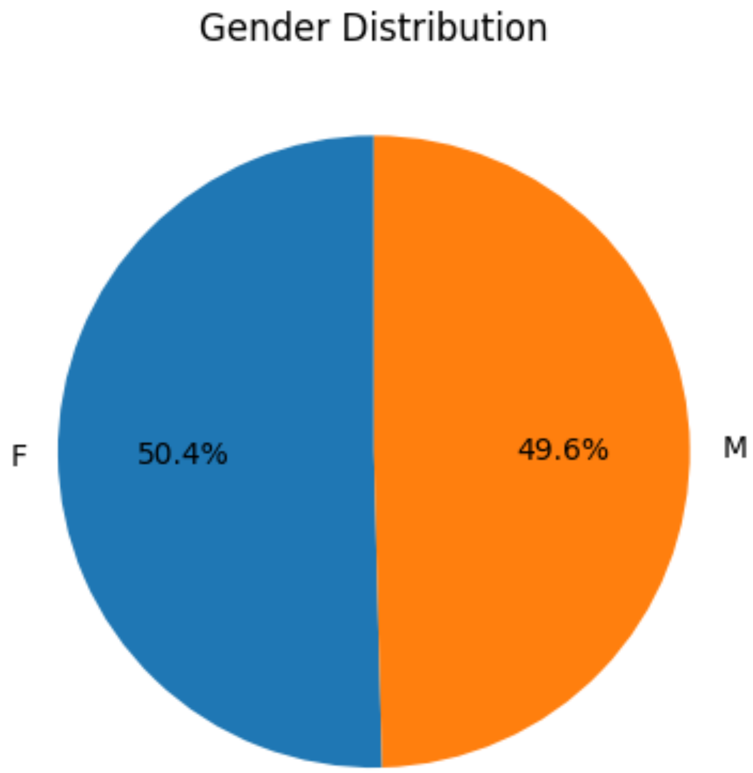
```
0
```

```
df.describe()
```

	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purc
count	10999.000000	10999.000000	10999.000000	10999.0
mean	4.054459	2.990545	210.196836	3.5
std	1.141490	1.413603	48.063272	1.5
min	2.000000	1.000000	96.000000	2.0
25%	3.000000	2.000000	169.000000	3.0
50%	4.000000	3.000000	214.000000	3.0
75%	5.000000	4.000000	251.000000	4.0
max	7.000000	5.000000	310.000000	10.0

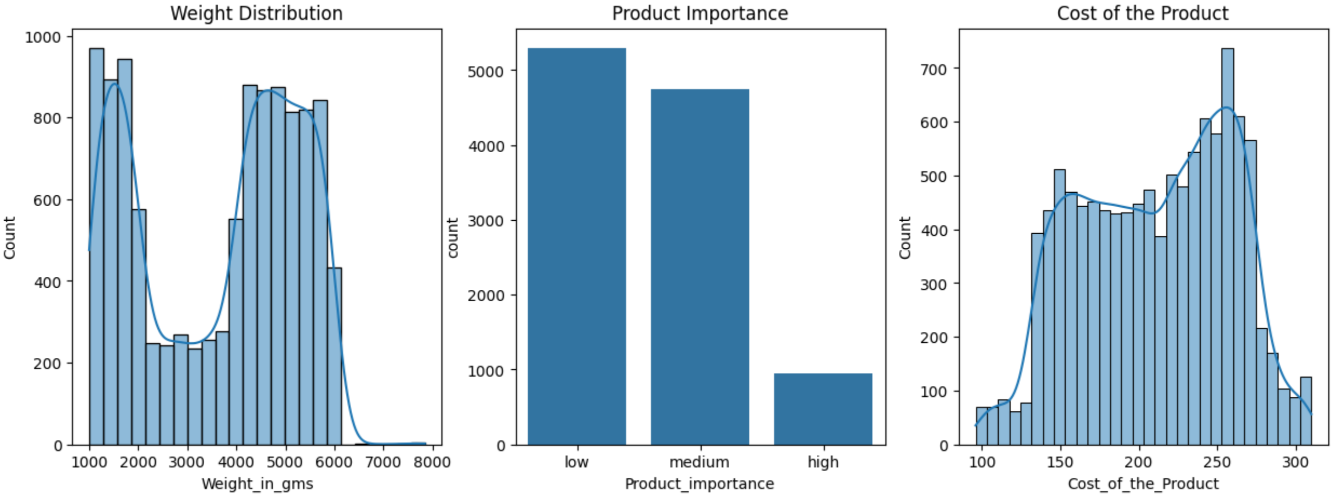
```
plt.pie(df['Gender'].value_counts(),labels = ['F','M'], autopct='%1.1f%%', star
plt.title('Gender Distribution')
```

```
Text(0.5, 1.0, 'Gender Distribution')
```



```
df.replace([np.inf, -np.inf], np.nan, inplace=True)


fig, ax = plt.subplots(1,3,figsize=(15,5))
sns.histplot(df['Weight_in_gms'], ax=ax[0], kde=True).set_title('Weight Distrib
sns.countplot(x = 'Product_importance', data = df, ax=ax[1]).set_title('Product
sns.histplot(df['Cost_of_the_Product'], ax=ax[2], kde=True).set_title('Cost of
```

```
Text(0.5, 1.0, 'Cost of the Product')
```



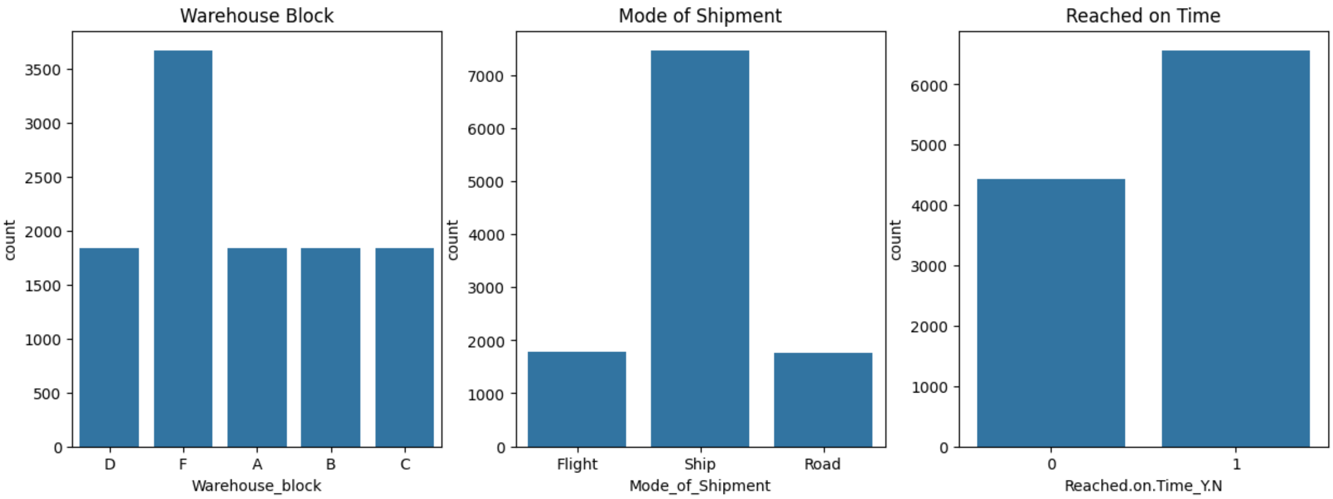
Weight Distribution

Product Importance

Cost of the Product

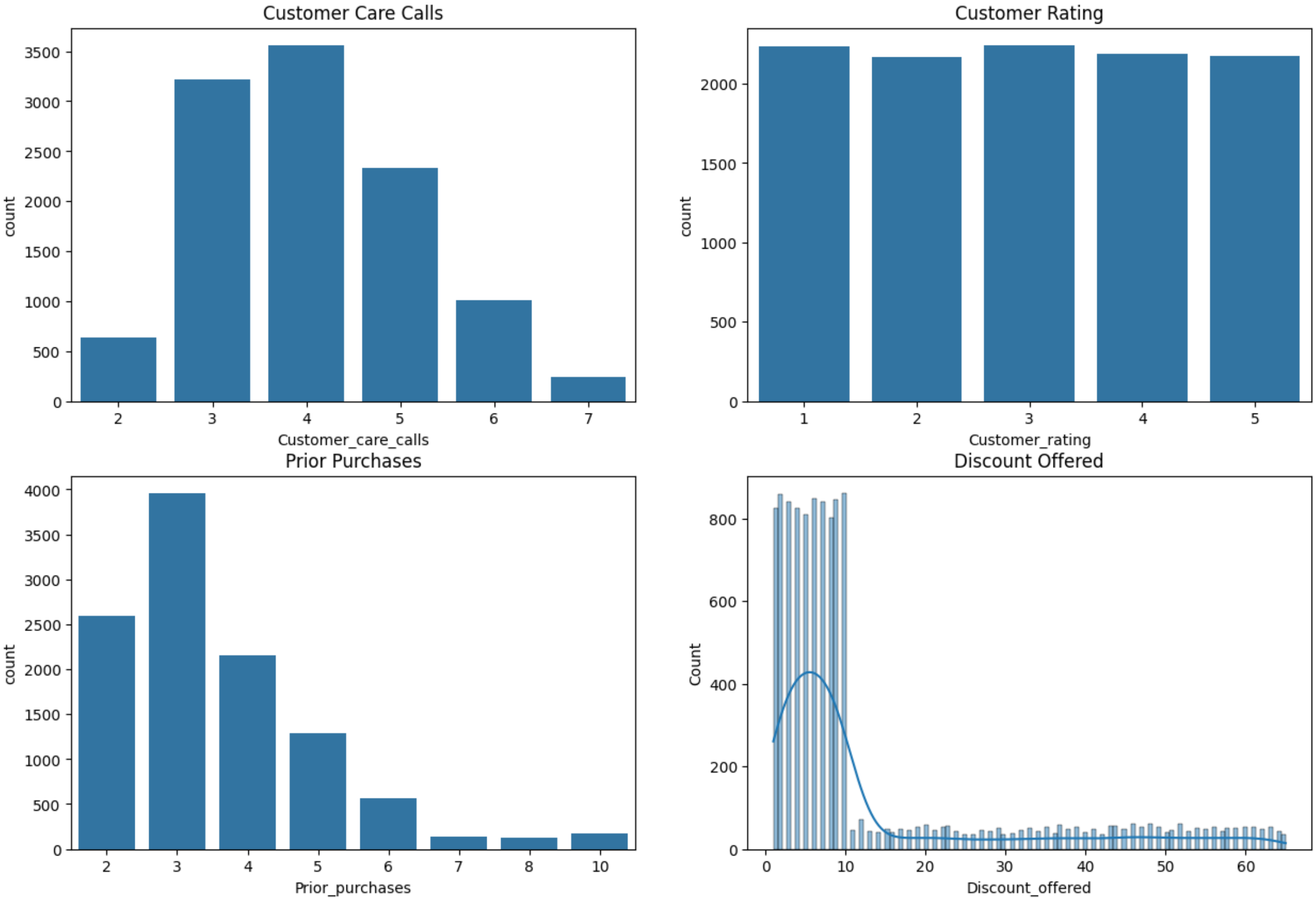
```
fig, ax = plt.subplots(1,3,figsize=(15,5))
sns.countplot(x = 'Warehouse_block', data = df, ax=ax[0]).set_title('Warehouse
sns.countplot(x = 'Mode_of_Shipment', data = df, ax=ax[1]).set_title('Mode of S
sns.countplot(x = 'Reached.on.Time_Y.N', data = df, ax=ax[2]).set_title('Reache
```

```
Text(0.5, 1.0, 'Reached on Time')
```



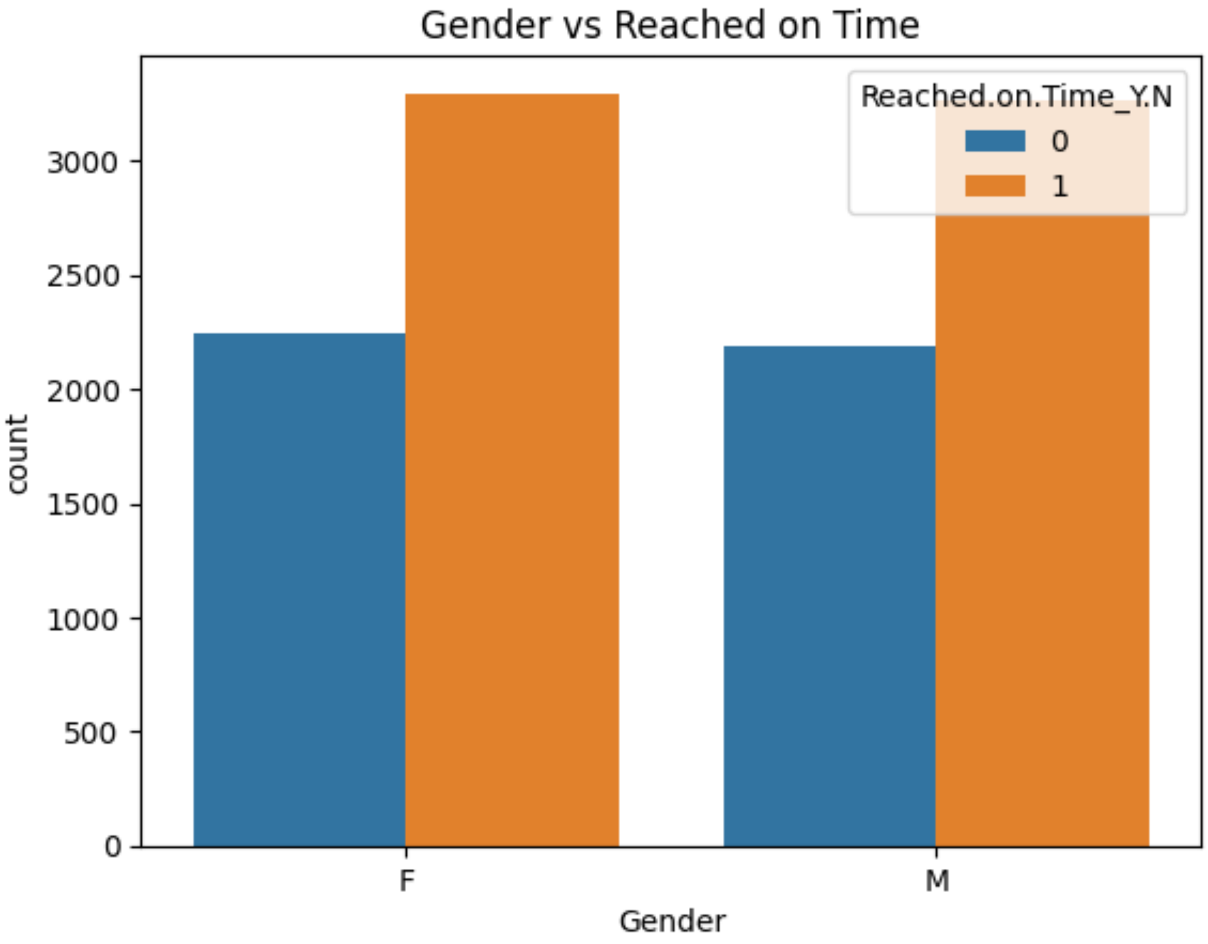
```
fig, ax = plt.subplots(2,2,figsize=(15,10))
sns.countplot(x = 'Customer_care_calls', data = df, ax=ax[0,0]).set_title('Cust
sns.countplot(x = 'Customer_rating', data = df, ax=ax[0,1]).set_title('Customer
sns.countplot(x = 'Prior_purchases', data = df, ax=ax[1,0]).set_title('Prior Pu
sns.histplot(x = 'Discount_offered', data = df, ax=ax[1,1], kde = True).set_tit
```

```
Text(0.5, 1.0, 'Discount Offered')
```



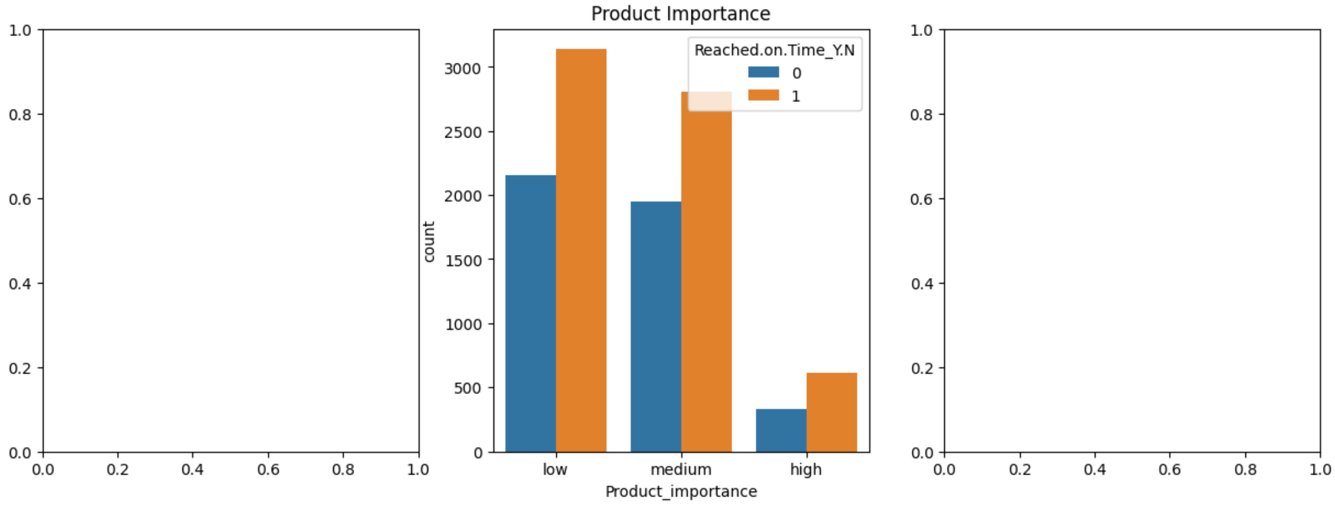
```
sns.countplot(x = 'Gender', data = df, hue = 'Reached.on.Time_Y.N').set_title('
```

```
Text(0.5, 1.0, 'Gender vs Reached on Time')
```



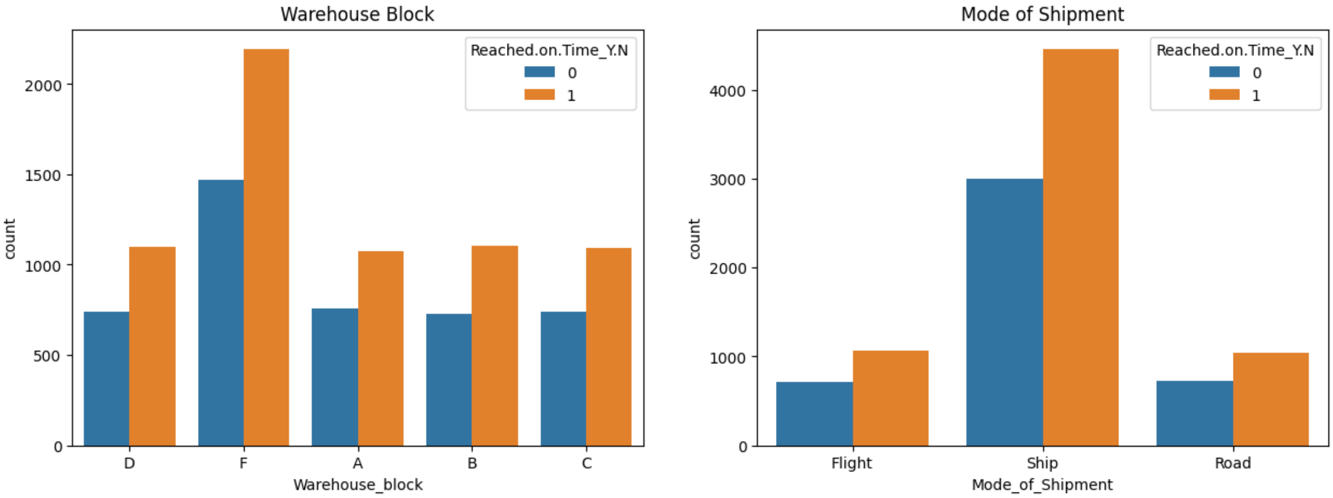
```
fig, ax = plt.subplots(1,3,figsize=(15,5))
# sns.violinplot(y = df['Weight_in_gms'], ax=ax[0], kde=True, x = df['Reached.o
sns.countplot(x = 'Product_importance', data = df, ax=ax[1], hue = 'Reached.on.
# sns.violinplot(y = df['Cost_of_the_Product'], ax=ax[2], kde=True, x = df['Rea
```

```
Text(0.5, 1.0, 'Product Importance')
```



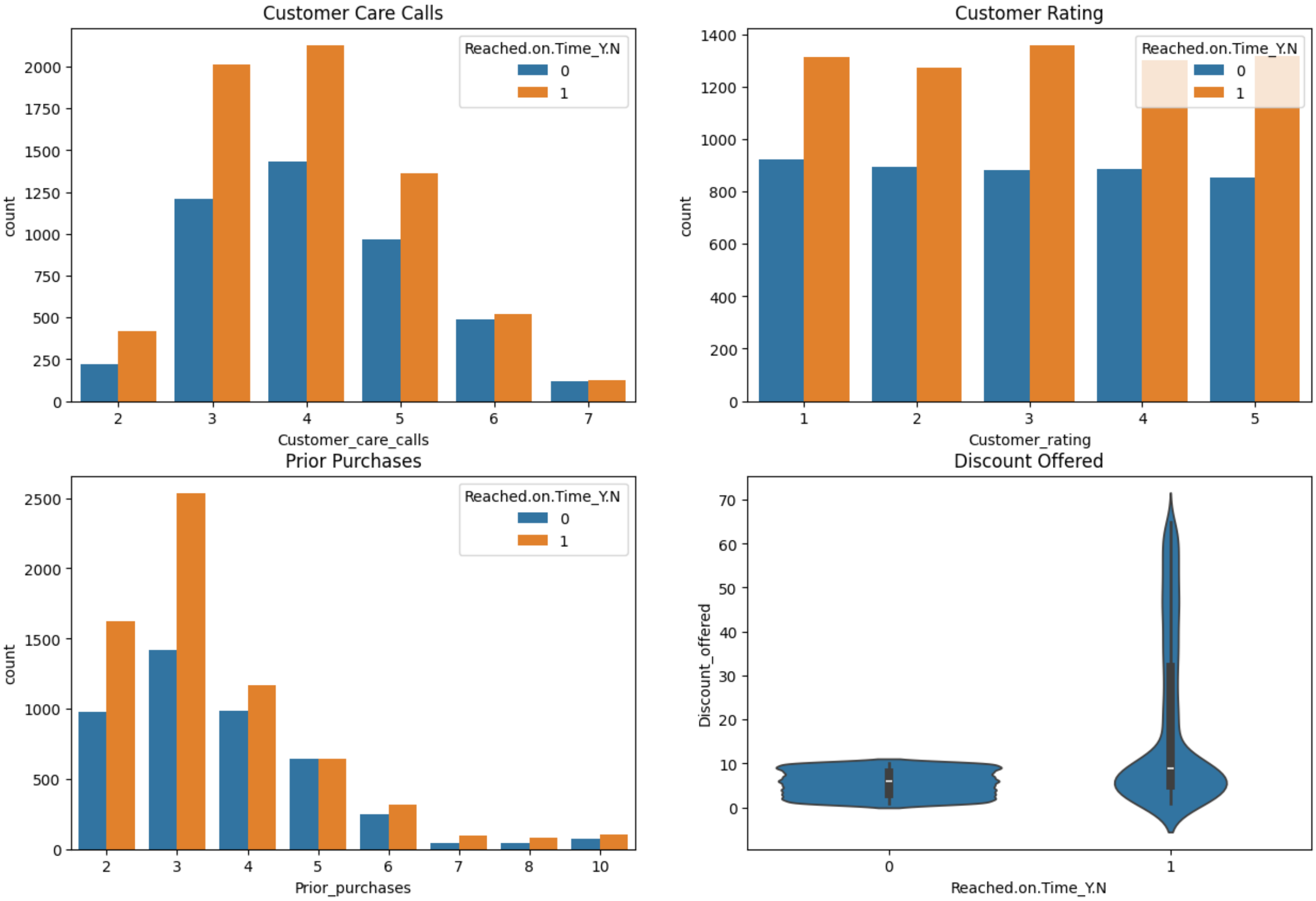
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
sns.countplot(x = 'Warehouse_block', data = df, ax=ax[0], hue = 'Reached.on.Tim
sns.countplot(x = 'Mode_of_Shipment', data = df, ax=ax[1], hue = 'Reached.on.Ti
```

```
Text(0.5, 1.0, 'Mode of Shipment')
```



```
fig, ax = plt.subplots(2,2,figsize=(15,10))
sns.countplot(x = 'Customer_care_calls', data = df, ax=ax[0,0],hue = 'Reached.o
sns.countplot(x = 'Customer_rating', data = df, ax=ax[0,1],hue = 'Reached.on.Ti
sns.countplot(x = 'Prior_purchases', data = df, ax=ax[1,0],hue = 'Reached.on.Ti
sns.violinplot(x = 'Reached.on.Time_Y.N', y = 'Discount_offered' ,data = df, ax
```

```
Text(0.5, 1.0, 'Discount Offered')
```

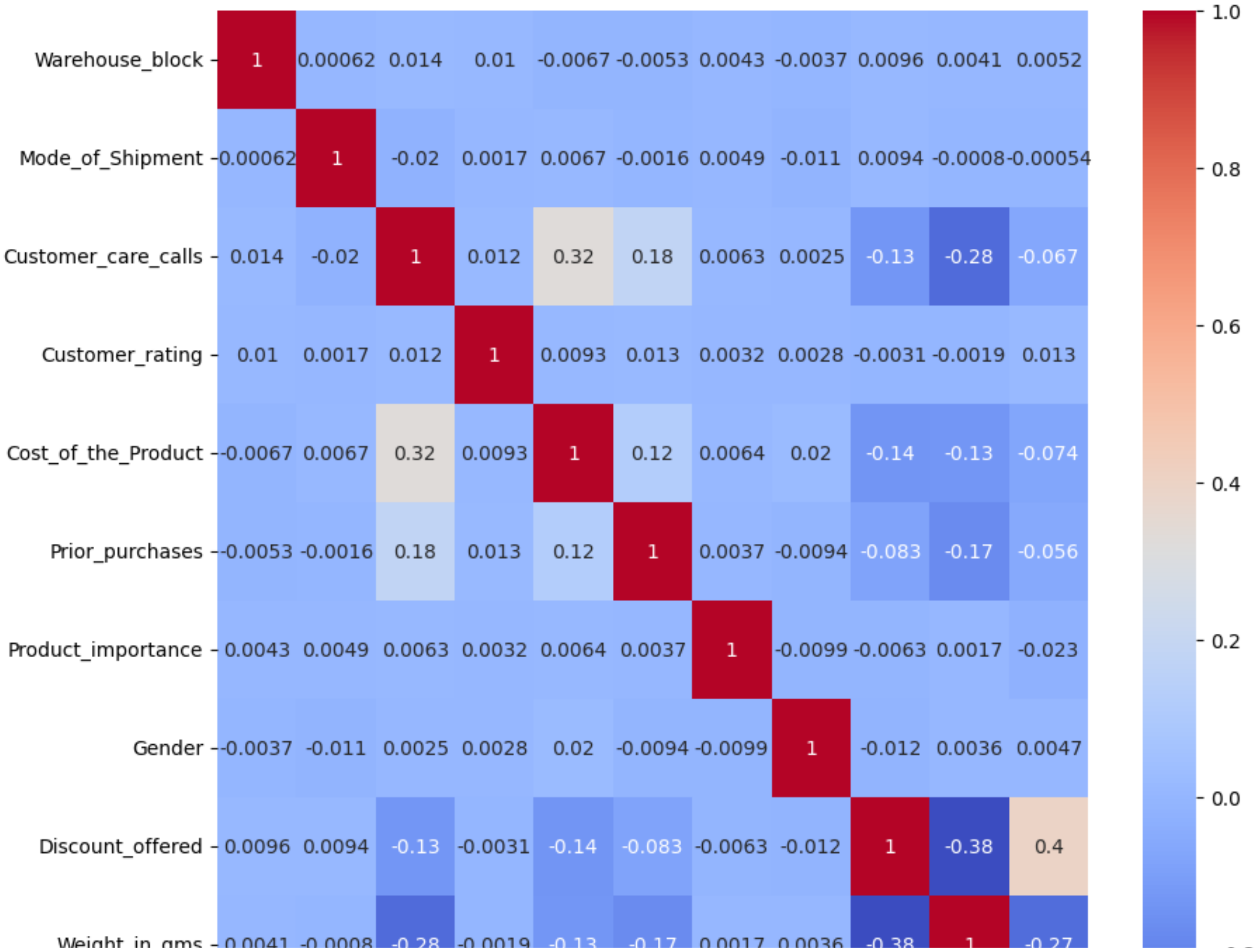


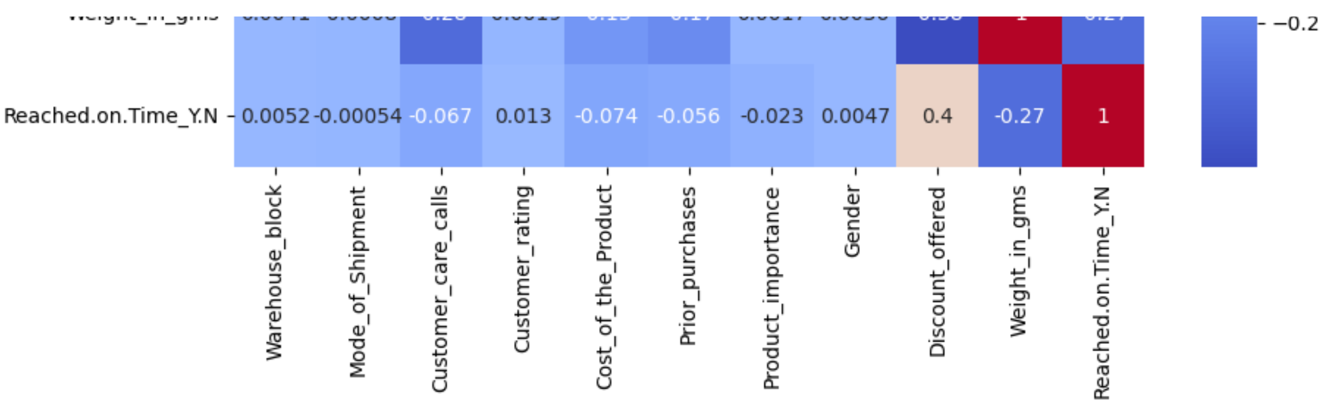
```
from sklearn.preprocessing import LabelEncoder

#Label encoding object
le = LabelEncoder()

#columns for label encoding
cols = ['Warehouse_block','Mode_of_Shipment','Product_importance', 'Gender']

#label encoding
for i in cols:
    le.fit(df[i])
    df[i] = le.transform(df[i])
    print(i, df[i].unique())
```

```
Warehouse_block [3 4 0 1 2]
Mode_of_Shipment [0 2 1]
Product_importance [1 2 0]
Gender [0 1]
```

```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

```
<Axes: >
```





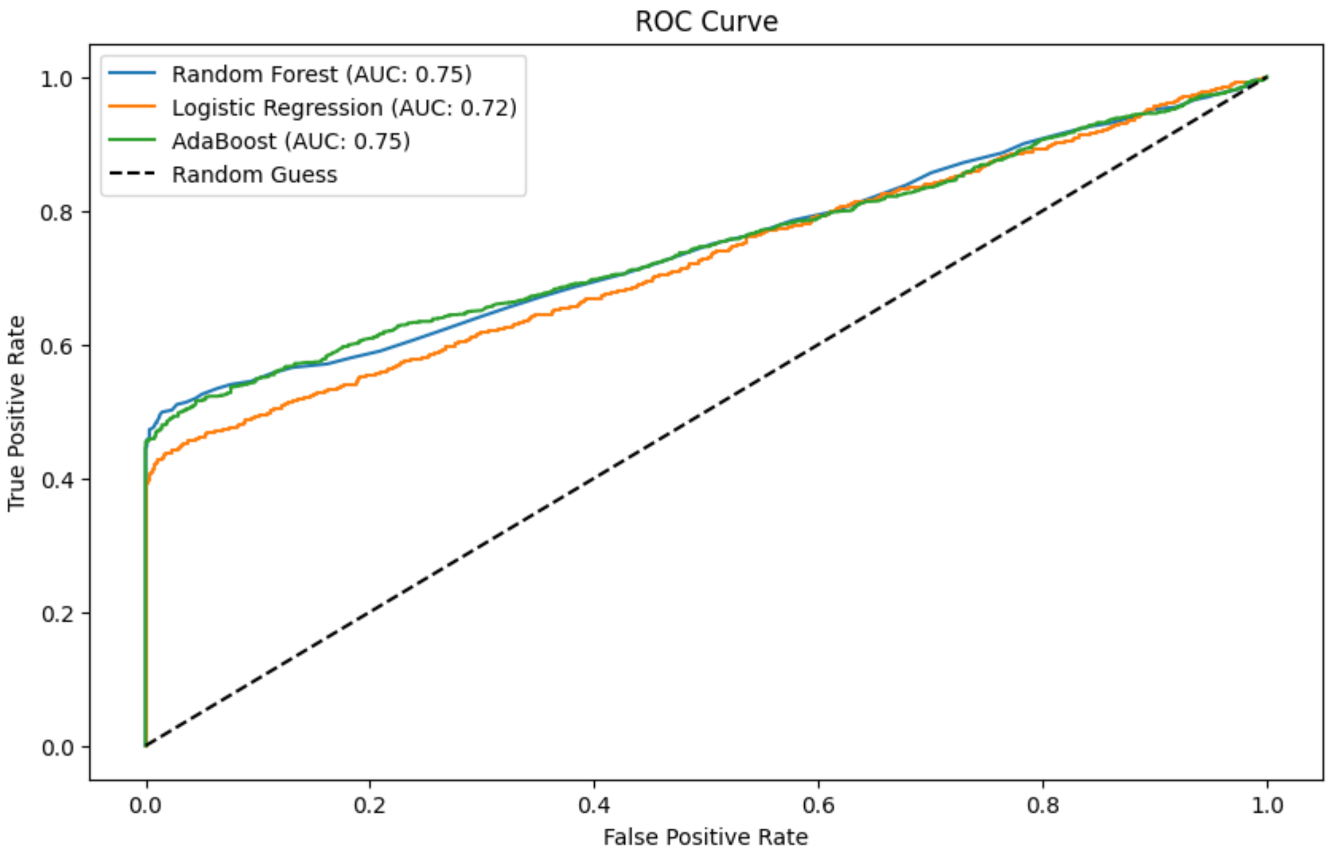
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.drop('Reached.on.Time_Y.
```

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score, roc_curve, ac
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv('Train.csv')

# Data Preprocessing
# Handle missing values (e.g., impute missing numerical columns with median)
df['Weight_in_gms'].fillna(df['Weight_in_gms'].median(), inplace=True)
```

```
# Encode categorical variables
from sklearn.preprocessing import LabelEncoder
categorical_cols = ['Warehouse_block', 'Mode_of_Shipment', 'Product_importance'
le = LabelEncoder()
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])

# Normalize numerical variables
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
numerical_cols = ['Cost_of_the_Product', 'Weight_in_gms', 'Discount_offered']
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Define target and features
X = df.drop(['Reached.on.Time_Y.N', 'ID'], axis=1)
y = df['Reached.on.Time_Y.N']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Model Training
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'AdaBoost': AdaBoostClassifier(n_estimators=100, random_state=42)
}

results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]
    auc_score = roc_auc_score(y_test, y_prob)
    results[name] = auc_score
    print(f"=== {name} Evaluation ===")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("AUC:", auc_score)
    print(classification_report(y_test, y_pred))
    print("\n")

# Plot ROC curves
plt.figure(figsize=(10, 6))
for name, model in models.items():
    y_prob = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    plt.plot(fpr, tpr, label=f"{name} (AUC: {results[name]:.2f})")
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

```
<ipython-input-23-39710984200f>:16: FutureWarning: A value is trying to be
The behavior will change in pandas 3.0. This inplace method will never work

For example, when doing 'df[col].method(value, inplace=True)', try using 'd


  df['Weight_in_gms'].fillna(df['Weight_in_gms'].median(), inplace=True)
=== Random Forest Evaluation ===
Accuracy: 0.6659090909090909
AUC: 0.7464992829469808
              precision    recall  f1-score   support

           0       0.57      0.70      0.63       895
           1       0.76      0.64      0.69      1305

    accuracy                           0.67      2200
   macro avg       0.67      0.67      0.66      2200
weighted avg       0.68      0.67      0.67      2200



=== Logistic Regression Evaluation ===
Accuracy: 0.6372727272727273
AUC: 0.7248605492412081
              precision    recall  f1-score   support

           0       0.56      0.54      0.55       895
           1       0.69      0.70      0.70      1305

    accuracy                           0.64      2200
   macro avg       0.62      0.62      0.62      2200
weighted avg       0.64      0.64      0.64      2200



/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.p
  warnings.warn(
=== AdaBoost Evaluation ===
Accuracy: 0.6859090909090909
AUC: 0.7469355080374152
              precision    recall  f1-score   support

           0       0.59      0.78      0.67       895
           1       0.81      0.62      0.70      1305

    accuracy                           0.69      2200
   macro avg       0.70      0.70      0.69      2200
```

```
weighted avg       0.72      0.69      0.69      2200
```



```
!pip install pulp
```

```
Collecting pulp
  Downloading PuLP-2.9.0-py3-none-any.whl.metadata (5.4 kB)
Downloading PuLP-2.9.0-py3-none-any.whl (17.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.7/17.7 MB 69.7 MB/s eta 0:00
Installing collected packages: pulp
Successfully installed pulp-2.9.0
```

```
# Import necessary libraries for optimization
from pulp import LpProblem, LpMinimize, LpVariable, lpSum




# Define data
shipment_modes = ['Air', 'Ship', 'Road']
costs = {'Air': 10, 'Ship': 5, 'Road': 2}
capacities = {'Air': 100, 'Ship': 300, 'Road': 500}
demand = 700  # Total demand in weight

# Create decision variables
x = LpVariable.dicts("Shipment", shipment_modes, lowBound=0)

# Define the optimization problem
prob = LpProblem("Minimize_Delivery_Costs", LpMinimize)

# Objective function
prob += lpSum([costs[mode] * x[mode] for mode in shipment_modes]), "Total Costs

# Constraints
prob += lpSum([x[mode] for mode in shipment_modes]) == demand, "Demand Constrai
for mode in shipment_modes:
    prob += x[mode] <= capacities[mode], f"Capacity Constraint for {mode}"

# Solve the problem
prob.solve()

# Display the results
print("Optimal Shipment Allocation:")
for mode in shipment_modes:
    print(f"{mode}: {x[mode].varValue} units")
print("Total Cost:", prob.objective.value())

# Sensitivity analysis
print("\nSensitivity Analysis:")
for mode in shipment_modes:
    costs[mode] += 1  # Simulate cost increase
    prob.solve()
    print(f"Updated Cost for {mode}: {prob.objective.value()}")
    costs[mode] -= 1  # Reset cost
```

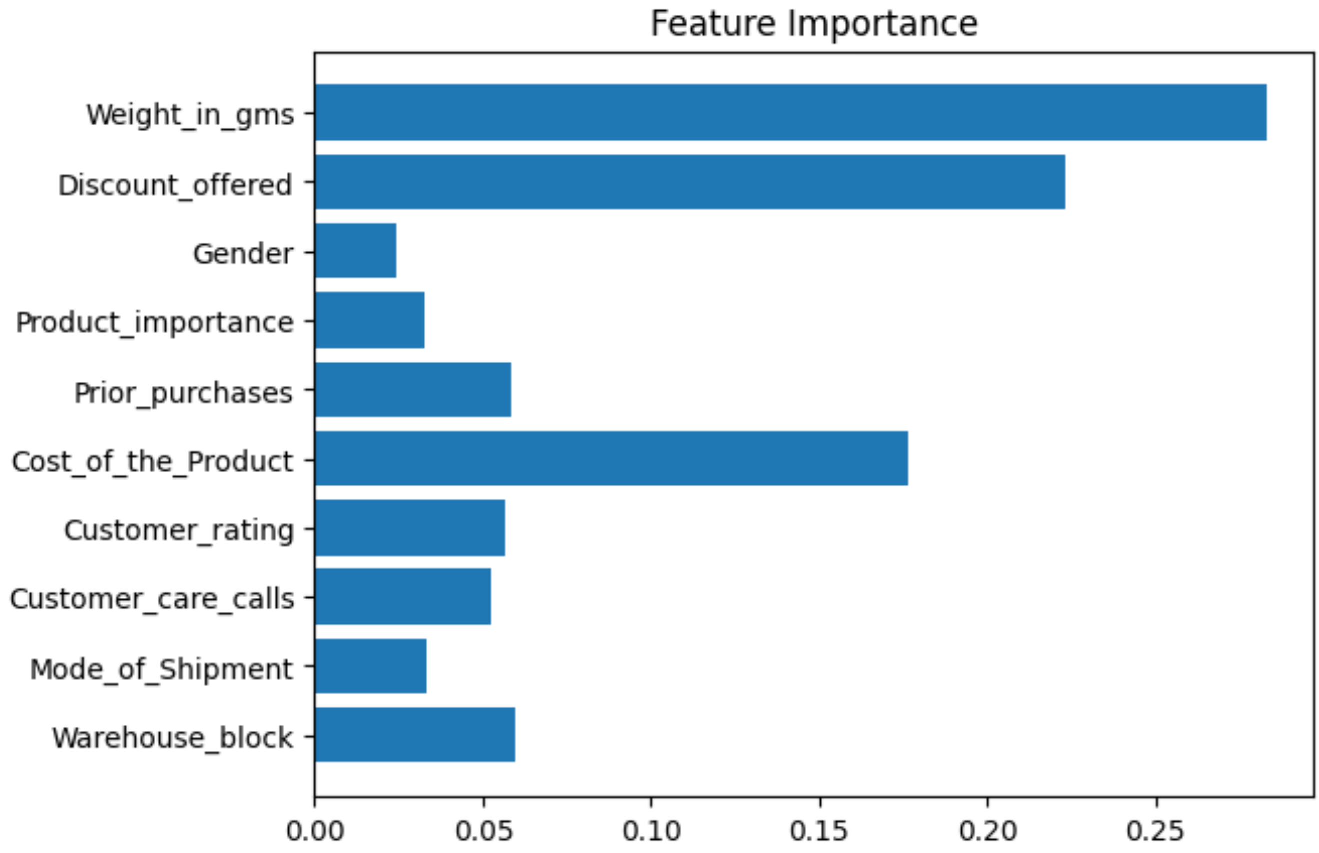
```
Optimal Shipment Allocation:
Air: 0.0 units
Ship: 200.0 units
Road: 500.0 units
Total Cost: 2000.0

Sensitivity Analysis:
Updated Cost for Air: 2000.0
Updated Cost for Ship: 2000.0
Updated Cost for Road: 2000.0
```

Start coding or generate with AI.

```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score

# Train Models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(n_estimators=100),
    'AdaBoost': AdaBoostClassifier(n_estimators=100)
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"{name} - AUC: {roc_auc_score(y_test, model.predict_proba(X_test)[:,
    print(classification_report(y_test, y_pred))
```

```
Logistic Regression - AUC: 0.7248605492412081
              precision    recall  f1-score   support

           0       0.56      0.54      0.55       895
           1       0.69      0.70      0.70      1305

    accuracy                           0.64      2200
   macro avg       0.62      0.62      0.62      2200
weighted avg       0.64      0.64      0.64      2200

Random Forest - AUC: 0.7525400800530833
              precision    recall  f1-score   support

           0       0.58      0.72      0.64       895
           1       0.77      0.64      0.70      1305

    accuracy                           0.67      2200
   macro avg       0.68      0.68      0.67      2200
weighted avg       0.69      0.67      0.68      2200

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.p
  warnings.warn(
AdaBoost - AUC: 0.7469355080374152
              precision    recall  f1-score   support

           0       0.59      0.78      0.67       895
           1       0.81      0.62      0.70      1305

    accuracy                           0.69      2200
   macro avg       0.70      0.70      0.69      2200
weighted avg       0.72      0.69      0.69      2200
```

```
import matplotlib.pyplot as plt
feature_importance = models['Random Forest'].feature_importances_
plt.barh(X.columns, feature_importance)
plt.title('Feature Importance')
plt.show()
```



```
#PHASE-III prescriptive
```

```
# Import necessary libraries for optimization
from pulp import LpProblem, LpMinimize, LpVariable, lpSum

# Define data
shipment_modes = ['Air', 'Ship', 'Road']
costs = {'Air': 10, 'Ship': 5, 'Road': 2}
capacities = {'Air': 100, 'Ship': 300, 'Road': 500}
demand = 700  # Total demand in weight

# Create decision variables
x = LpVariable.dicts("Shipment", shipment_modes, lowBound=0)

# Define the optimization problem
prob = LpProblem("Minimize Delivery Costs", LpMinimize)
prob = LpProblem("Minimize_Delivery_Costs", LpMinimize)

# Objective function
prob += lpSum([costs[mode] * x[mode] for mode in shipment_modes]), "Total Costs"

# Constraints
prob += lpSum([x[mode] for mode in shipment_modes]) == demand, "Demand Constrain
for mode in shipment_modes:
    prob += x[mode] <= capacities[mode], f"Capacity Constraint for {mode}"

# Solve the problem
prob.solve()

# Display the results
print("Optimal Shipment Allocation:")
for mode in shipment_modes:
    print(f"{mode}: {x[mode].varValue} units")
print("Total Cost:", prob.objective.value())

# Sensitivity analysis
print("\nSensitivity Analysis:")
for mode in shipment_modes:
    costs[mode] += 1  # Simulate cost increase
    prob.solve()
    print(f"Updated Cost for {mode}: {prob.objective.value()}")
    costs[mode] -= 1  # Reset cost
```

```
Optimal Shipment Allocation:
Air: 0.0 units
Ship: 200.0 units
Road: 500.0 units
Total Cost: 2000.0

Sensitivity Analysis:
Updated Cost for Air: 2000.0
Updated Cost for Ship: 2000.0
Updated Cost for Road: 2000.0
```

Start coding or generate with AI.