E-COMMERCE DATA

ID: ID Number of Customers.

Warehouse block: The Company have big Warehouse which is divided in to block such as A,B,C,D,E.

Mode of shipment:The Company Ships the products in multiple way such as Ship, Flight and Road.

Customer care calls: The number of calls made from enquiry for enquiry of the shipment.

Customer rating: The company has rated from every customer. 1 is the lowest (Worst), 5 is the highest (Best).

Cost of the product: Cost of the Product in US Dollars.

Prior purchases: The Number of Prior Purchase.

Product importance: The company has categorized the product in the various parameter such as low, medium, high.

Gender: Male and Female.

Discount offered: Discount offered on that specific product.

Weight in gms: It is the weight in grams.

Reached on time: It is the target variable, where 1 Indicates that the product has NOT reached on time and 0 indicates it has reached on time.

Standerd imports

In [1]:
```
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
```

```
import warnings
from pylab import rcParams
rcParams["figure.figsize"]=(20,8)
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv(r"E:\PORTFOLIO-PROJECTS\CAREERERA - INTERNSHIP DATA\E-Commerce\E_Commerce.csv")
df.head()
```

Out[2]:

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importance
0	1	D	Flight	4	2	177	3	low
1	2	F	Flight	4	5	216	2	low
2	3	A	Flight	2	2	183	4	low
3	4	B	Flight	3	3	176	4	medium
4	5	C	Flight	2	2	184	3	medium

In [3]:
```
df.describe()
```

Out[3]:

	ID	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Discount_offered	Weight_in_gms	Reached.on
count	10999.00000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	109
mean	5500.00000	4.054459	2.990545	210.196836	3.567597	13.373216	3634.016729	
std	3175.28214	1.141490	1.413603	48.063272	1.522860	16.205527	1635.377251	
min	1.00000	2.000000	1.000000	96.000000	2.000000	1.000000	1001.000000	
25%	2750.50000	3.000000	2.000000	169.000000	3.000000	4.000000	1839.500000	
50%	5500.00000	4.000000	3.000000	214.000000	3.000000	7.000000	4149.000000	
75%	8249.50000	5.000000	4.000000	251.000000	4.000000	10.000000	5050.000000	
max	10999.00000	7.000000	5.000000	310.000000	10.000000	65.000000	7846.000000	

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10999 entries, 0 to 10998
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10999 non-null  int64
 1   Warehouse_block      10999 non-null  object
 2   Mode_of_Shipment     10999 non-null  object
 3   Customer_care_calls  10999 non-null  int64
 4   Customer_rating      10999 non-null  int64
 5   Cost_of_the_Product  10999 non-null  int64
 6   Prior_purchases      10999 non-null  int64
 7   Product_importance   10999 non-null  object
 8   Gender               10999 non-null  object
 9   Discount_offered     10999 non-null  int64
 10  Weight_in_gms        10999 non-null  int64
 11  Reached.on.Time_Y.N  10999 non-null  int64
dtypes: int64(8), object(4)
memory usage: 1.0+ MB
```

In [5]:
```
df.isna().sum()
```

Out[5]:
```
ID                     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
```

Data Preparing

In [6]:
```
df=df.drop("ID",axis=1 )
```

In [7]:
```
Warehouse_block = df["Warehouse_block"].value_counts()
Warehouse_block
```

Out[7]:
```
F    3666
D    1834
A    1833
B    1833
C    1833
Name: Warehouse_block, dtype: int64
```

In [8]:
```
df["Reached.on.Time_Y.N"].value_counts()
```

Out[8]:
```
1    6563
0    4436
Name: Reached.on.Time_Y.N, dtype: int64
```

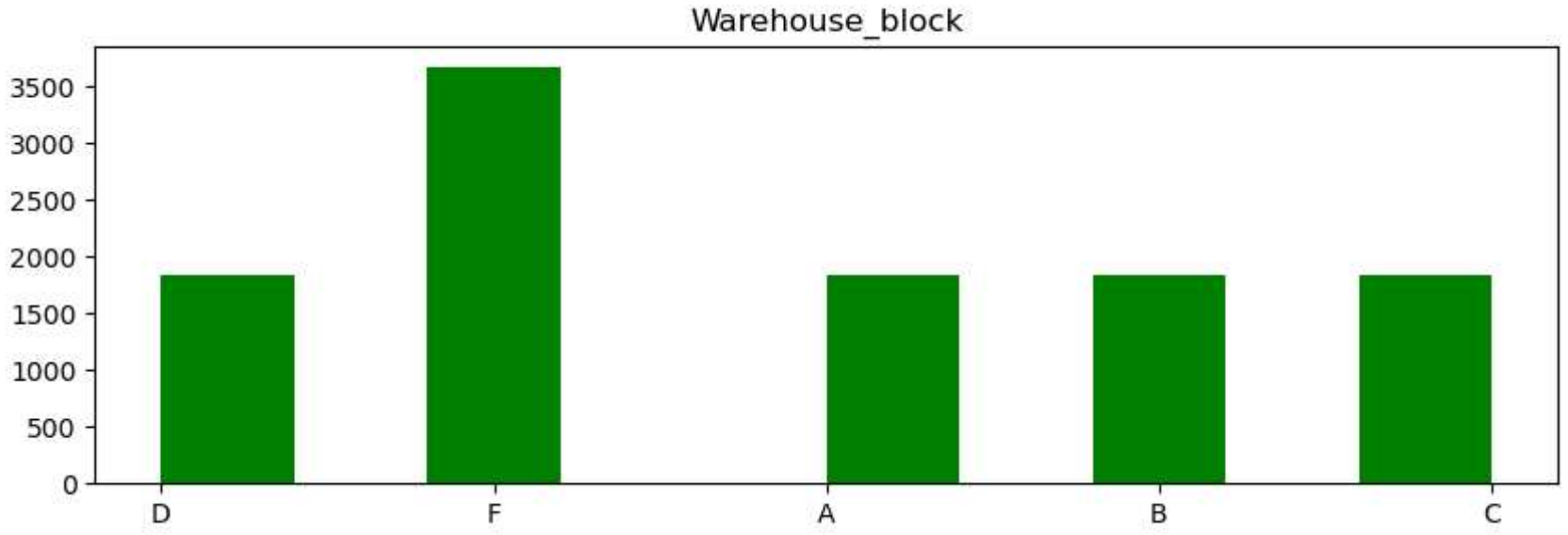
In [9]:
```
df.columns
```

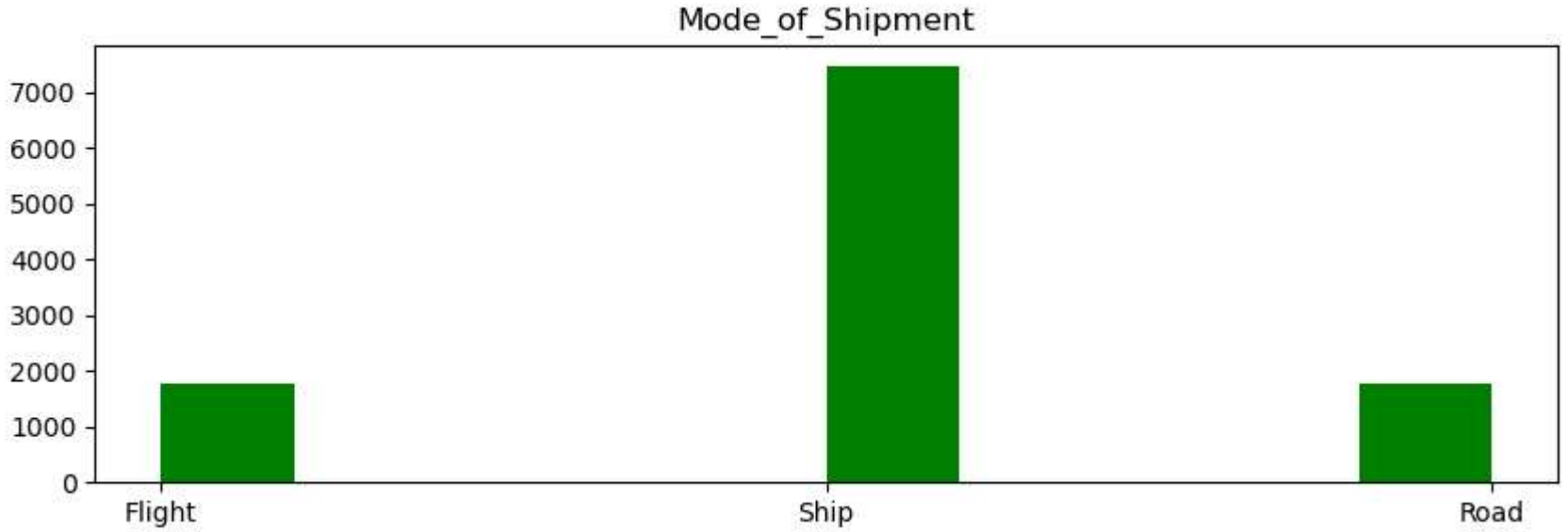
Out[9]:
```
Index(['Warehouse_block', 'Mode_of_Shipment', 'Customer_care_calls',
       'Customer_rating', 'Cost_of_the_Product', 'Prior_purchases',
       'Product_importance', 'Gender', 'Discount_offered', 'Weight_in_gms',
       'Reached.on.Time_Y.N'],
      dtype='object')
```

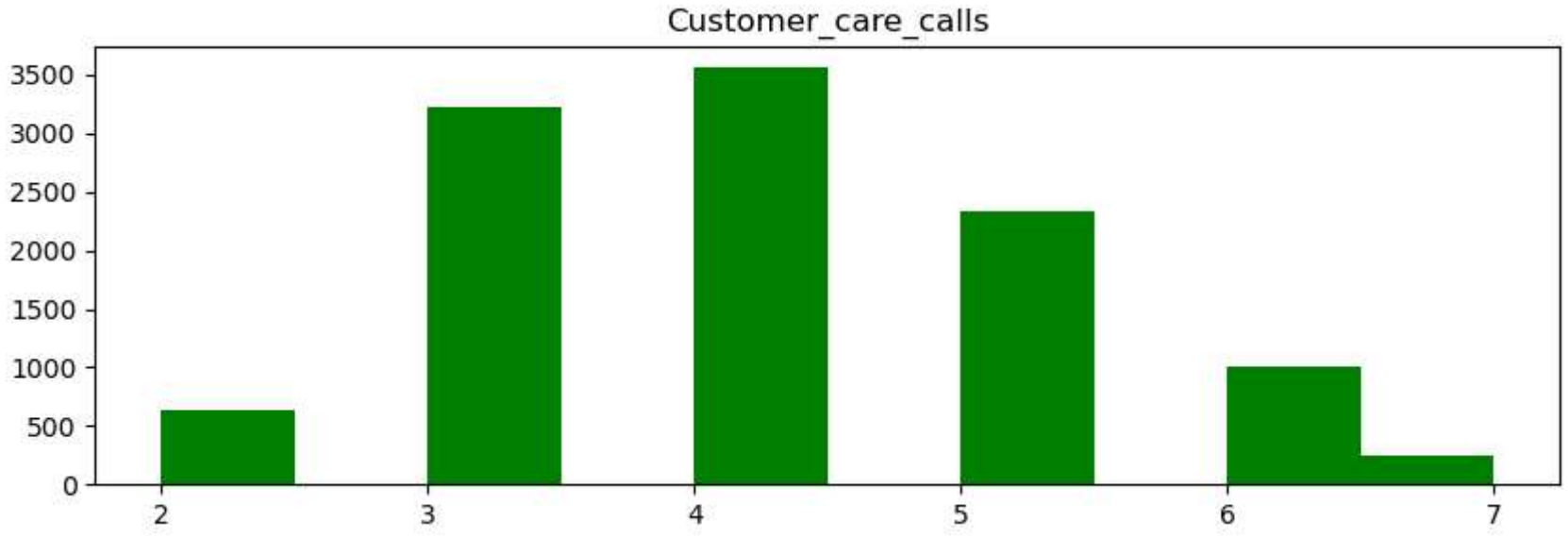
In [10]:
```
catogary_1= ['Warehouse_block', 'Mode_of_Shipment','Customer_rating','Prior_purchases',
       'Product_importance', 'Gender','Reached.on.Time_Y.N']
catogary_2 =['Discount_offered', 'Weight_in_gms']
catogary_3 =['Discount_offered', 'Weight_in_gms','Cost_of_the_Product','Reached.on.Time_Y.N']
```

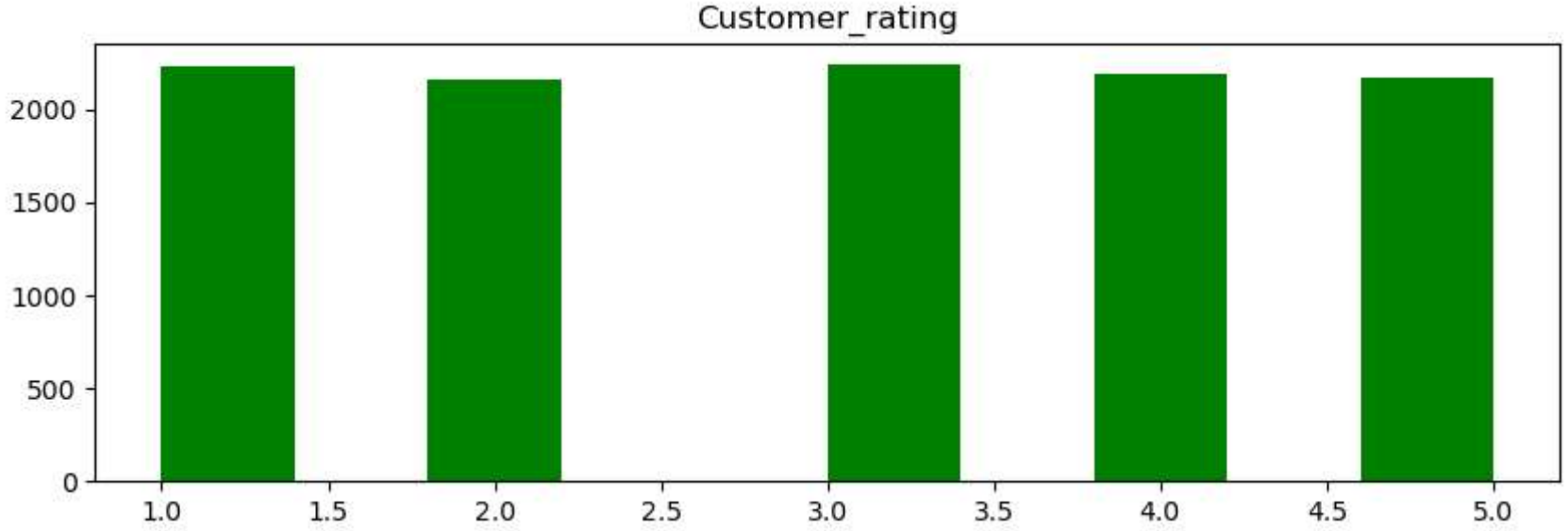
Data visualization

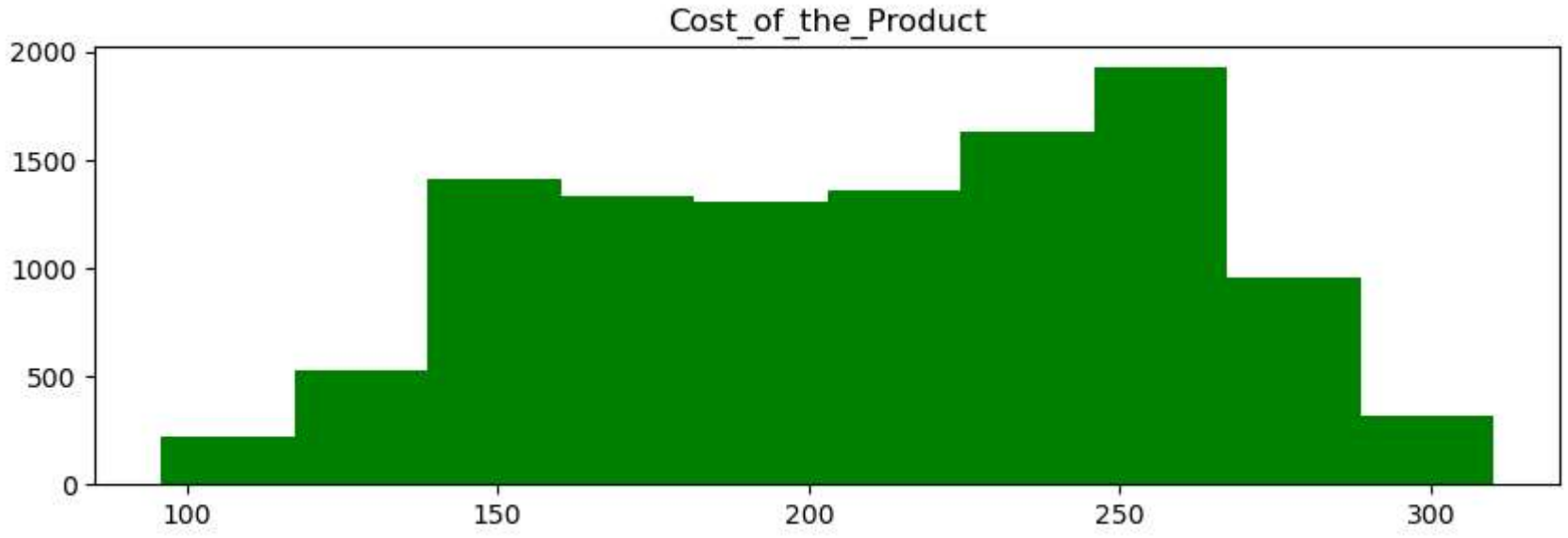
In [13]:
```
for i in df.columns :
    plt.figure(figsize=(10, 3))
    plt.title(i)
    plt.hist(df[i],color="green")
    plt.show()
```

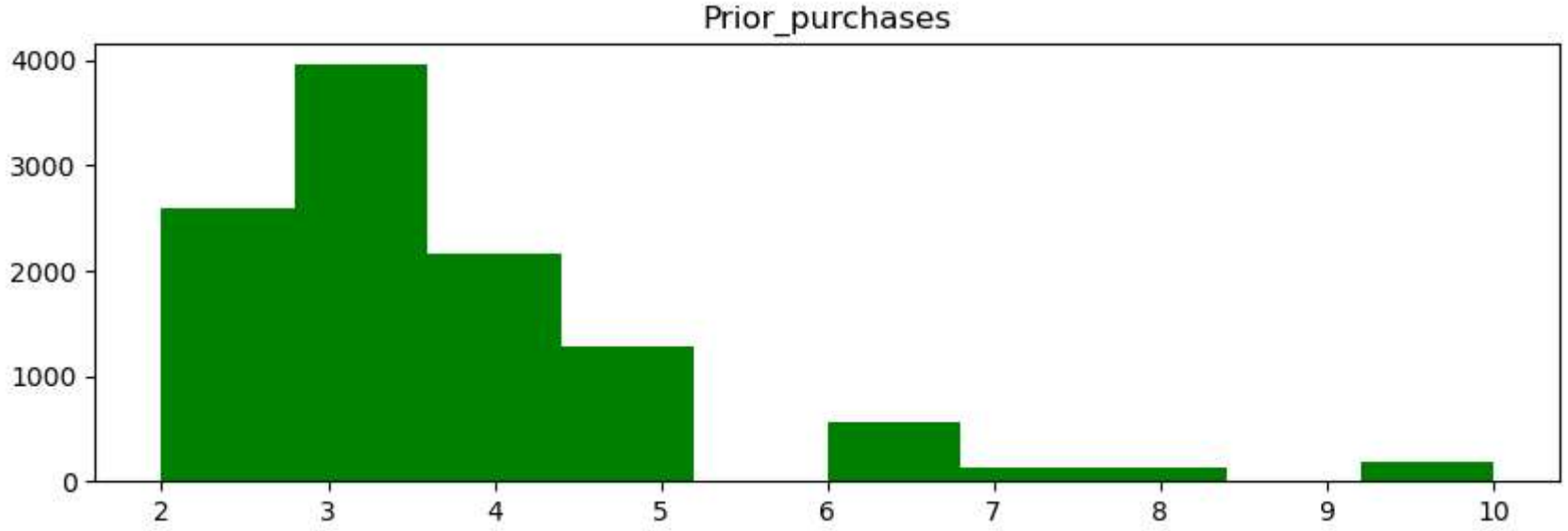

Warehouse_block
3500
3000
2500
2000
1500
1000
500
0
D
F
A
B
C

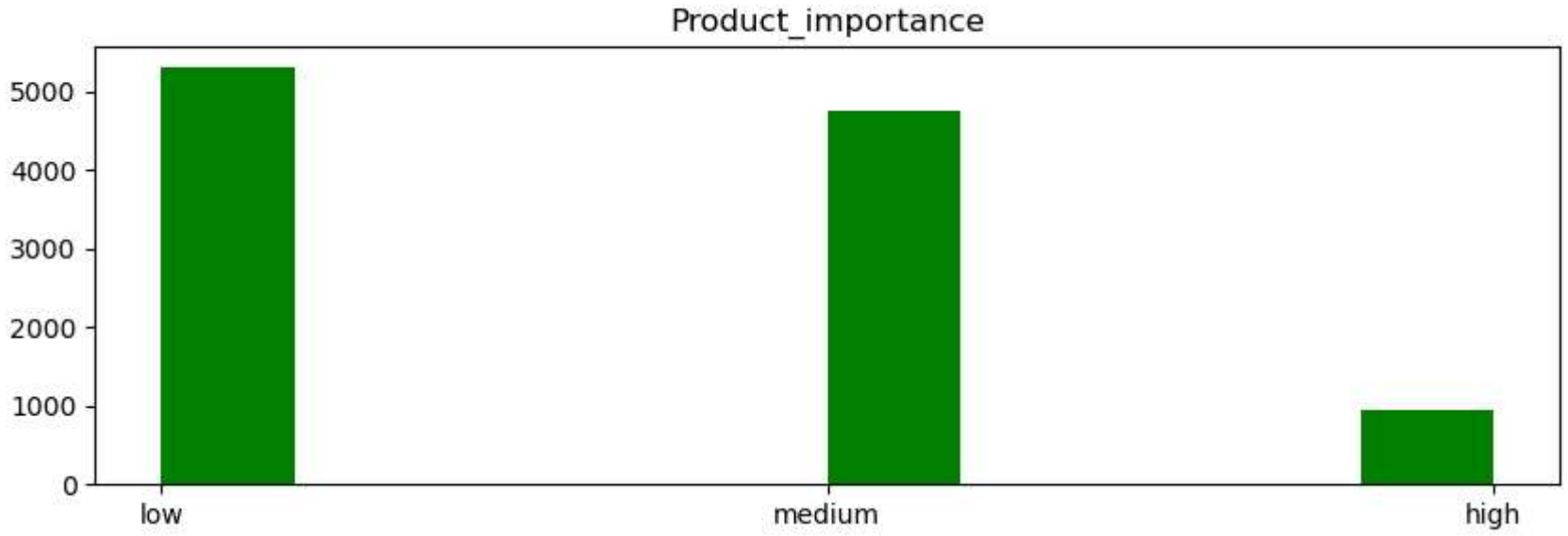

Mode_of_Shipment
7000
6000
5000
4000
3000
2000
1000
0
Flight
Ship
Road


Customer_care_calls
3500
3000
2500
2000
1500
1000
500
0
2
3
4
5
6
7

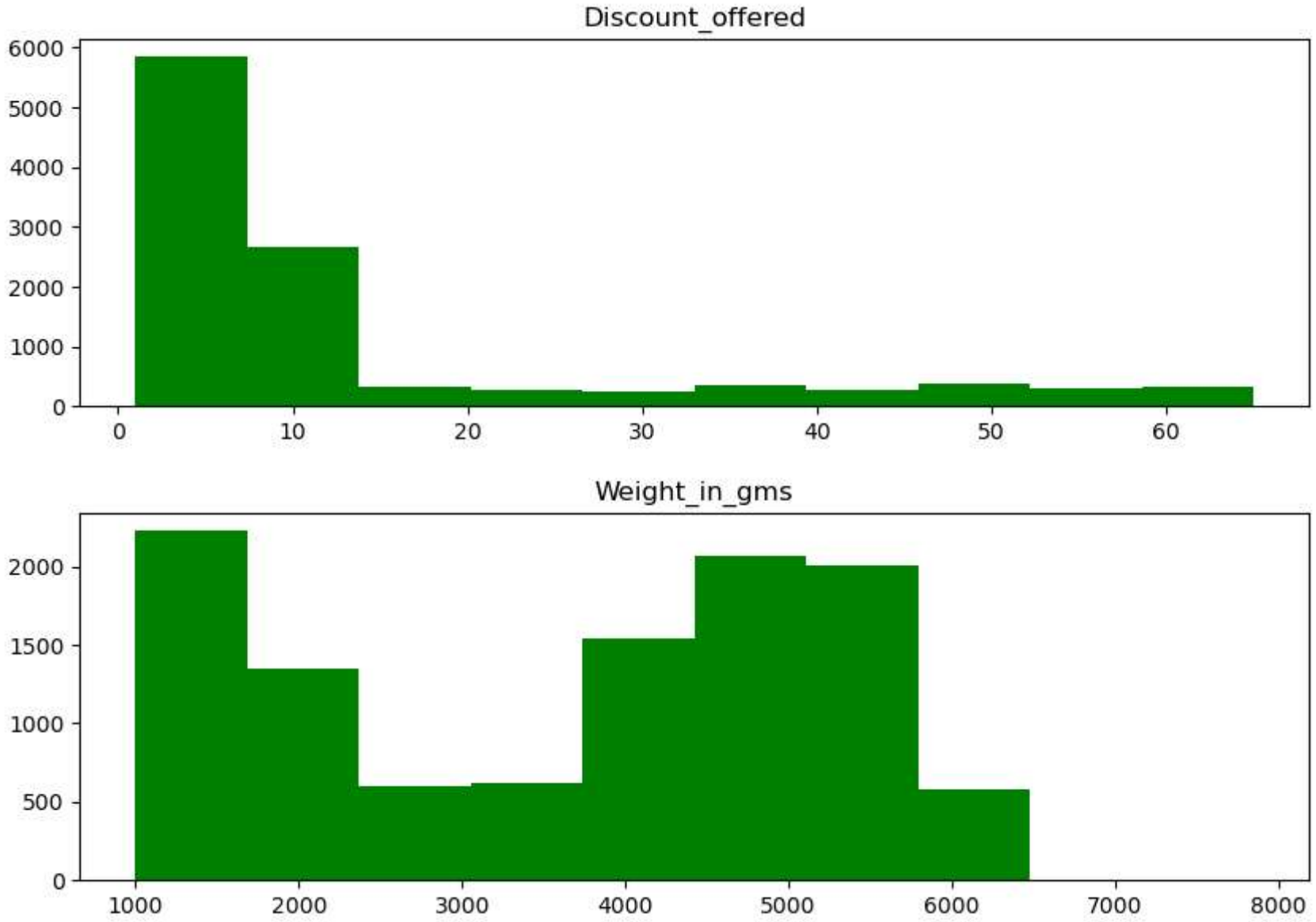

Customer_rating
2000
1500
1000
500
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

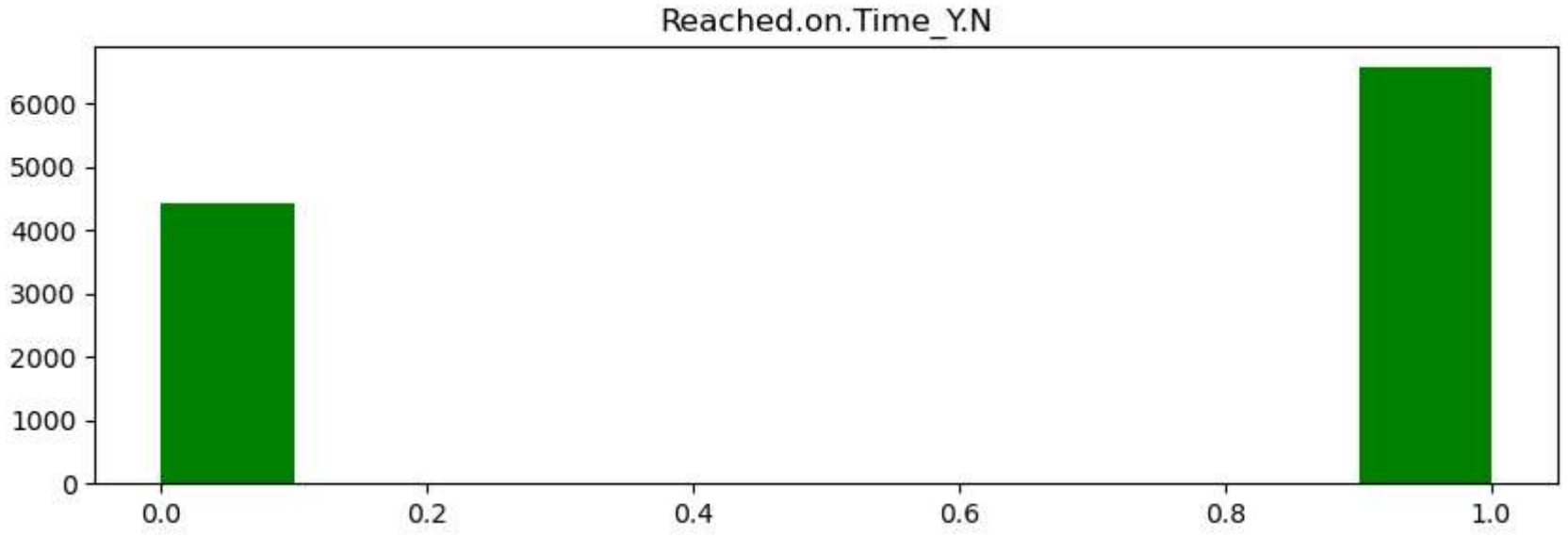

Cost_of_the_Product
2000
1500
1000
500
0
100
150
200
250
300


Prior_purchases
4000
3000
2000
1000
0
2
3
4
5
6
7
8
9
10

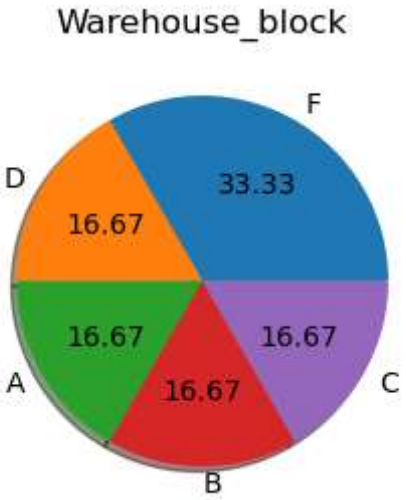

Product_importance
5000
4000
3000
2000
1000
0
low
medium
high

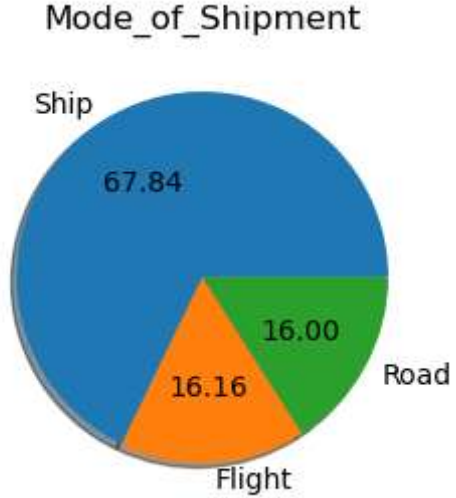

Gender
5000
4000
3000
2000
1000
0
F
M

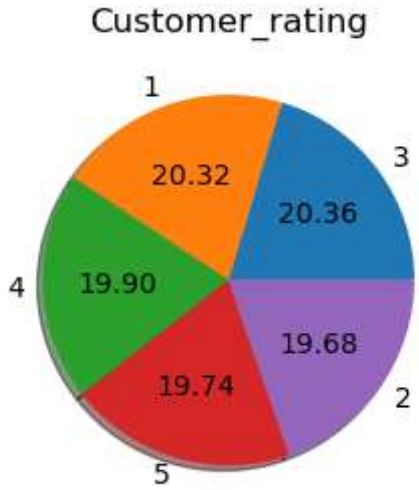

Discount_offered
6000
5000
4000
3000
2000
1000
0
0
10
20
30
40
50
60
Weight_in_gms
2000
1500
1000
500
0
1000
2000
3000
4000
5000
6000
7000
8000

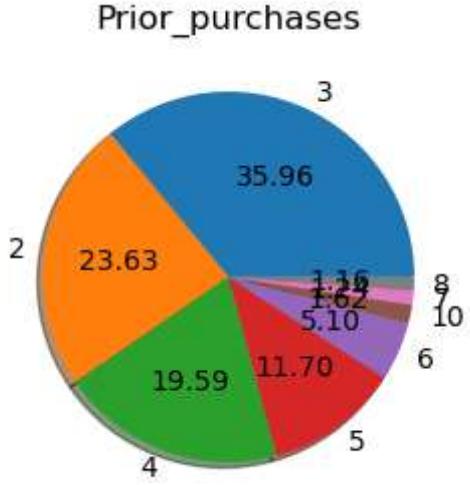


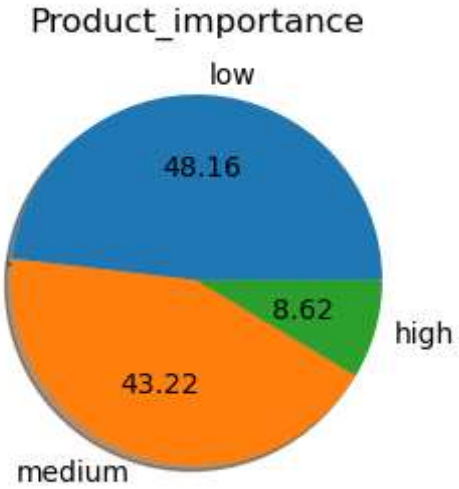
In [15]:
```
for i in catogary_1 :
    x=df[i].value_counts()
    y= x.index
    plt.figure(figsize=(8, 3))
    plt.title(i)
    plt.pie(x,labels=y
       ,shadow=True
       ,autopct="%.2F")
    plt.show()
```

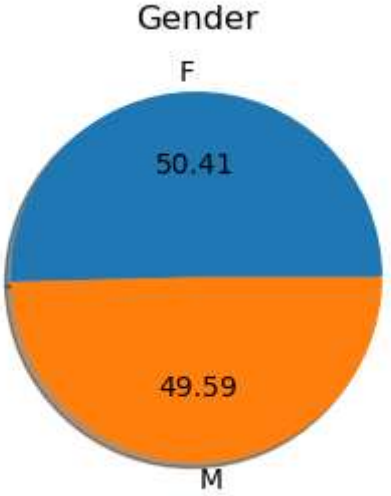

Warehouse_block
F
D
33.33
16.67
16.67
16.67
A
16.67
C
B

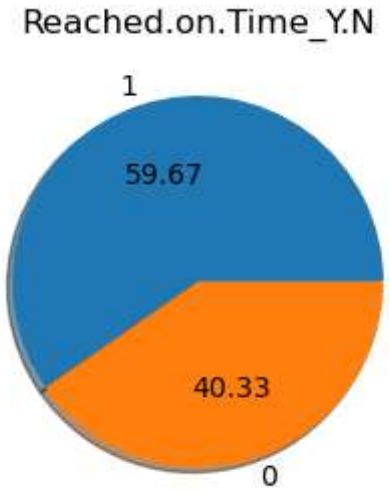

Mode_of_Shipment
Ship
67.84
16.00
Road
16.16
Flight


Customer_rating
1
20.32
3
20.36
4
19.90
19.68
19.74
2
5

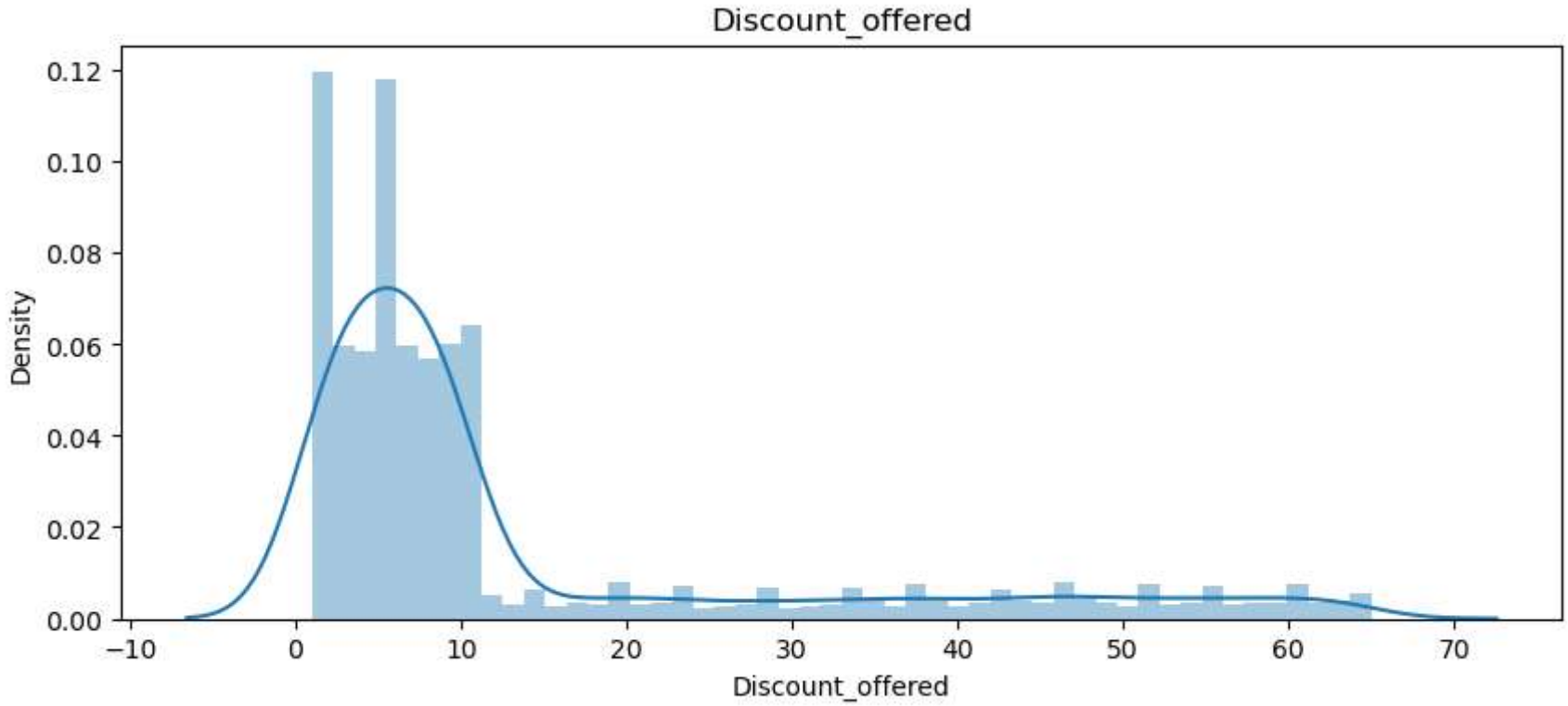

Prior_purchases
3
35.96
2
23.63
1.62
5.10
10
6
19.59
11.70
5
4

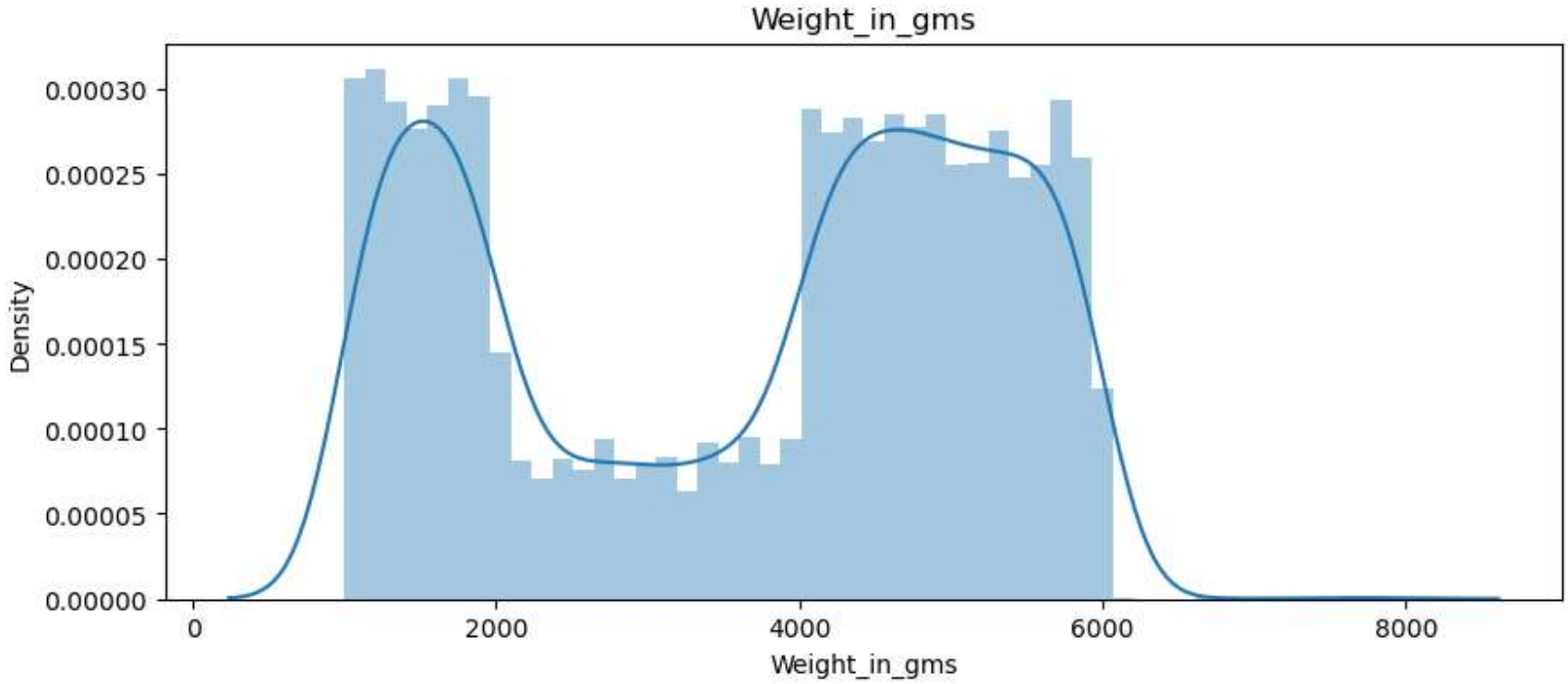

Product_importance
low
48.16
8.62
high
43.22
medium


Gender
F
50.41
49.59
M

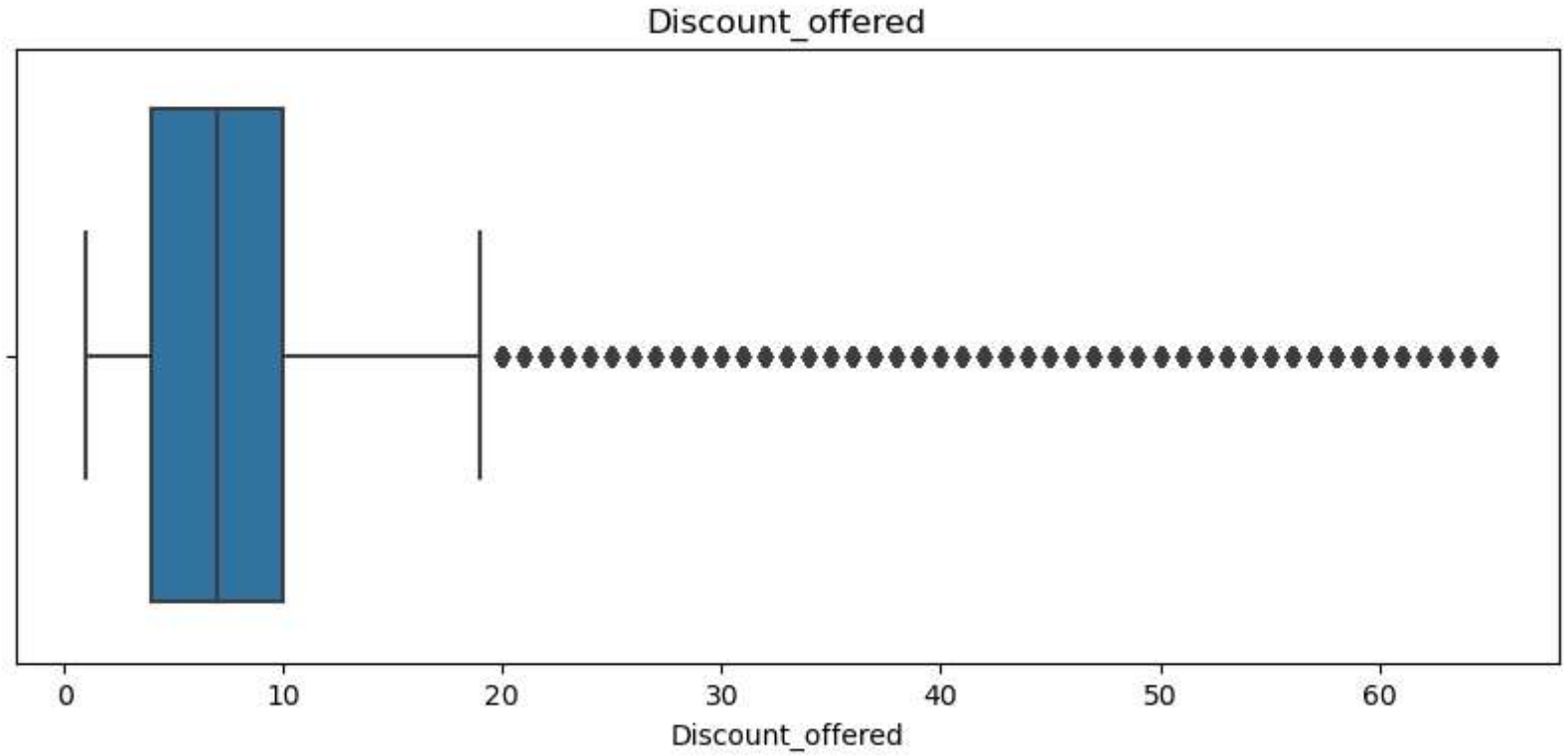


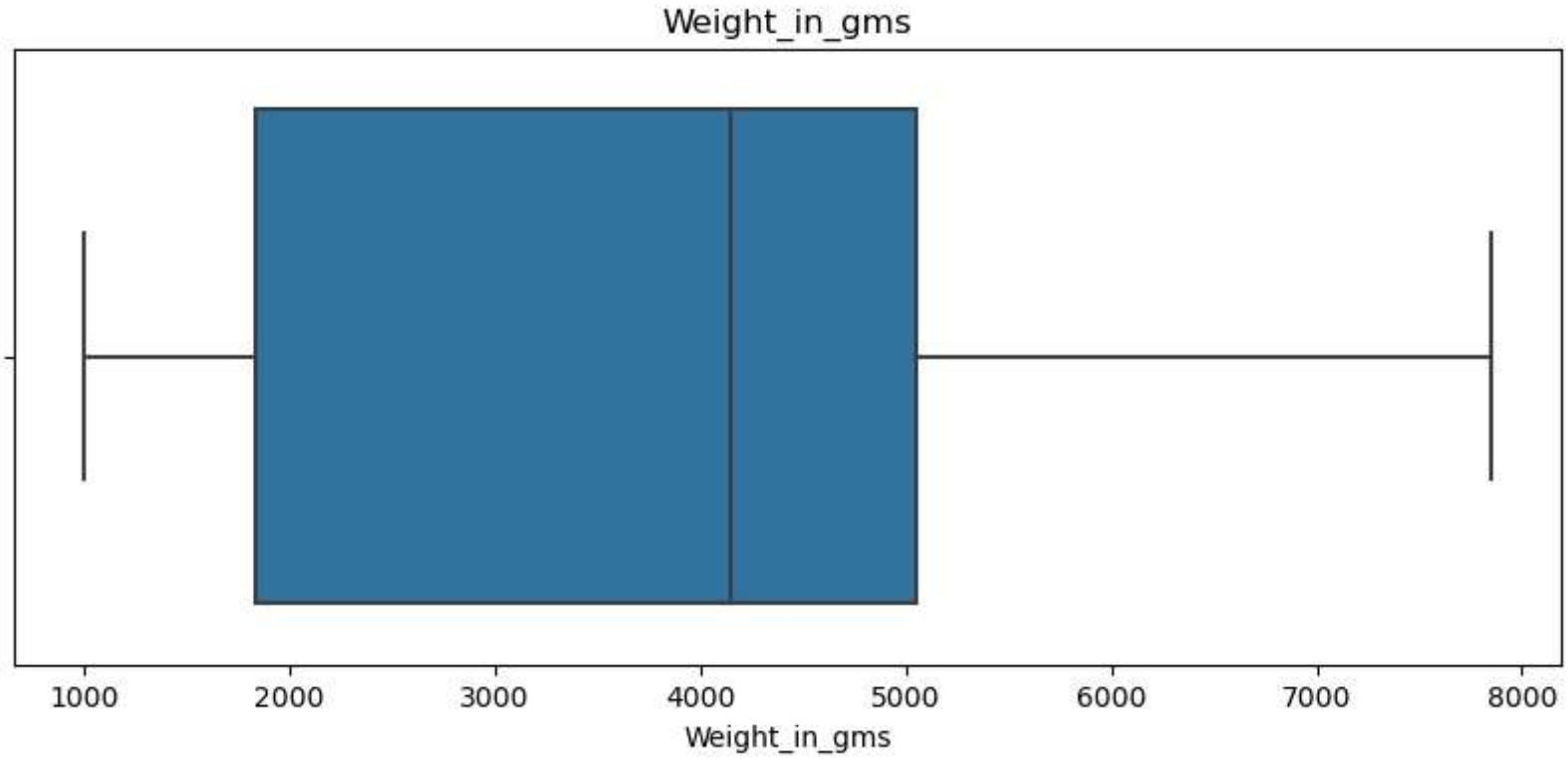
In [16]:
```
for i in catogary_2:
    plt.figure(figsize=(10, 4))
    plt.title(i)
    x=sns.distplot(df[i],kde=True, bins = 50)
    plt.show()
```


Discount_offered
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
−10
0
10
20
30
40
50
60
70
Discount_offered

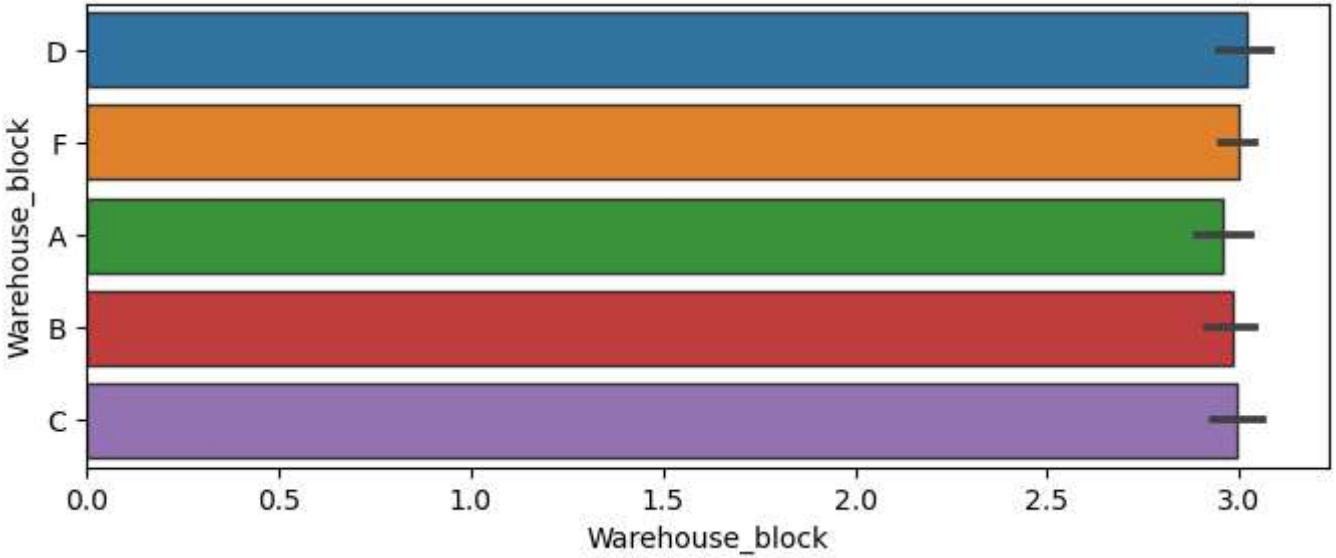


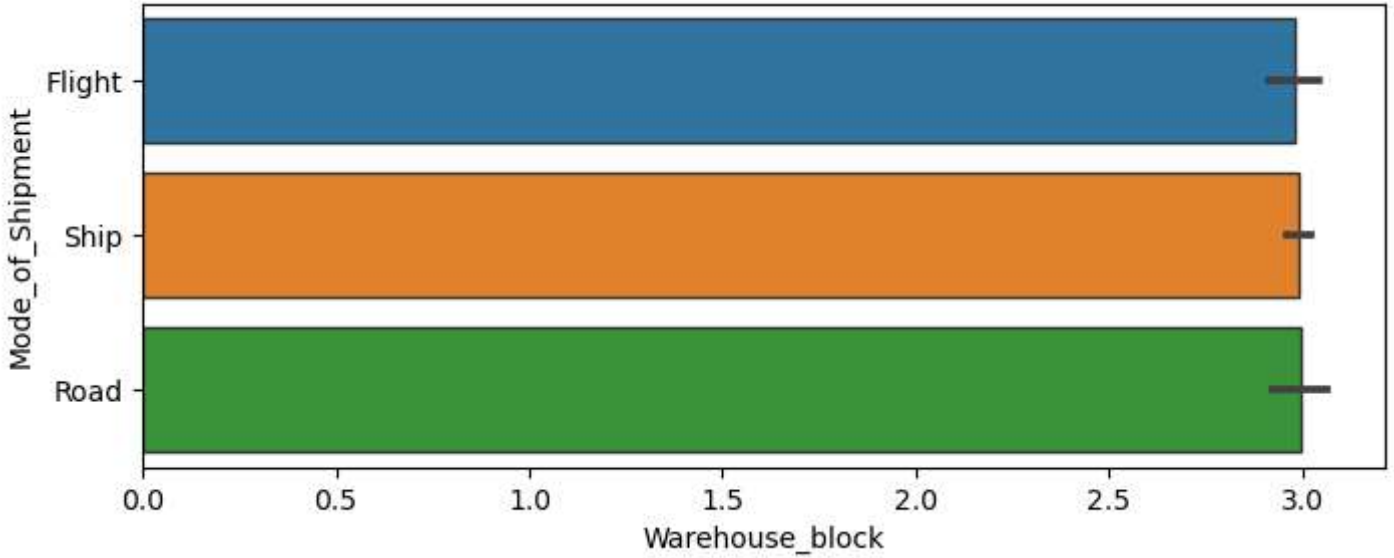
In [17]:
```
for i in catogary_2:
    plt.figure(figsize=(10, 4))
    plt.title(i)
    sns.boxplot(df[i])
    plt.show()
```

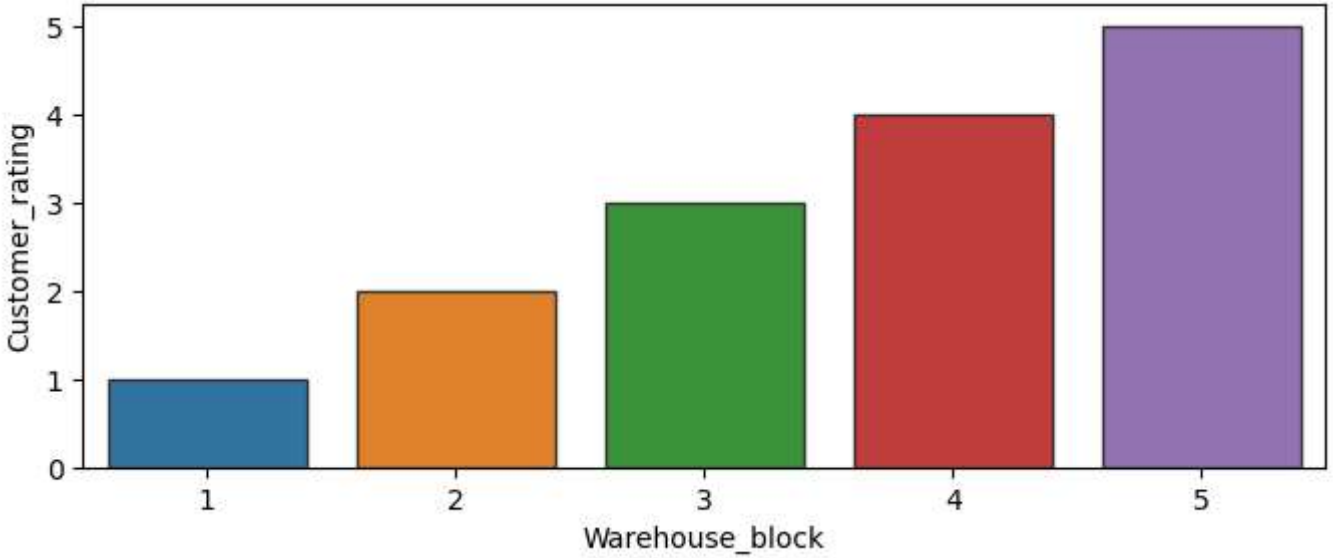

Discount_offered
0
10
20
30
40
50
60
Discount_offered

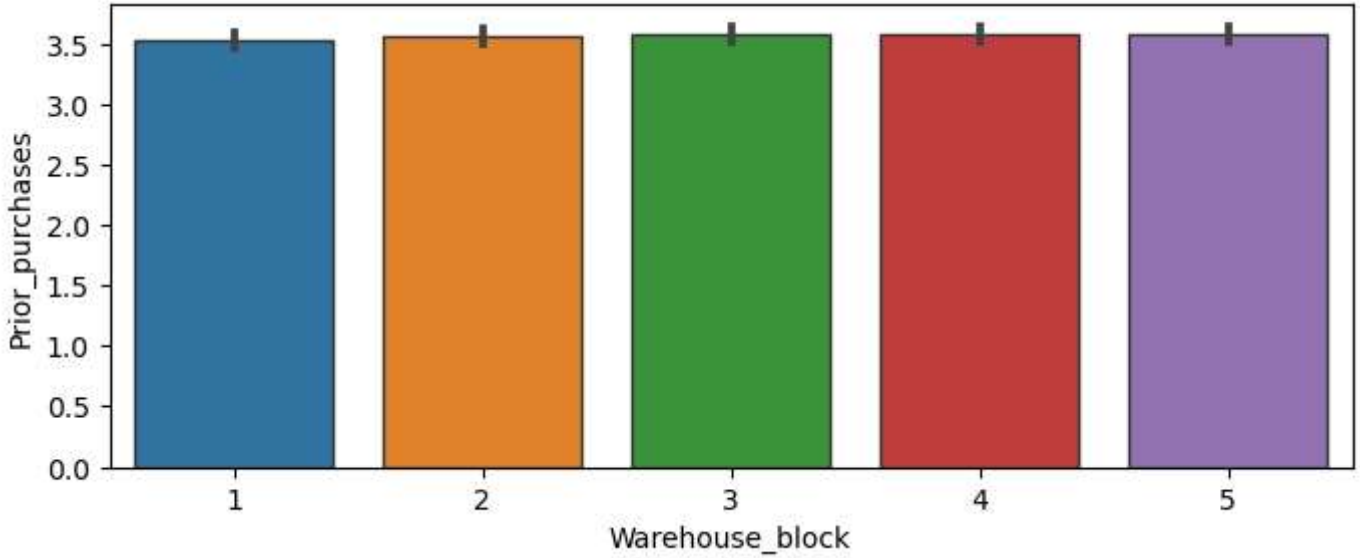


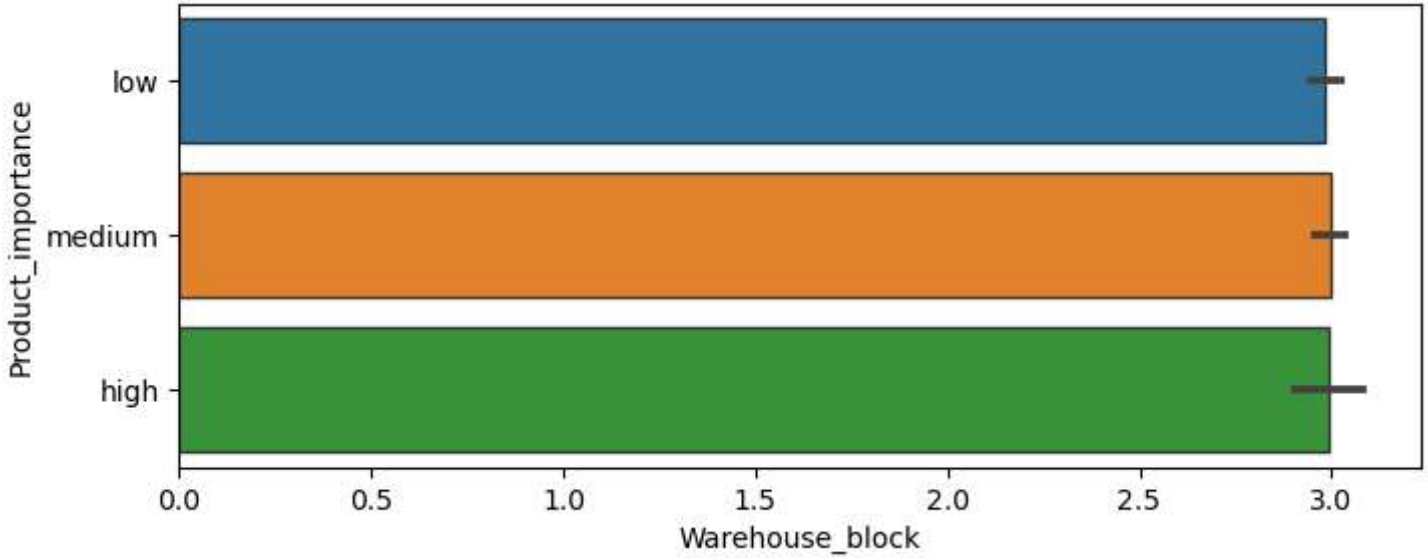
In [22]:
```
for i in catogary_1:
    plt.figure(figsize=(8, 3))
    sns.barplot(x=df["Customer_rating"] ,y= df[i], edgecolor='.2')
    plt.xlabel("Warehouse_block")
    plt.ylabel(i)
    plt.show()
```

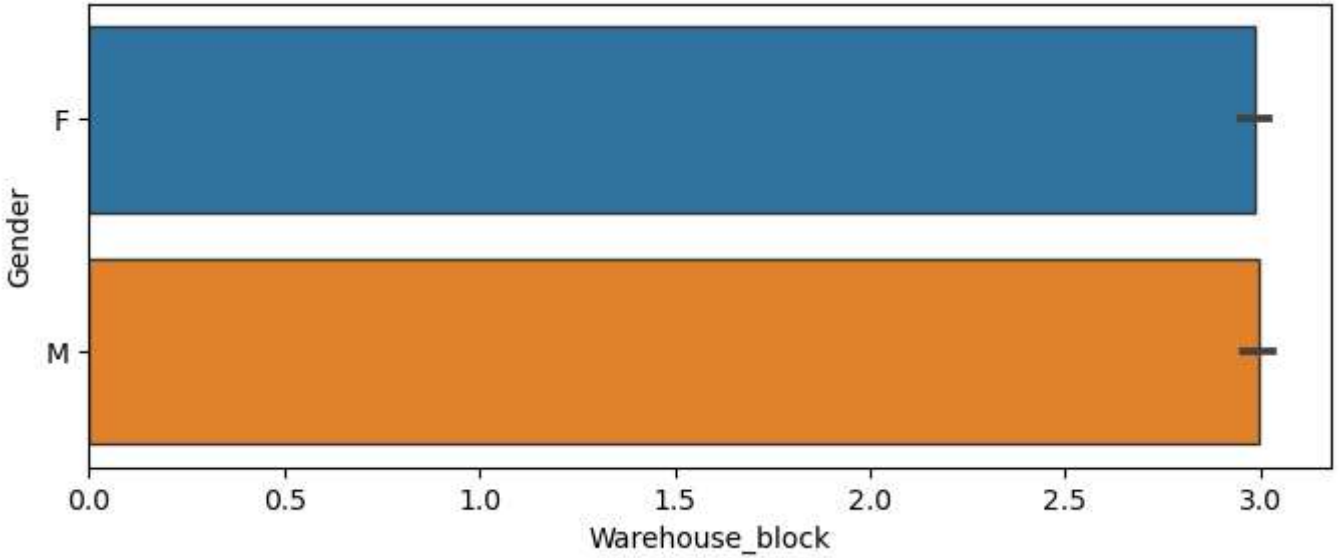

D
F
A
B
C
Warehouse_block
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Warehouse_block

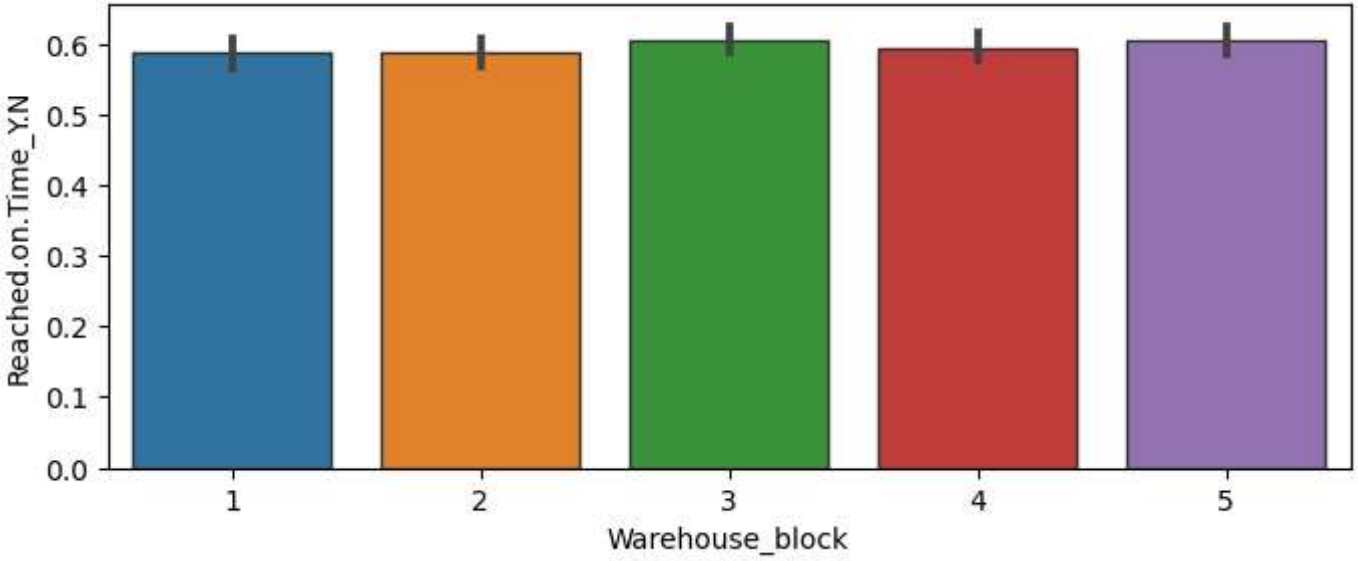

Flight
Ship
Road
Mode_of_Shipment
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Warehouse_block


Customer_rating
0
1
2
3
4
5
1
2
3
4
5
Warehouse_block

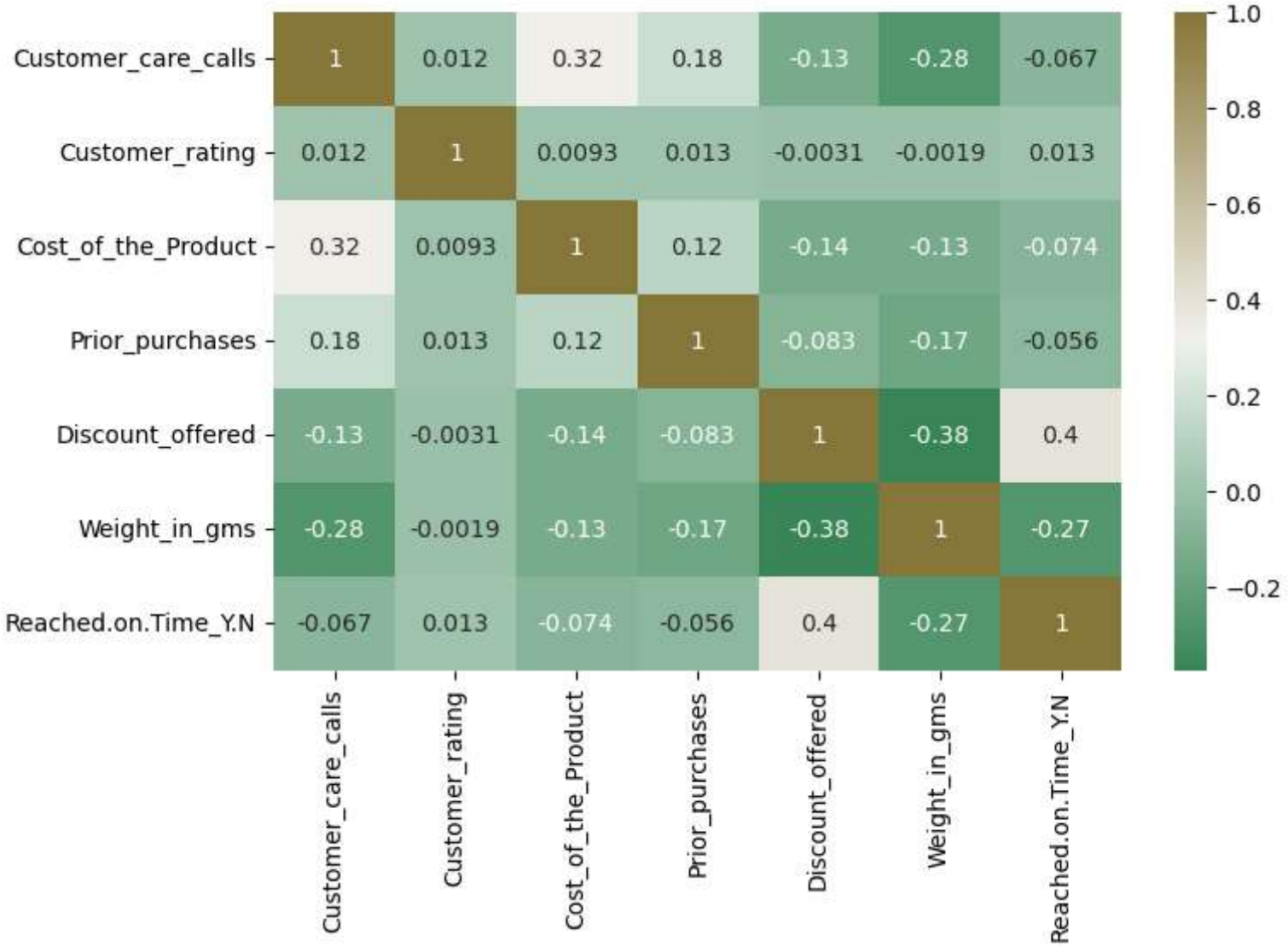

Prior_purchases
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1
2
3
4
5
Warehouse_block


low
medium
high
Product_importance
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Warehouse_block


F
M
Gender
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Warehouse_block

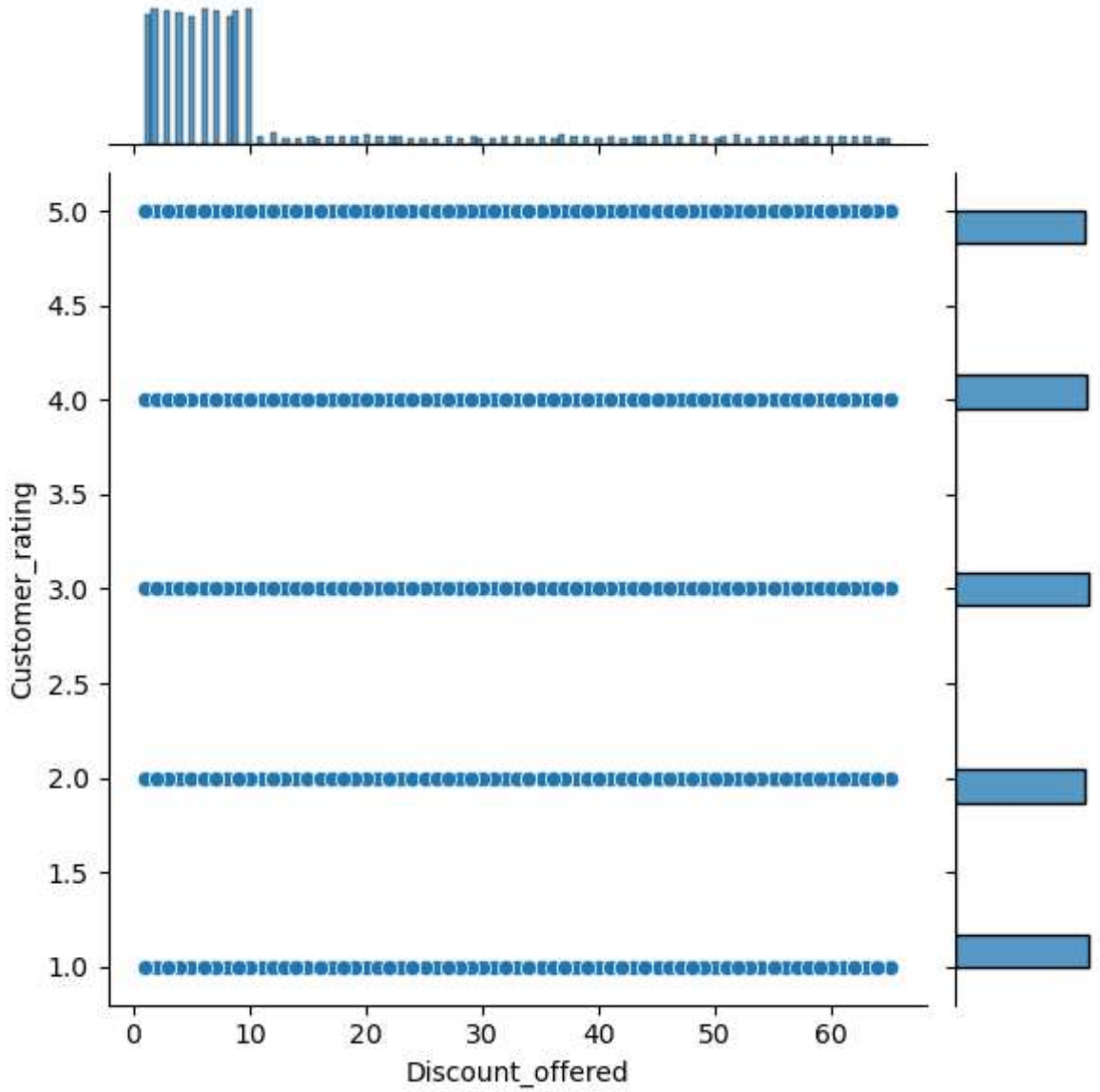


In [26]:
```
color  = sns.diverging_palette(500, 70, as_cmap=True)
fig, ax = plt.subplots(figsize=(8,5))
dataplot=sns.heatmap(df.corr() , cmap = color , annot=True)
plt.show()
```

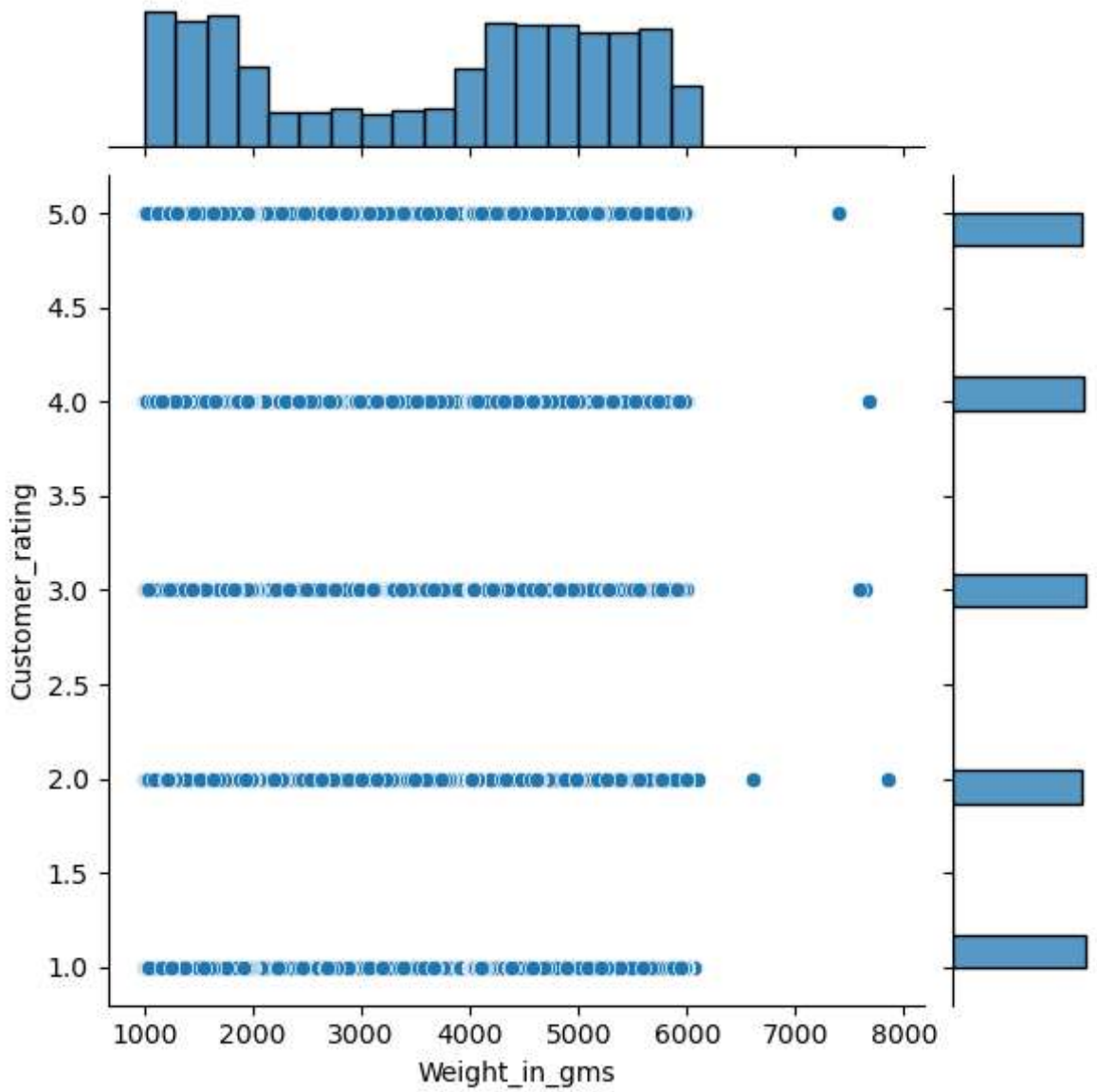


In [35]:
```
for i in catogary_3 :
    plt.figure(figsize=(10, 10))
    sns.jointplot(x=i ,y='Customer_rating',data=df)
```

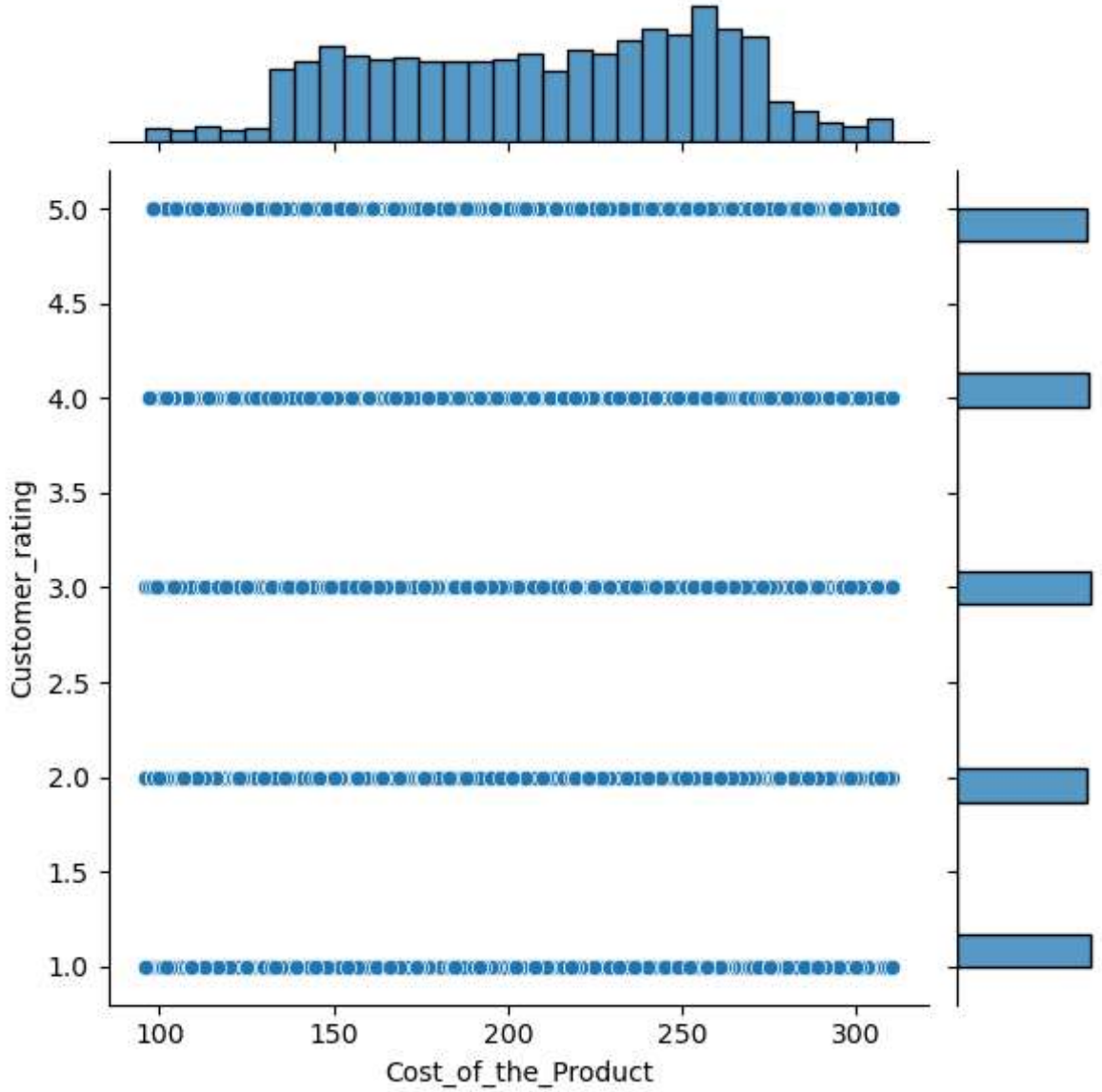
```
<Figure size 1000x1000 with 0 Axes>
```



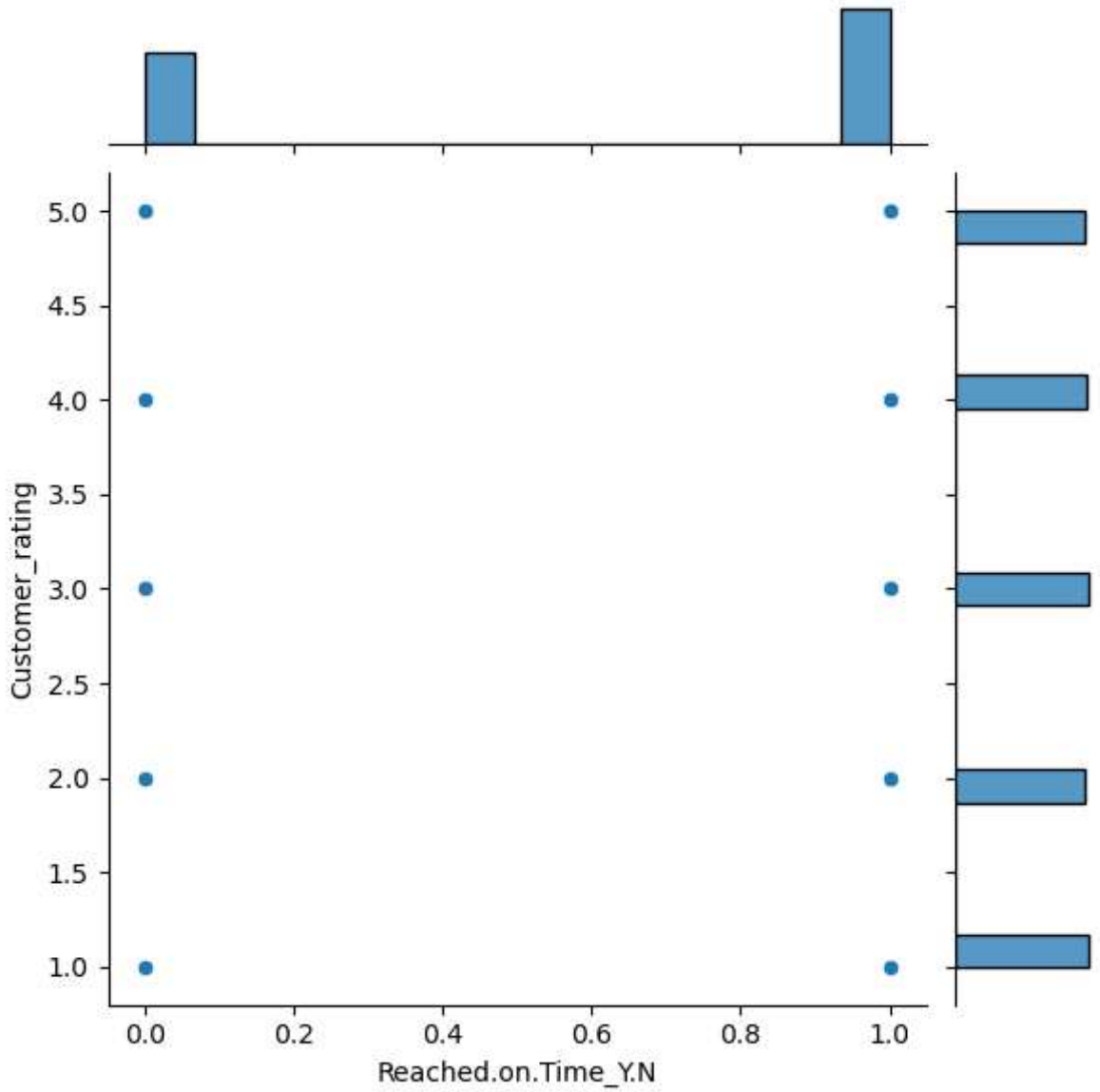
<Figure size 1000x1000 with 0 Axes>



<Figure size 1000x1000 with 0 Axes>



<Figure size 1000x1000 with 0 Axes>



Data Prapering

In [36]:
```
df.columns
```

Out[36]:
```
Index(['Warehouse_block', 'Mode_of_Shipment', 'Customer_care_calls',
       'Customer_rating', 'Cost_of_the_Product', 'Prior_purchases',
       'Product_importance', 'Gender', 'Discount_offered', 'Weight_in_gms',
       'Reached.on.Time_Y.N'],
      dtype='object')
```

In [37]:
```
x=df.drop("Reached.on.Time_Y.N",axis=1)
x.head()
```

Out[37]:

	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importance	Ger
0	D	Flight	4	2	177	3	low	
1	F	Flight	4	5	216	2	low	
2	A	Flight	2	2	183	4	low	
3	B	Flight	3	3	176	4	medium	
4	C	Flight	2	2	184	3	medium	

In [38]:
```
y=df["Reached.on.Time_Y.N"]
y.head()
```

Out[38]:
```
0    1
1    1
2    1
3    1
4    1
Name: Reached.on.Time_Y.N, dtype: int64
```

In [39]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
catogary =['Warehouse_block', 'Mode_of_Shipment', 'Product_importance', 'Gender']
one_hot=OneHotEncoder()
transformer = ColumnTransformer([("one_hot",one_hot,catogary)],remainder="passthrough")
transformed_x = transformer.fit_transform(x)
features = pd.DataFrame(transformed_x)
features.head()
```

Out[39]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	4.0	2.0	177.0	3.0	44.0	1233.0
1	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	4.0	5.0	216.0	2.0	59.0	3088.0
2	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	2.0	2.0	183.0	4.0	48.0	3374.0
3	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	3.0	3.0	176.0	4.0	10.0	1177.0
4	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	2.0	2.0	184.0	3.0	46.0	2484.0

Data spliting

X_train

X_test

Y_train

Y_test

In [40]:
```
from sklearn.model_selection import train_test_split
x_train ,x_test ,y_train ,y_test = train_test_split(features,y,test_size=0.2,shuffle=True)
```

In [41]:
```
x_train.shape ,x_test.shape ,y_train.shape ,y_test.shape
```

Out[41]:
```
((8799, 19), (2200, 19), (8799,), (2200,))
```

Machine Learning (Random Forest Model)

In [42]:
```
from sklearn.model_selection import RandomizedSearchCV
grid = {"n_estimators": [10, 100, 200, 500, 1000, 1200],
        "max_depth": [None, 5, 10, 20, 30],
        "max_features": ["auto", "sqrt"],
        "min_samples_split": [2, 4, 6],
        "min_samples_leaf": [1, 2, 4]}
np.random.seed(42)
from sklearn.ensemble import RandomForestClassifier
model =RandomForestClassifier(n_jobs=-1)
```

```
Search_cv =RandomizedSearchCV(estimator=model
                              ,param_distributions=grid
                              ,n_iter=10
                              ,cv=5
                              ,verbose=2)
Search_cv.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; total time=  13.6s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; total time=   0.9s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; total time=   1.2s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; total time=   1.2s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; total time=   0.9s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   1.
8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   1.
2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   0.
9s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   1.
4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   1.
7s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; total time=  13.3
s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; total time=  11.6
s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; total time=   9.8
s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; total time=   9.6
s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; total time=   9.6
s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   2.
0s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   3.
1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   2.
9s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   1.
9s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   3.
0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   1.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.8s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.8s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.8s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=1200; total time=  1
4.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=1200; total time=  1
4.5s
```

```
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=1200; total time=  1
4.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=1200; total time=  1
2.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=1200; total time=  1
2.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=500; total time=   3.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=500; total time=   3.3s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=500; total time=   3.3s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=500; total time=   3.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=500; total time=   4.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   8.0
s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   7.8
s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   7.7
s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=  13.0
s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   9.0
s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=10; total time=   0.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=10; total time=   0.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=10; total time=   0.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=10; total time=   0.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1200; total time=  15.1
s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1200; total time=  13.4
s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1200; total time=  17.2
s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1200; total time=  16.6
s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1200; total time=  15.2
s
```

Out[42]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(n_jobs=-1),
                   param_distributions={'max_depth': [None, 5, 10, 20, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 4, 6],
                                        'n_estimators': [10, 100, 200, 500,
                                                         1000, 1200]},
                   verbose=2)
```

In [43]:
```
Search_cv.best_params_
```

Out[43]:
```
{'n_estimators': 500,
 'min_samples_split': 4,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 10}
```

In [44]:
```
from sklearn.ensemble import RandomForestClassifier
Random = RandomForestClassifier( n_estimators = 500,
 min_samples_split = 4,
 min_samples_leaf = 2,
 max_features = 'sqrt',
 max_depth = 10).fit(x_train,y_train)
```

In [45]:
```
Random.score(x_test,y_test)
```

Out[45]:
```
0.6672727272727272
```

In [46]:
```
y_predict_Random = Random.predict(x_test)
y_predict_Random
```

Out[46]:
```
array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

In [47]:
```
y_test= np.array(y_test)
```

In [48]:
```
compare_rf = pd.DataFrame(data=[y_test,y_predict_Random])
compare_rf
```

Out[48]:

	0	1	2	3	4	5	6	7	8	9	...	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199
0	0	0	1	0	0	1	0	1	1	0	...	1	1	1	1	0	1	0	0	1	1
1	0	0	1	0	0	1	0	0	0	0	...	0	0	1	0	0	1	0	0	0	0

2 rows × 2200 columns

In [49]:
```
compare_rf =compare_rf.transpose()
```

In [50]:
```
compare_rf.rename(columns = {0:'Real',1:'predict'}, inplace = True)
```

In [51]:
```
compare_rf
```

Out[51]:

	Real	predict
0	0	0
1	0	0
2	1	1
3	0	0
4	0	0
...	...	...
2195	1	1
2196	0	0
2197	0	0
2198	1	0
2199	1	0

2200 rows × 2 columns

In [52]:
```
from sklearn.model_selection import cross_val_score
np.random.seed(42)
for i in range(10, 100, 10):
    print(f"Trying model with {i} estimators...")
    model = RandomForestClassifier(n_estimators= i).fit(x_train,y_train)
    print(f"Model accruacy on test set: {model.score(x_test, y_test)}")
    print(f"Cross-validation score: {np.mean(cross_val_score(model,features, y, cv=5)) * 100}%")
    print("")
```

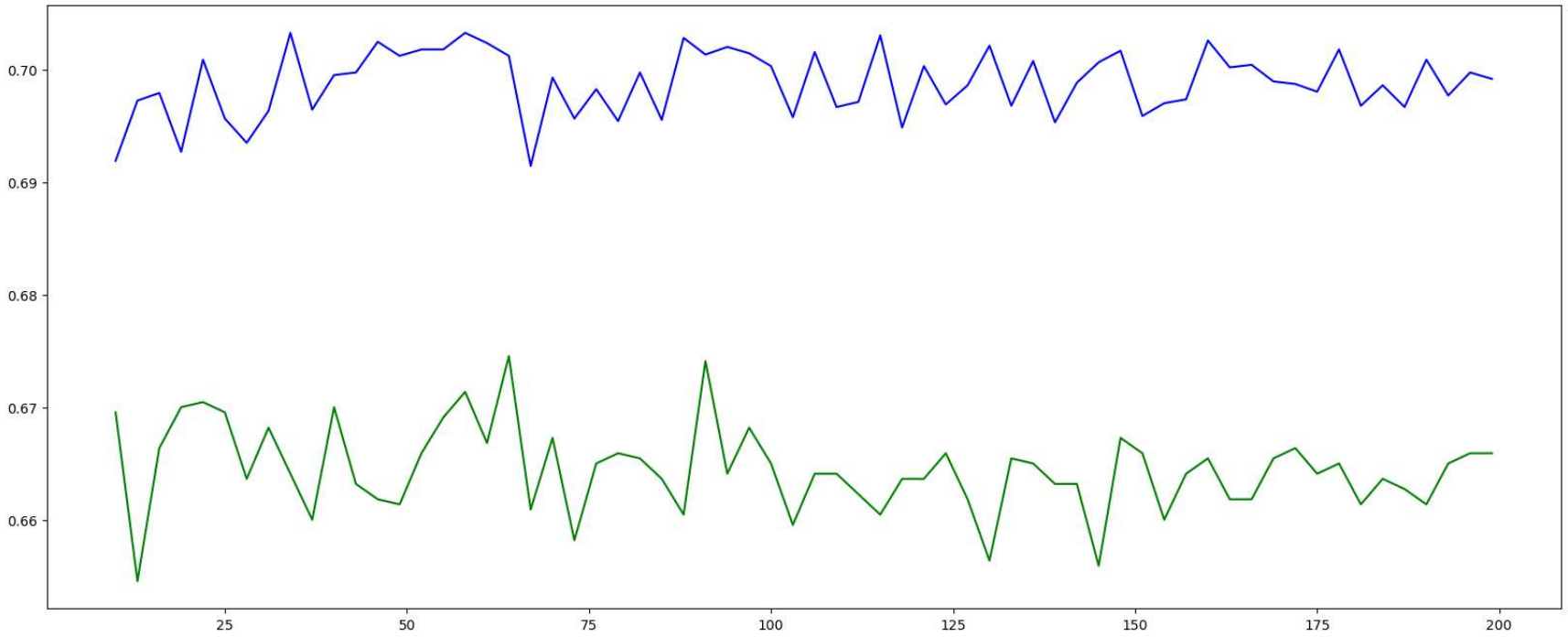
```
Trying model with 10 estimators...
Model accruacy on test set: 0.6577272727272727
Cross-validation score: 63.24029930960353%

Trying model with 20 estimators...
Model accruacy on test set: 0.6668181818181819
Cross-validation score: 61.30404729422465%

Trying model with 30 estimators...
Model accruacy on test set: 0.65
Cross-validation score: 61.42207615031625%

Trying model with 40 estimators...
Model accruacy on test set: 0.639090909090909
Cross-validation score: 60.71298110711481%

Trying model with 50 estimators...
Model accruacy on test set: 0.6445454545454545
Cross-validation score: 60.87664227541445%

Trying model with 60 estimators...
Model accruacy on test set: 0.6422727272727272
Cross-validation score: 61.149328207036255%

Trying model with 70 estimators...
Model accruacy on test set: 0.6509090909090909
Cross-validation score: 60.78575385505809%

Trying model with 80 estimators...
Model accruacy on test set: 0.6445454545454545
Cross-validation score: 60.631191864070445%

Trying model with 90 estimators...
Model accruacy on test set: 0.6372727272727273
Cross-validation score: 60.50384885691843%
```

In [56]:

```
NUM=[]
MM=[]
MMM=[]
import matplotlib.pyplot as plt
import numpy as np
for a in range (10 ,200 ,3):
    Random_Force_regressor=RandomForestClassifier(
        n_estimators= a , max_depth=5 , n_jobs=-1 )
```

```
    Random_Force_regressor.fit(x_train , y_train)
    y_pred=Random_Force_regressor.predict(x_test)
    TES=Random_Force_regressor.score(x_test , y_test)
    score=Random_Force_regressor.score(x_train , y_train)
    MMM.append(TES)
    MM.append(score)
    NUM.append(a)
```

In [57]:
```
plt.plot(NUM, MM, 'b')
plt.plot(NUM, MMM, 'g')
plt.show()
```



Machine Learning (KNeighborsClassifier)

In [58]:
```
# tunning parameters in knn calssification
from sklearn.model_selection import RandomizedSearchCV
grid = {"n_neighbors": [1,2,3,4,5,6,8,9,10],
        "weights": ['uniform', 'distance'],
        "p": [1, 2],
        "algorithm":  ['auto', 'ball_tree', 'kd_tree', 'brute'],
        "leaf_size": [15, 20,30]}
```

```
from sklearn.neighbors import KNeighborsClassifier
model =KNeighborsClassifier(n_jobs=-1)
search_cv =RandomizedSearchCV(estimator=model
                              ,param_distributions=grid
                              ,n_iter=20
                              ,cv=5
                              ,verbose=2)
search_cv.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] END algorithm=auto, leaf_size=20, n_neighbors=3, p=1, weights=distance; total time=   1.0s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=3, p=1, weights=distance; total time=   1.1s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=3, p=1, weights=distance; total time=   1.4s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=3, p=1, weights=distance; total time=   1.3s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=3, p=1, weights=distance; total time=   0.9s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   1.6s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   1.6s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   1.0s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   1.3s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   1.9s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   9.4s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   1.0s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   0.9s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   0.9s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   1.7s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   1.0s
[CV] END algorithm=kd_tree, leaf_size=30, n_neighbors=3, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=30, n_neighbors=3, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=30, n_neighbors=3, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=30, n_neighbors=3, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=30, n_neighbors=3, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=brute, leaf_size=30, n_neighbors=1, p=2, weights=distance; total time=   1.0s
[CV] END algorithm=brute, leaf_size=30, n_neighbors=1, p=2, weights=distance; total time=   1.5s
[CV] END algorithm=brute, leaf_size=30, n_neighbors=1, p=2, weights=distance; total time=   0.8s
[CV] END algorithm=brute, leaf_size=30, n_neighbors=1, p=2, weights=distance; total time=   0.7s
[CV] END algorithm=brute, leaf_size=30, n_neighbors=1, p=2, weights=distance; total time=   0.7s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=4, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=4, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=4, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=4, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=4, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=brute, leaf_size=15, n_neighbors=1, p=1, weights=uniform; total time=   1.4s
[CV] END algorithm=brute, leaf_size=15, n_neighbors=1, p=1, weights=uniform; total time=   1.5s
[CV] END algorithm=brute, leaf_size=15, n_neighbors=1, p=1, weights=uniform; total time=   1.1s
[CV] END algorithm=brute, leaf_size=15, n_neighbors=1, p=1, weights=uniform; total time=   1.4s
[CV] END algorithm=brute, leaf_size=15, n_neighbors=1, p=1, weights=uniform; total time=   1.7s
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
```

```
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=10, p=2, weights=distance; total time=   0.1s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=10, p=2, weights=distance; total time=   0.1s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=10, p=2, weights=distance; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=10, p=2, weights=distance; total time=   0.1s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=10, p=2, weights=distance; total time=   0.1s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=8, p=1, weights=distance; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=8, p=1, weights=distance; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=8, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=8, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=8, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=3, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=2, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=2, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=2, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=2, p=1, weights=uniform; total time=   0.1s
[CV] END algorithm=kd_tree, leaf_size=20, n_neighbors=2, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=6, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=6, p=1, weights=distance; total time=   0.0s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=6, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=6, p=1, weights=distance; total time=   0.0s
[CV] END algorithm=ball_tree, leaf_size=20, n_neighbors=6, p=1, weights=distance; total time=   0.1s
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=9, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=9, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=9, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=9, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=ball_tree, leaf_size=15, n_neighbors=9, p=1, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=kd_tree, leaf_size=15, n_neighbors=1, p=2, weights=uniform; total time=   0.2s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   0.9s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   0.9s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   0.9s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   0.9s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=4, p=2, weights=distance; total time=   1.3s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   1.0s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   1.0s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   0.9s
[CV] END algorithm=auto, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   0.9s
```

```
[CV] END algorithm=auto, leaf_size=20, n_neighbors=9, p=2, weights=uniform; total time=   1.1s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=2, p=2, weights=distance; total time=   0.7s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=2, p=2, weights=distance; total time=   0.7s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=2, p=2, weights=distance; total time=   0.7s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=2, p=2, weights=distance; total time=   0.8s
[CV] END algorithm=brute, leaf_size=20, n_neighbors=2, p=2, weights=distance; total time=   0.7s
[CV] END algorithm=auto, leaf_size=30, n_neighbors=8, p=1, weights=distance; total time=   0.9s
[CV] END algorithm=auto, leaf_size=30, n_neighbors=8, p=1, weights=distance; total time=   1.0s
[CV] END algorithm=auto, leaf_size=30, n_neighbors=8, p=1, weights=distance; total time=   1.0s
[CV] END algorithm=auto, leaf_size=30, n_neighbors=8, p=1, weights=distance; total time=   1.0s
[CV] END algorithm=auto, leaf_size=30, n_neighbors=8, p=1, weights=distance; total time=   1.0s
```

Out[58]:
```
RandomizedSearchCV(cv=5, estimator=KNeighborsClassifier(n_jobs=-1), n_iter=20,
                   param_distributions={'algorithm': ['auto', 'ball_tree',
                                                      'kd_tree', 'brute'],
                                        'leaf_size': [15, 20, 30],
                                        'n_neighbors': [1, 2, 3, 4, 5, 6, 8, 9,
                                                        10],
                                        'p': [1, 2],
                                        'weights': ['uniform', 'distance']},
                   verbose=2)
```

In [59]:
```
search_cv.best_params_
```

Out[59]:
```
{'weights': 'distance',
 'p': 2,
 'n_neighbors': 10,
 'leaf_size': 20,
 'algorithm': 'kd_tree'}
```

In [60]:
```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(weights = 'uniform',
 p = 2,
 n_neighbors = 4,
 leaf_size = 30,
 algorithm = 'ball_tree').fit(x_train,y_train)
```

In [61]:
```
KNN.score(x_test,y_test)
```

Out[61]:
```
0.6627272727272727
```

In [62]:
```
y_predict_knn = KNN.predict(x_test)
y_predict_knn
```

Out[62]:
```
array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

In [63]:
```
y_test= np.array(y_test)
```

In [64]:
```
compare_knn = pd.DataFrame(data=[y_test,y_predict_knn])
compare_knn
```

Out[64]:

	0	1	2	3	4	5	6	7	8	9	...	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199
0	0	0	1	0	0	1	0	1	1	0	...	1	1	1	1	0	1	0	0	1	1
1	0	0	1	0	0	1	0	0	0	0	...	0	0	1	0	0	1	0	0	0	0

2 rows × 2200 columns

In [65]:
```
compare_knn =compare_knn.transpose()
```

In [66]:
```
compare_knn.rename(columns = {0:'Real',1:'predict'}, inplace = True)
```

In [67]:
```
compare_knn
```

Out[67]:

	Real	predict
0	0	0
1	0	0
2	1	1
3	0	0
4	0	0
...	...	...
2195	1	1
2196	0	0
2197	0	0
2198	1	0
2199	1	0

2200 rows × 2 columns

Machine Learning (Naive Bayes Model)

In [68]:
```
from sklearn.naive_bayes import GaussianNB
NB =GaussianNB().fit(x_train,y_train)
```

In [69]:
```
NB.score(x_test,y_test)
```

Out[69]:
```
0.639090909090909
```

In [70]:
```
y_predict_NB = NB.predict(x_test)
y_predict_NB
```

Out[70]:
```
array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

In [71]:
```
compare_NB = pd.DataFrame(data=[y_test,y_predict_NB])
compare_NB
```

Out[71]:

	0	1	2	3	4	5	6	7	8	9	...	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199
0	0	0	1	0	0	1	0	1	1	0	...	1	1	1	1	0	1	0	0	1	1
1	0	0	1	0	0	0	0	0	0	0	...	0	0	1	0	0	1	0	0	0	0

2 rows × 2200 columns

In [72]:
```
compare_NB =compare_knn.transpose()
```

In [73]:
```
compare_NB.rename(columns = {0:'Real',1:'predict'}, inplace = True)
```

In [74]:
```
compare_NB
```

Out[74]:

	Real	predict	2	3	4	5	6	7	8	9	...	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199
Real	0	0	1	0	0	1	0	1	1	0	...	1	1	1	1	0	1	0	0	1	1
predict	0	0	1	0	0	1	0	0	0	0	...	0	0	1	0	0	1	0	0	0	0

2 rows × 2200 columns

Clustring(K_Mean)

In [75]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2,init='k-means++',
    tol=0.001).fit(x_train)

clus=kmeans.predict(x_test)
clus
```

Out[75]:
```
array([1, 1, 0, ..., 1, 1, 0])
```

In [76]:
```
compare_k_mean = pd.DataFrame(data=[y_test,clus])
compare_k_mean
```

Out[76]:

	0	1	2	3	4	5	6	7	8	9	...	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199
0	0	0	1	0	0	1	0	1	1	0	...	1	1	1	1	0	1	0	0	1	1
1	1	1	0	1	1	0	0	1	1	1	...	1	1	0	1	1	0	0	1	1	0

2 rows × 2200 columns

In [77]:
```
compare_K_mean =compare_k_mean.transpose()
```

In [78]:
```
compare_K_mean.rename(columns = {0:'Real',1:'predict'}, inplace = True)
```

In [79]:
```
compare_K_mean
```

Out[79]:

	Real	predict
0	0	1
1	0	1
2	1	0
3	0	1
4	0	1
...	...	...
2195	1	0
2196	0	0
2197	0	1
2198	1	1
2199	1	0

2200 rows × 2 columns

In [80]:
```
np.mean(y_test==clus)
```

Out[80]:
```
0.3886363636363636
```

Classification Matrix

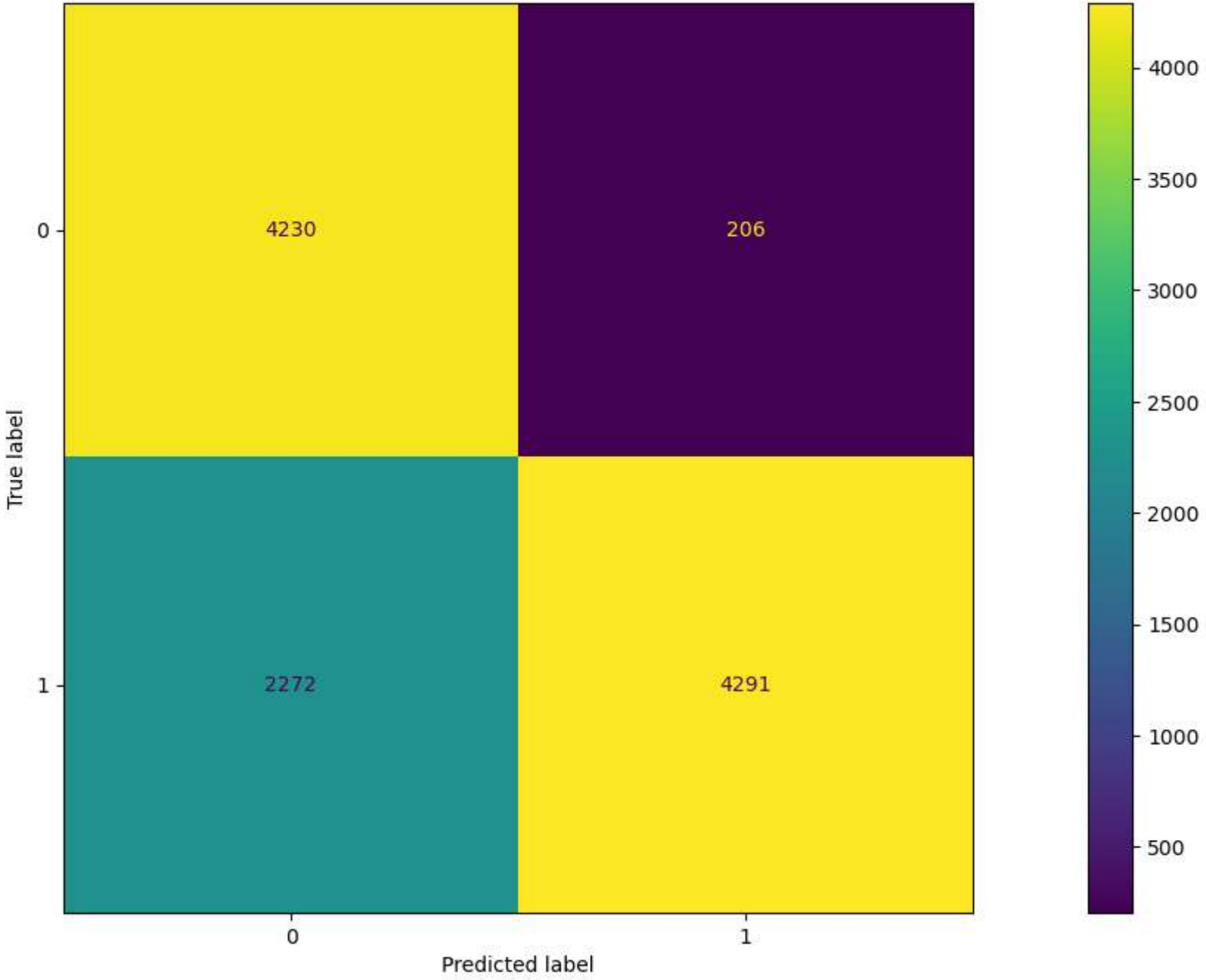
In [82]:
```
def classification_matrix(y_true, y_preds, module,x,y):
    from sklearn.metrics import accuracy_score , precision_score ,recall_score,ConfusionMatrixDisplay,f1_score,classifi
    accuracy = accuracy_score(y_true, y_preds)
    precision = precision_score(y_true, y_preds)
    recall = recall_score(y_true, y_preds)
    f1 = f1_score(y_true, y_preds)
    metric_dict = {"accuracy": round(accuracy, 2),
                   "precision": round(precision, 2),
                   "recall": round(recall, 2),
                   "f1": round(f1, 2)}
    print("_____________________________________________________________")
    print("___accuracy_score , precision_score ,recall_score,f1_score__")
```

```
    print("_____________________________________________________________")
    print(f"Acc: {accuracy * 100:.2f}%")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1 score: {f1:.2f}" )
    print("_____________________________________________________________")
    print("_____________________Classification Report___________________")
    print("_____________________________________________________________")
    print(classification_report(y_true,y_preds))
    print("_____________________________________________________________")
    print("_____________________Confusion Matrix________________________")
    print("_____________________________________________________________")
    ConfusionMatrixDisplay.from_estimator(estimator = Search_cv , X=x ,y=y)
```

Random Forest Model

In [83]:
```
classification_matrix(y_test,y_predict_Random,Random ,features ,y)
```

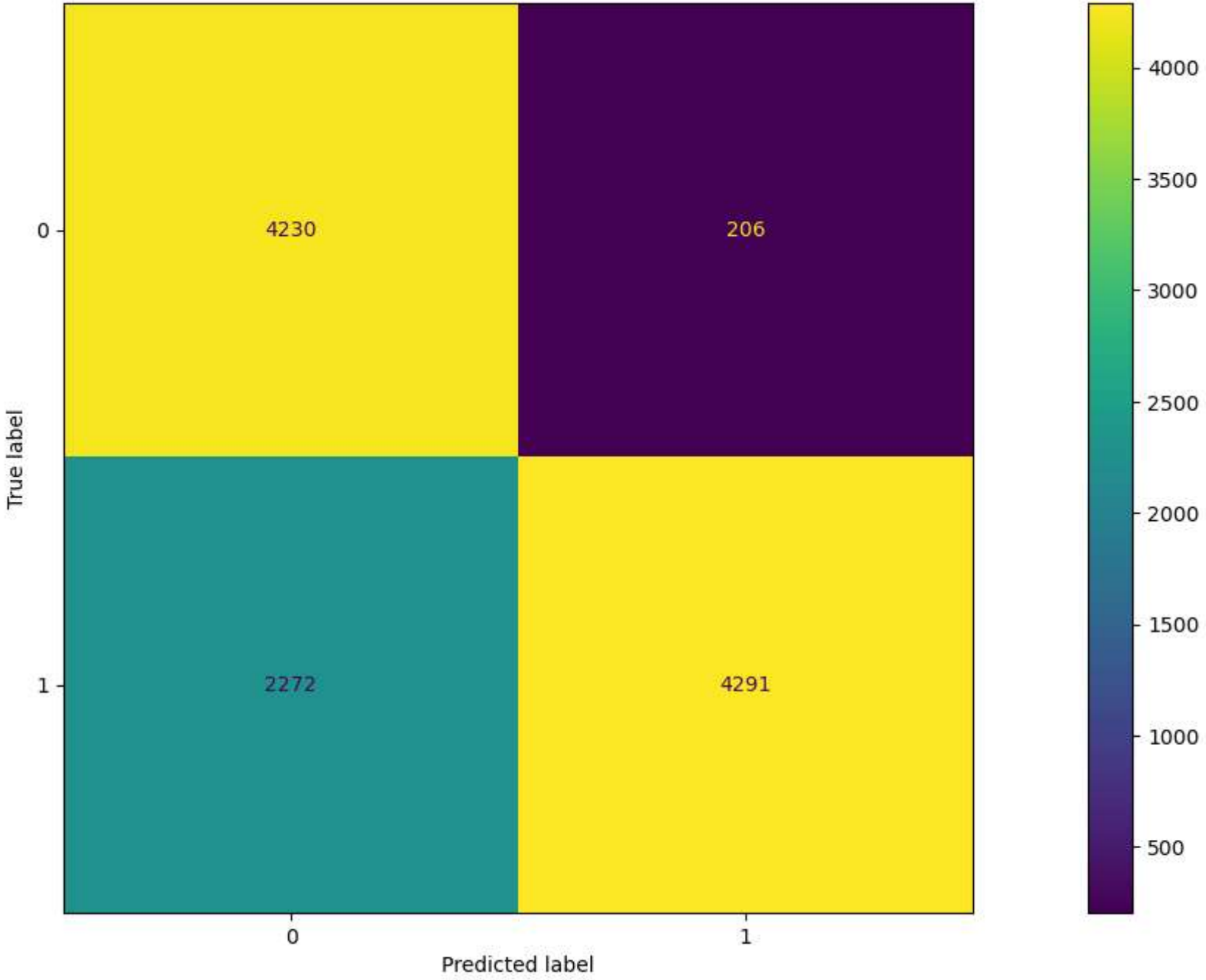
```
_____________________________________________________________
___accuracy_score , precision_score ,recall_score,f1_score__
_____________________________________________________________
Acc: 66.73%
Precision: 0.87
Recall: 0.54
F1 score: 0.66
_____________________________________________________________
_____________________Classification Report____________________
_____________________________________________________________
              precision    recall  f1-score   support

           0       0.54      0.87      0.67       854
           1       0.87      0.54      0.66      1346

    accuracy                           0.67      2200
   macro avg       0.71      0.70      0.67      2200
weighted avg       0.74      0.67      0.67      2200

_____________________________________________________________
_____________________Confusion Matrix________________________
_____________________________________________________________
```


4230
206
2272
4291
0
1
True label
0
1
Predicted label
4000
3500
3000
2500
2000
1500
1000
500

kNN Model

In [84]:
```
classification_matrix(y_test,y_predict_knn,KNN ,features ,y)
```

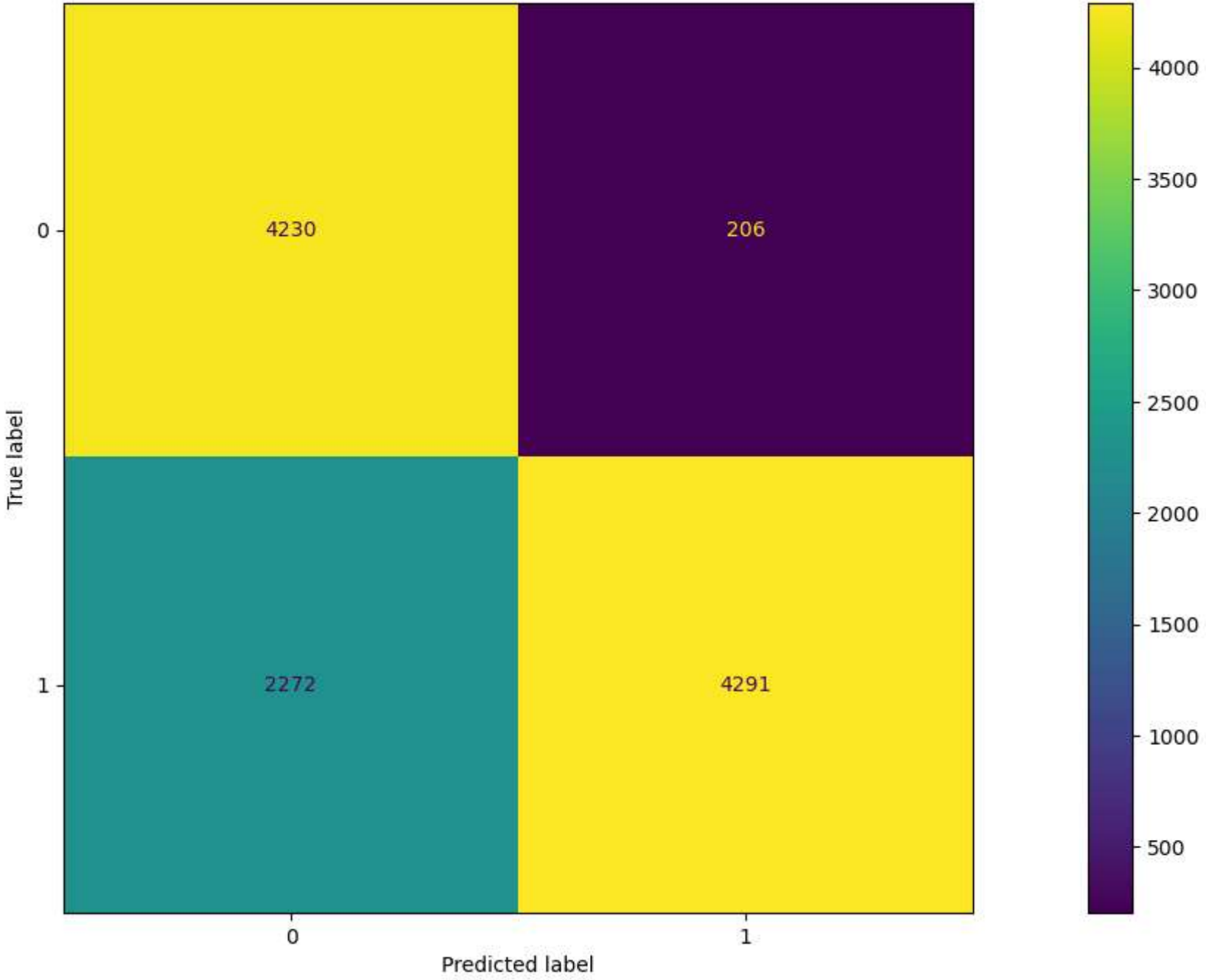
```
_____________________________________________________________
___accuracy_score , precision_score ,recall_score,f1_score__
_____________________________________________________________
Acc: 66.27%
Precision: 0.82
Recall: 0.58
F1 score: 0.68
_____________________________________________________________
_____________________Classification Report____________________
_____________________________________________________________
              precision    recall  f1-score   support

           0       0.54      0.80      0.65       854
           1       0.82      0.58      0.68      1346

    accuracy                           0.66      2200
   macro avg       0.68      0.69      0.66      2200
weighted avg       0.71      0.66      0.67      2200


_____________________________________________________________
_____________________Confusion Matrix________________________
_____________________________________________________________
```


4230
206
2272
4291
0
1
True label
0
1
Predicted label
4000
3500
3000
2500
2000
1500
1000
500

NB Model

In [87]:
```
classification_matrix(y_test,y_predict_NB,NB ,features ,y)
```

```
_____________________________________________________________
___accuracy_score , precision_score ,recall_score,f1_score__
_____________________________________________________________
Acc: 63.91%
Precision: 0.96
Recall: 0.43
F1 score: 0.59
_____________________________________________________________
_____________________Classification Report____________________
_____________________________________________________________
              precision    recall  f1-score   support

           0       0.52      0.97      0.68       854
           1       0.96      0.43      0.59      1346

    accuracy                           0.64      2200
   macro avg       0.74      0.70      0.63      2200
weighted avg       0.79      0.64      0.63      2200


_____________________________________________________________
_____________________Confusion Matrix________________________
_____________________________________________________________
```


4230
206
2272
4291
0
1
True label
0
1
Predicted label
4000
3500
3000
2500
2000
1500
1000
500

Conclusion

In conclusion, the analysis conducted on the e-commerce data involved data preparation, visualization, and the application of various machine learning models. The Random Forest Model, kNN Model, and Naive Bayes Model were used to gain insights and make predictions related to shipping events. Additionally, clustering was performed using the K-Means algorithm to identify patterns in the shipping data. The classification matrix was used to evaluate the performance of the models. The results of this analysis can help the e-commerce company improve their shipping operations, make informed decisions, and enhance customer satisfaction.