In [1]:

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

In [2]:

```
df = pd.read_csv(r"C:\Users\amits\Downloads\E_Commerce.csv")
```

In [3]:

```
df.head()
```

Out[3]:

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

In [4]:

```
# Data Preprocessing
```

In [5]:

```
df.shape
```

Out[5]:

```
(10999, 12)
```

In [6]:

```
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 [7]:

```
# Checking if there are any null values are present?
```

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
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
```

There are no nan values are present in our dataset

In [9]:

```
df.describe()
```

Out[9]:

	ID	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Discount_offe
count	10999.00000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000
mean	5500.00000	4.054459	2.990545	210.196836	3.567597	13.373
std	3175.28214	1.141490	1.413603	48.063272	1.522860	16.205
min	1.00000	2.000000	1.000000	96.000000	2.000000	1.000
25%	2750.50000	3.000000	2.000000	169.000000	3.000000	4.000
50%	5500.00000	4.000000	3.000000	214.000000	3.000000	7.000
75%	8249.50000	5.000000	4.000000	251.000000	4.000000	10.000
max	10999.00000	7.000000	5.000000	310.000000	10.000000	65.000

In [10]:

```
df.tail()
```

Out[10]:

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Produc
10994	10995	A	Ship	4	1	25
10995	10996	B	Ship	4	1	23
10996	10997	C	Ship	5	4	24
10997	10998	F	Ship	5	2	22
10998	10999	D	Ship	2	5	15

In [11]:

```
# Exploratory Data Analysis
```

In [12]:

```
df['Reached.on.Time_Y.N'].value_counts()
```

Out[12]:

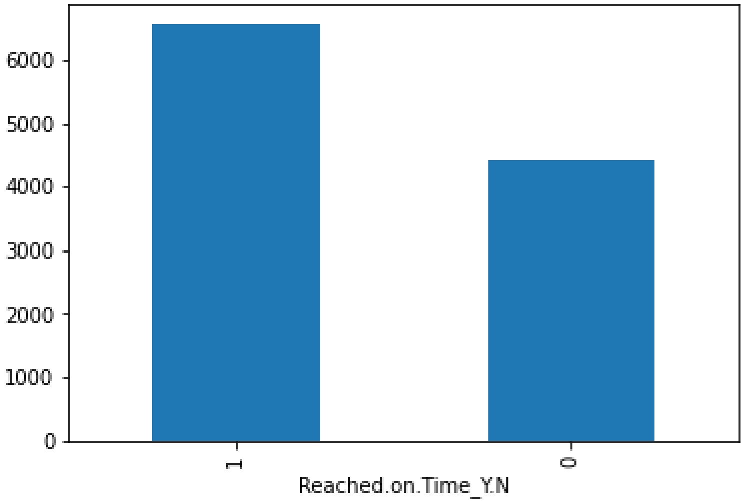
```
Reached.on.Time_Y.N
1    6563
0    4436
Name: count, dtype: int64
```

In [13]:

```
df['Reached.on.Time_Y.N'].value_counts().plot(kind = 'bar')
```

Out[13]:

```
<Axes: xlabel='Reached.on.Time_Y.N'>
```



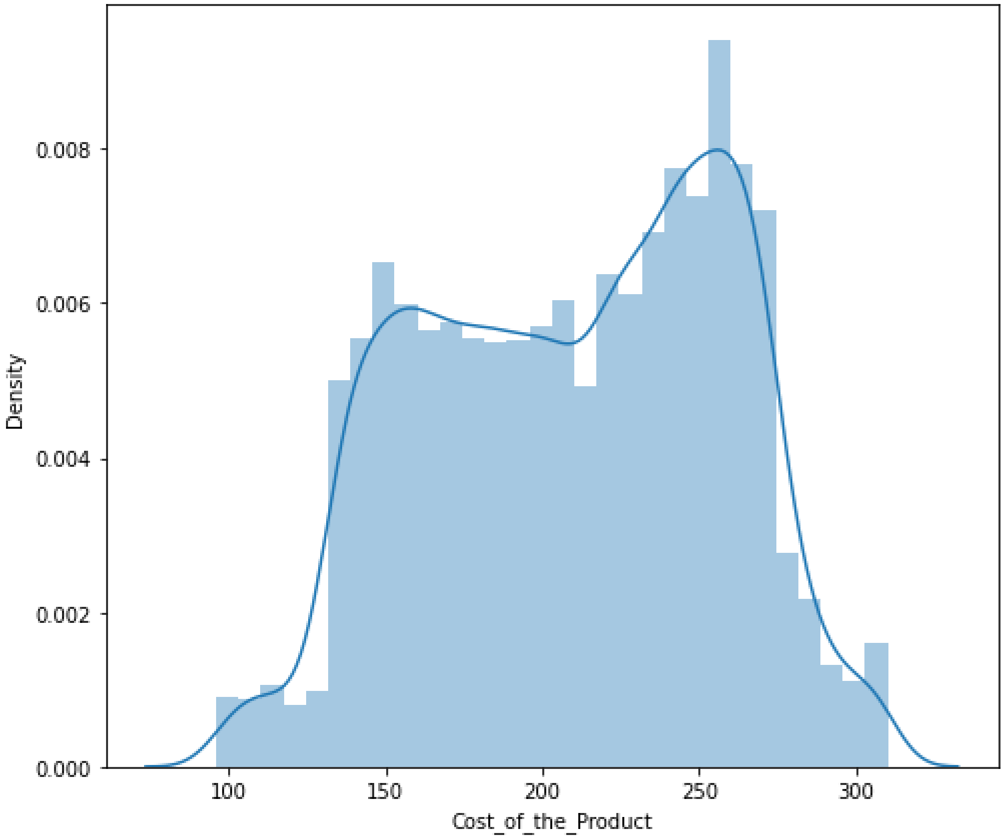
In [14]:

```
plt.figure(figsize=(8,7))
sns.distplot(df['Cost_of_the_Product'])
```

```
C:\Users\amits\AppData\Local\Temp/ipykernel_23648/1537927566.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Cost_of_the_Product'])
```

Out[14]:

```
<Axes: xlabel='Cost_of_the_Product', ylabel='Density'>
```

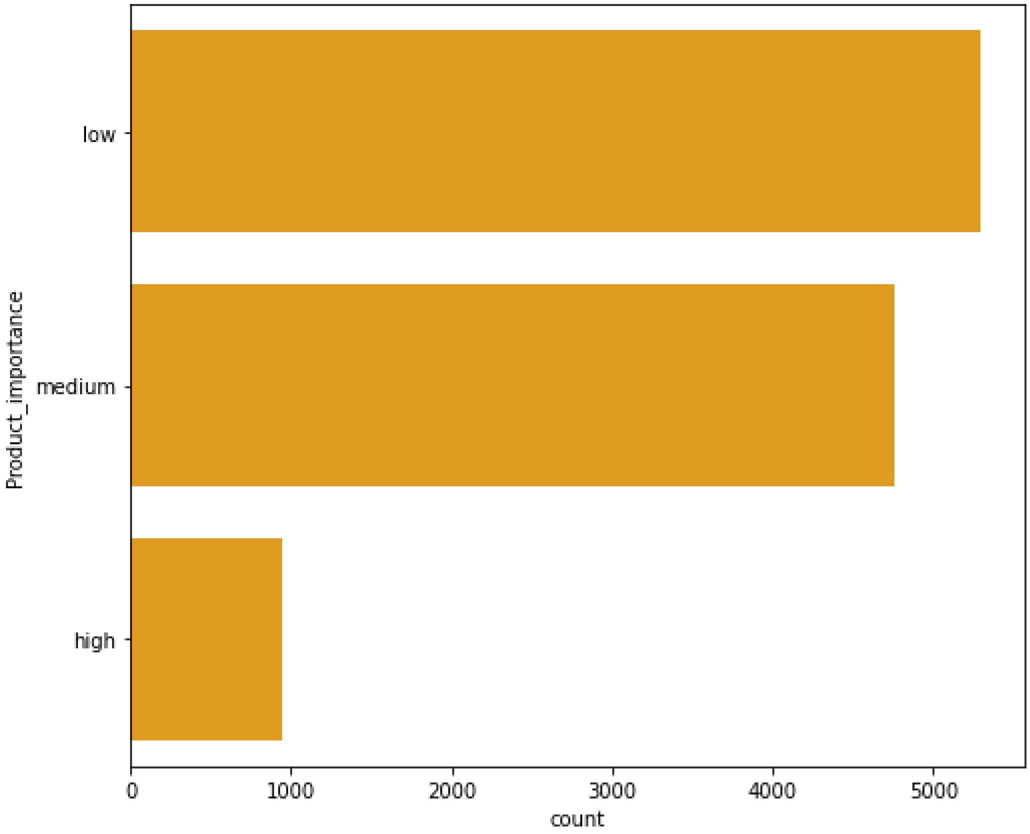


In [15]:

```
plt.figure(figsize = (8,7))
sns.countplot(df['Product_importance'], color = 'orange')
```

Out[15]:

```
<Axes: xlabel='count', ylabel='Product_importance'>
```



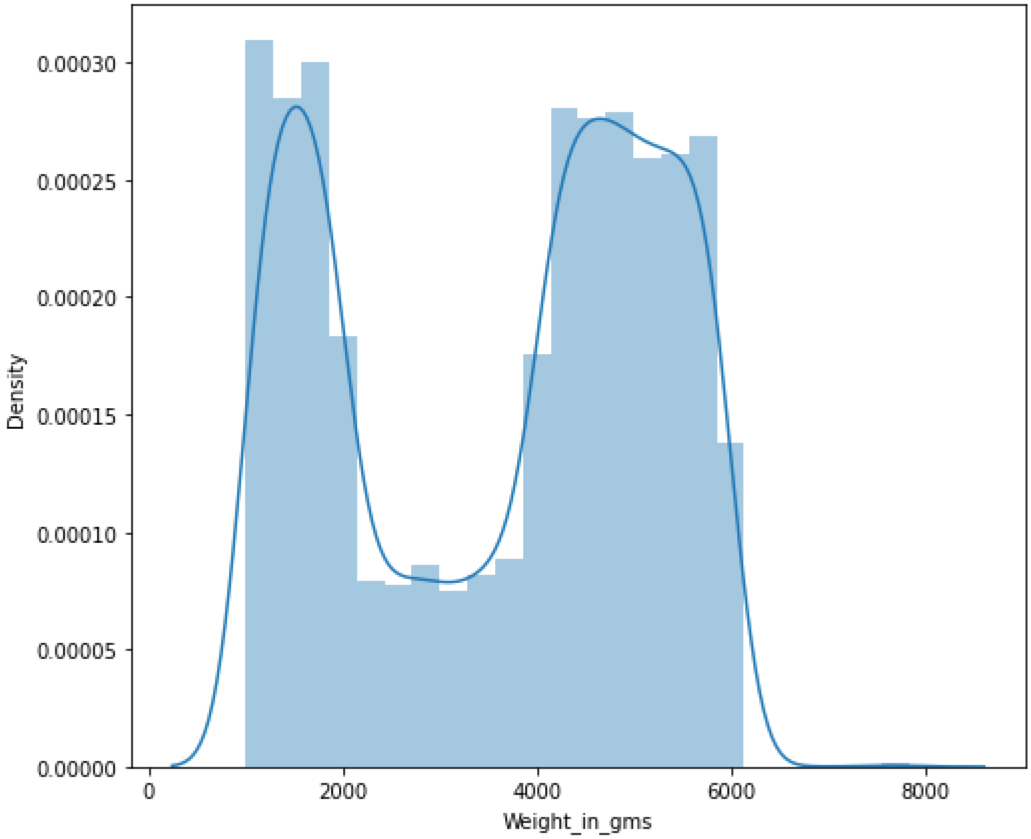
In [16]:

```
plt.figure(figsize = (8,7))
sns.distplot(df['Weight_in_gms'])
```

```
C:\Users\amits\AppData\Local\Temp/ipykernel_23648/3661095990.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Weight_in_gms'])
```

Out[16]:

```
<Axes: xlabel='Weight_in_gms', ylabel='Density'>
```



In [17]:

```
df['Gender'].value_counts()
```

Out[17]:

```
Gender
F    5545
M    5454
Name: count, dtype: int64
```

In [18]:

```
df.head()
```

Out[18]:

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

In [19]:

```
df['Weight_in_gms'].dtype
```

Out[19]:

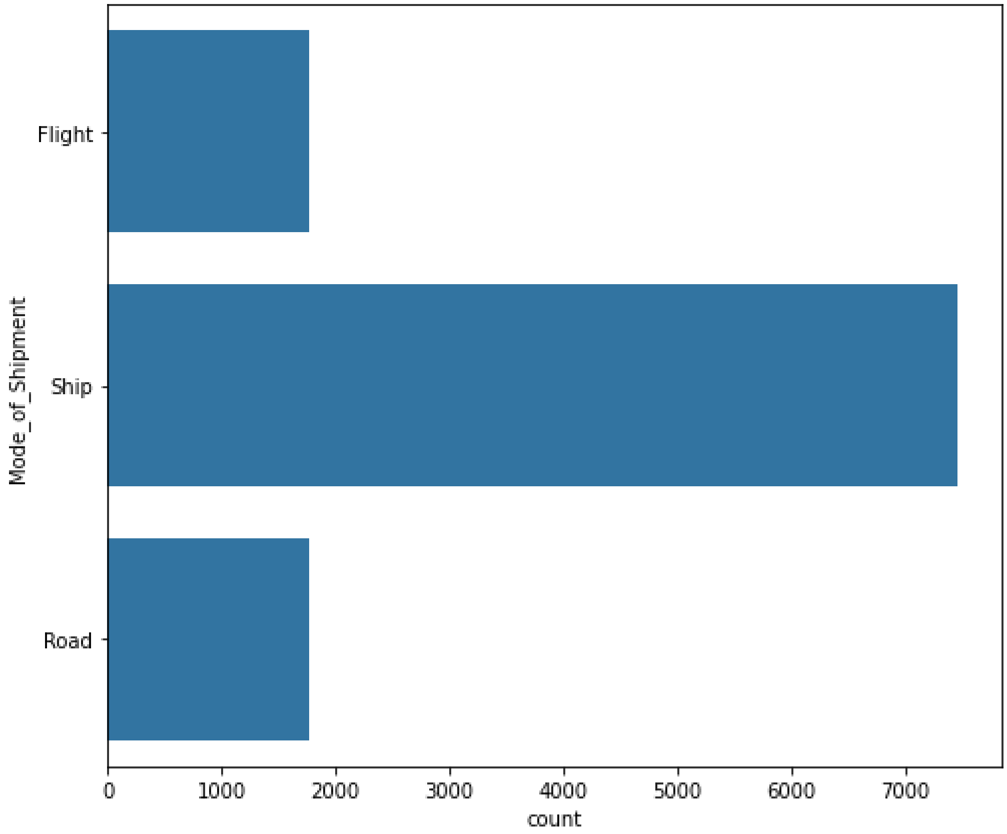
```
dtype('int64')
```

In [20]:

```
plt.figure(figsize = (8,7))
sns.countplot(df['Mode_of_Shipment'])
```

Out[20]:

```
<Axes: xlabel='count', ylabel='Mode_of_Shipment'>
```

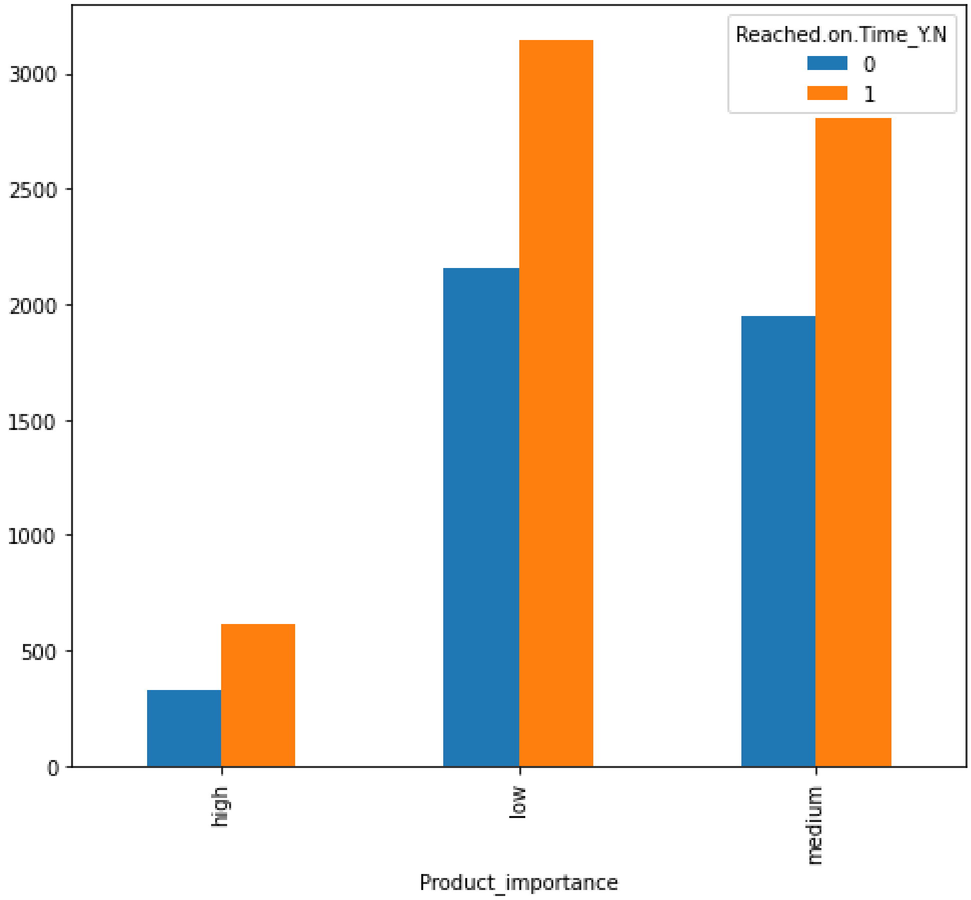


In [21]:

```
df.groupby(['Product_importance', 'Reached.on.Time_Y.N']).size().unstack().plot(kind = '
```

Out[21]:

```
<Axes: xlabel='Product_importance'>
```

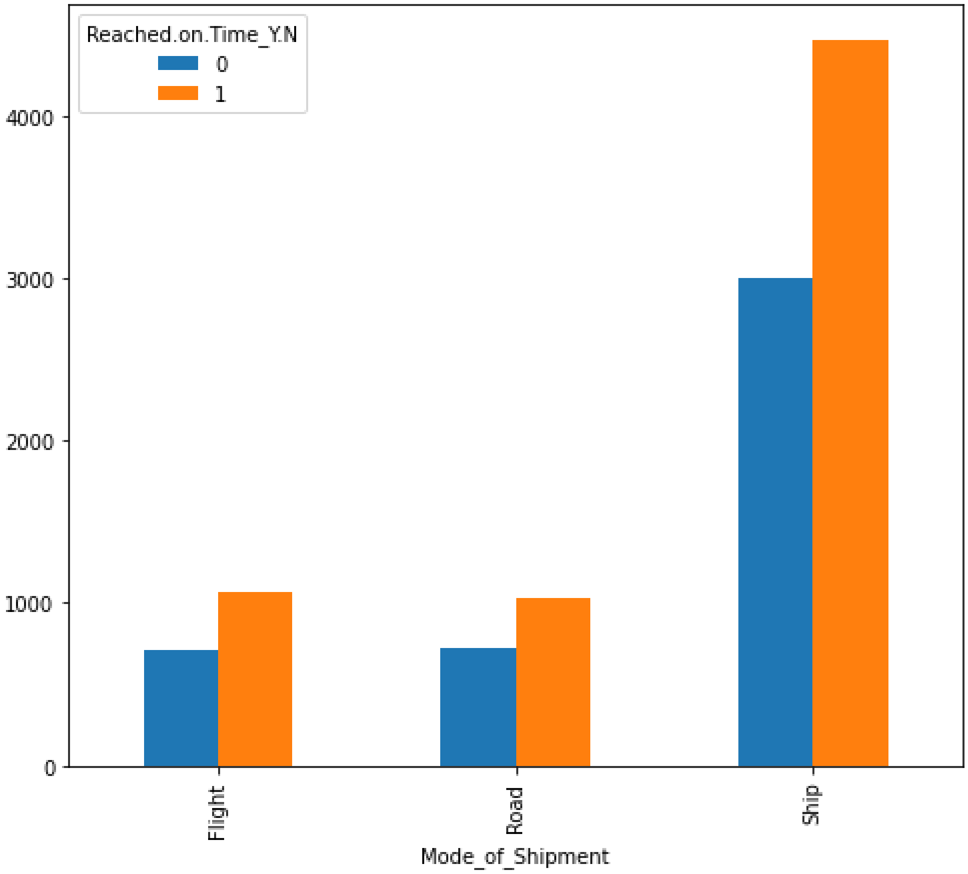


In [22]:

```
df.groupby(['Mode_of_Shipment', 'Reached.on.Time_Y.N']).size().unstack().plot(kind = 'ba
```

Out[22]:

```
<Axes: xlabel='Mode_of_Shipment'>
```

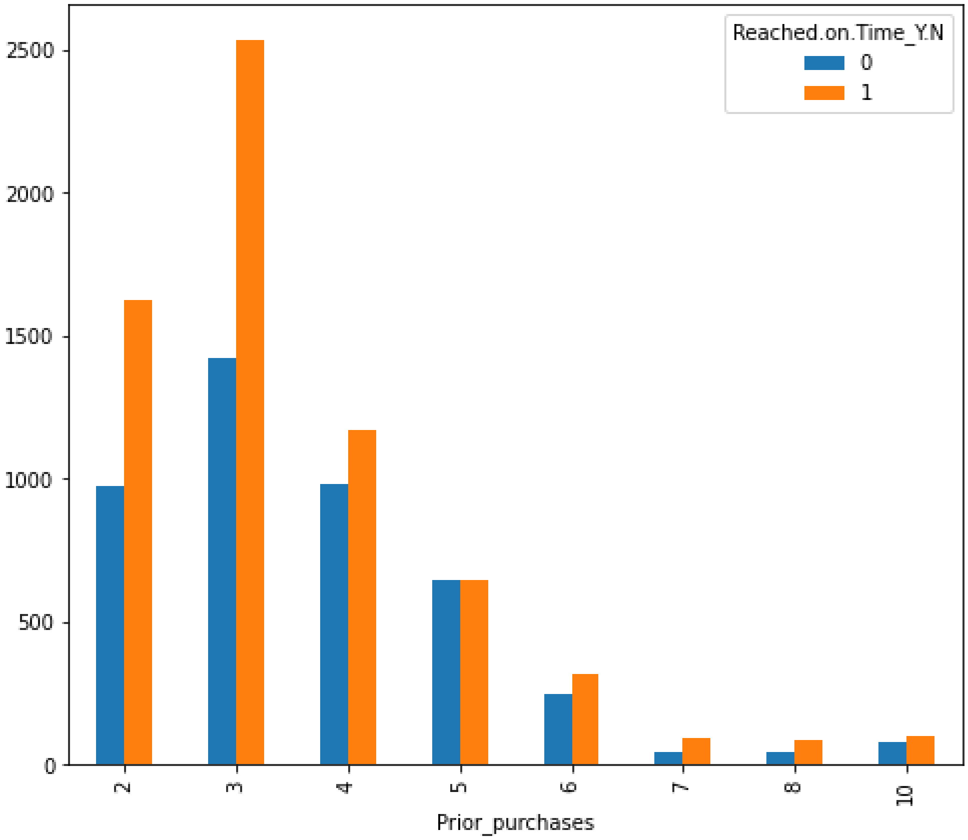


In [23]:

```
df.groupby(['Prior_purchases', 'Reached.on.Time_Y.N']).size().unstack().plot(kind = 'bar
```

Out[23]:

```
<Axes: xlabel='Prior_purchases'>
```

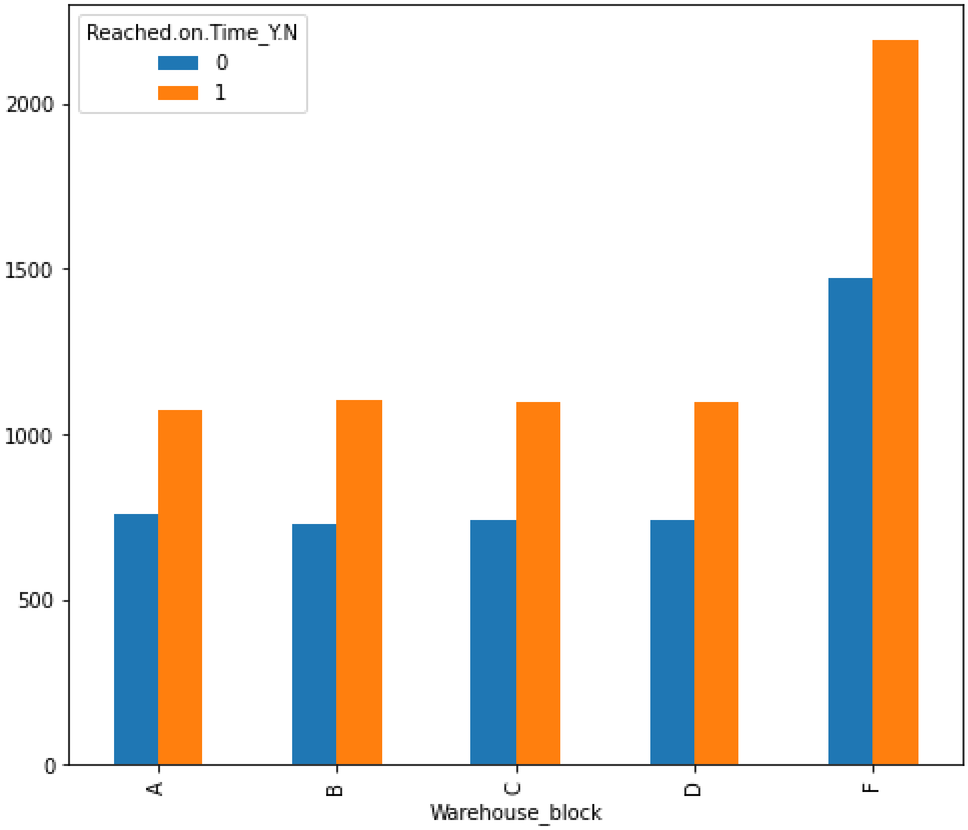


In [24]:

```
df.groupby(['Warehouse_block', 'Reached.on.Time_Y.N']).size().unstack().plot(kind = 'bar
```

Out[24]:

```
<Axes: xlabel='Warehouse_block'>
```



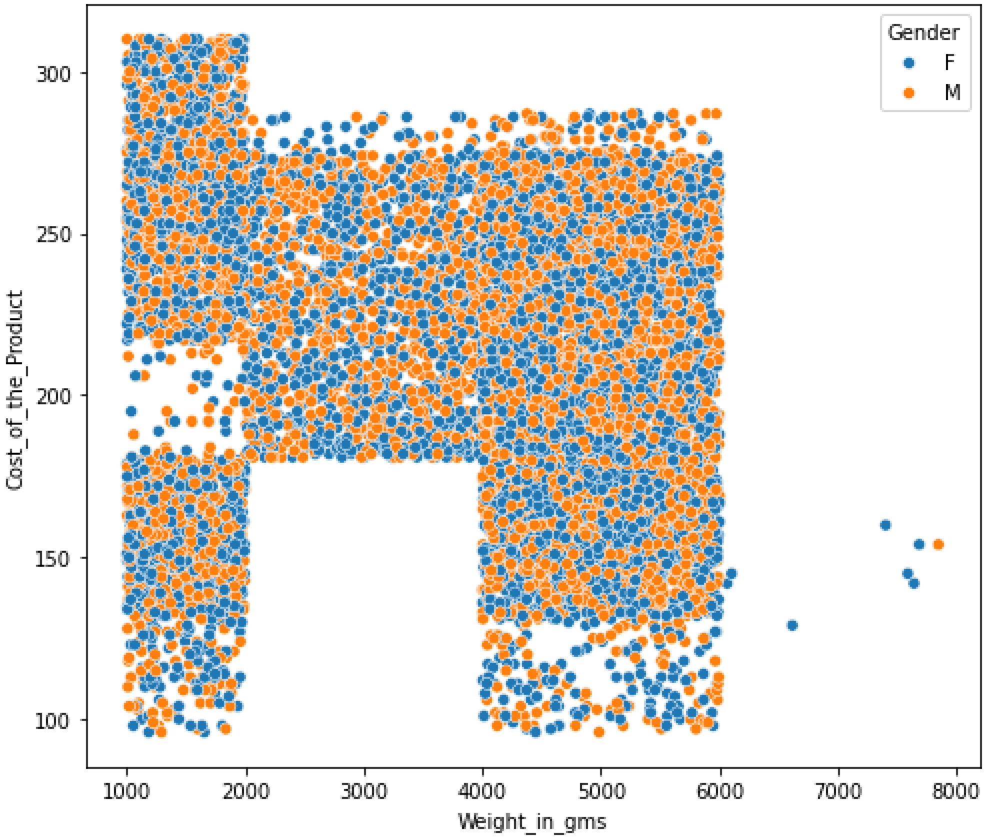
In [25]:
```
df.head()
```

Out[25]:

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

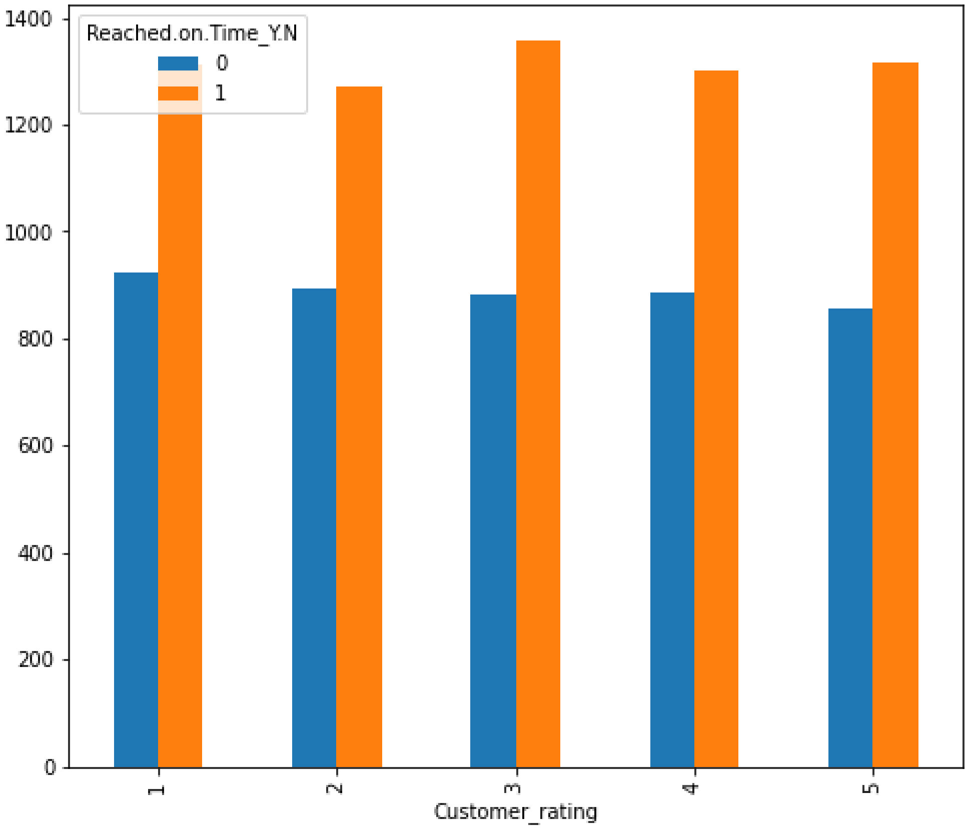
In [26]:
```
plt.figure(figsize=(8,7))
sns.scatterplot(x = df['Weight_in_gms'], y= df['Cost_of_the_Product'], hue= df['Gender']
```

Out[26]:
```
<Axes: xlabel='Weight_in_gms', ylabel='Cost_of_the_Product'>
```



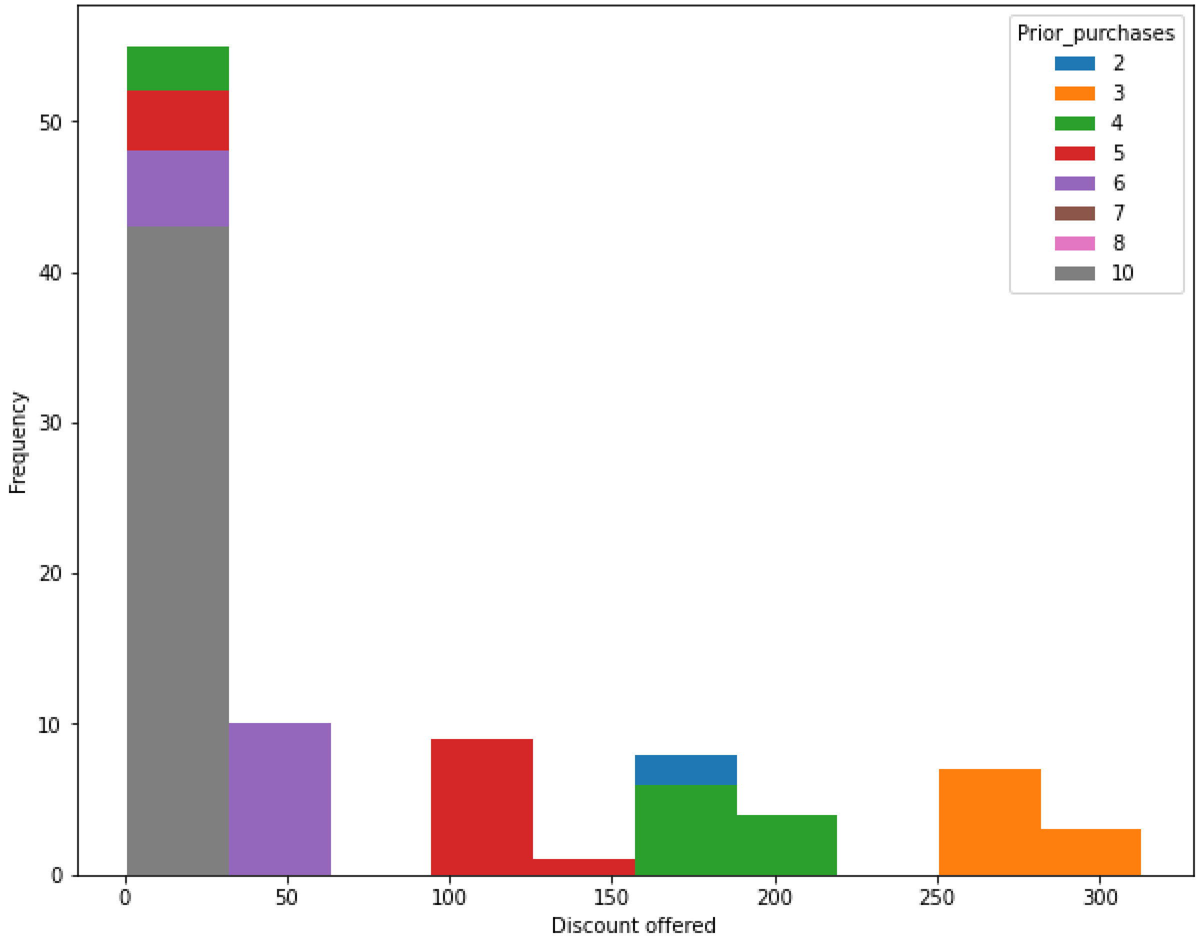
```
In [27]: df.groupby(['Customer_rating', 'Reached.on.Time_Y.N']).size().unstack().plot(kind = 'bar
```

```
Out[27]: <Axes: xlabel='Customer_rating'>
```



```
In [60]: df.groupby(['Discount_offered','Prior_purchases']).size().unstack().plot(kind = 'hist',
```

```
Out[60]: <Axes: xlabel='Discount offered', ylabel='Frequency'>
```



In [30]:
```
df.head()
```

Out[30]:

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

In [32]:
```
# Feature Engineering
```

In [33]:
```
df.head()
```

Out[33]:

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

In [34]:
```
df1 = pd.get_dummies(df['Mode_of_Shipment'], dtype = int)
```

In [35]:
```
df = pd.concat([df, df1], axis=1)
```

In [36]:
```
df.head()
```

Out[36]:

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

In [37]:
```
df = df.drop('Mode_of_Shipment', axis=1)
```

In [38]:
```
df = df.drop('ID', axis=1)
```

In [39]:
```
df['Warehouse_block'].unique()
```

Out[39]:
```
array(['D', 'F', 'A', 'B', 'C'], dtype=object)
```

In [40]:
```
df['Warehouse_block'] = df['Warehouse_block'].map({'A':0, 'B':1, 'C':2, 'D':3, 'F':4})
```

In [41]:
```
df.head()
```

Out[41]:

	Warehouse_block	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_imp
0	3	4	2	177	3	
1	4	4	5	216	2	
2	0	2	2	183	4	
3	1	3	3	176	4	
4	2	2	2	184	3	

In [42]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

df['Product_importance'] = le.fit_transform(df['Product_importance'])
df['Gender'] = le.fit_transform(df['Gender'])
```

In [43]:
```
df.head()
```

Out[43]:

	Warehouse_block	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_imp
0	3	4	2	177	3	
1	4	4	5	216	2	
2	0	2	2	183	4	
3	1	3	3	176	4	
4	2	2	2	184	3	

In [46]:
```
# correlation
```

In [47]:
```
df.corr()
```

Out[47]:

	Warehouse_block	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purc
Warehouse_block	1.000000	0.014496	0.010169	-0.006679	-0.0

Customer_care_calls	0.014496	1.000000	0.012209	0.323182	0.1
Customer_rating	0.010169	0.012209	1.000000	0.009270	0.0
Cost_of_the_Product	-0.006679	0.323182	0.009270	1.000000	0.1
Prior_purchases	-0.005262	0.180771	0.013179	0.123676	1.0
Product_importance	0.004260	0.006273	0.003157	0.006366	0.0
Gender	-0.003700	0.002545	0.002775	0.019759	-0.0
Discount_offered	0.009569	-0.130750	-0.003124	-0.138312	-0.0
Weight_in_gms	0.004086	-0.276615	-0.001897	-0.132604	-0.1
Reached.on.Time_Y.N	0.005214	-0.067126	0.013119	-0.073587	-0.0
Flight	-0.000239	0.019093	-0.002481	-0.008130	-0.0
Road	-0.000794	0.003292	0.001516	0.002531	0.0
Ship	0.000811	-0.017629	0.000765	0.004419	-0.0

In [48]:

```
# Matrix Plot
```

In [49]:

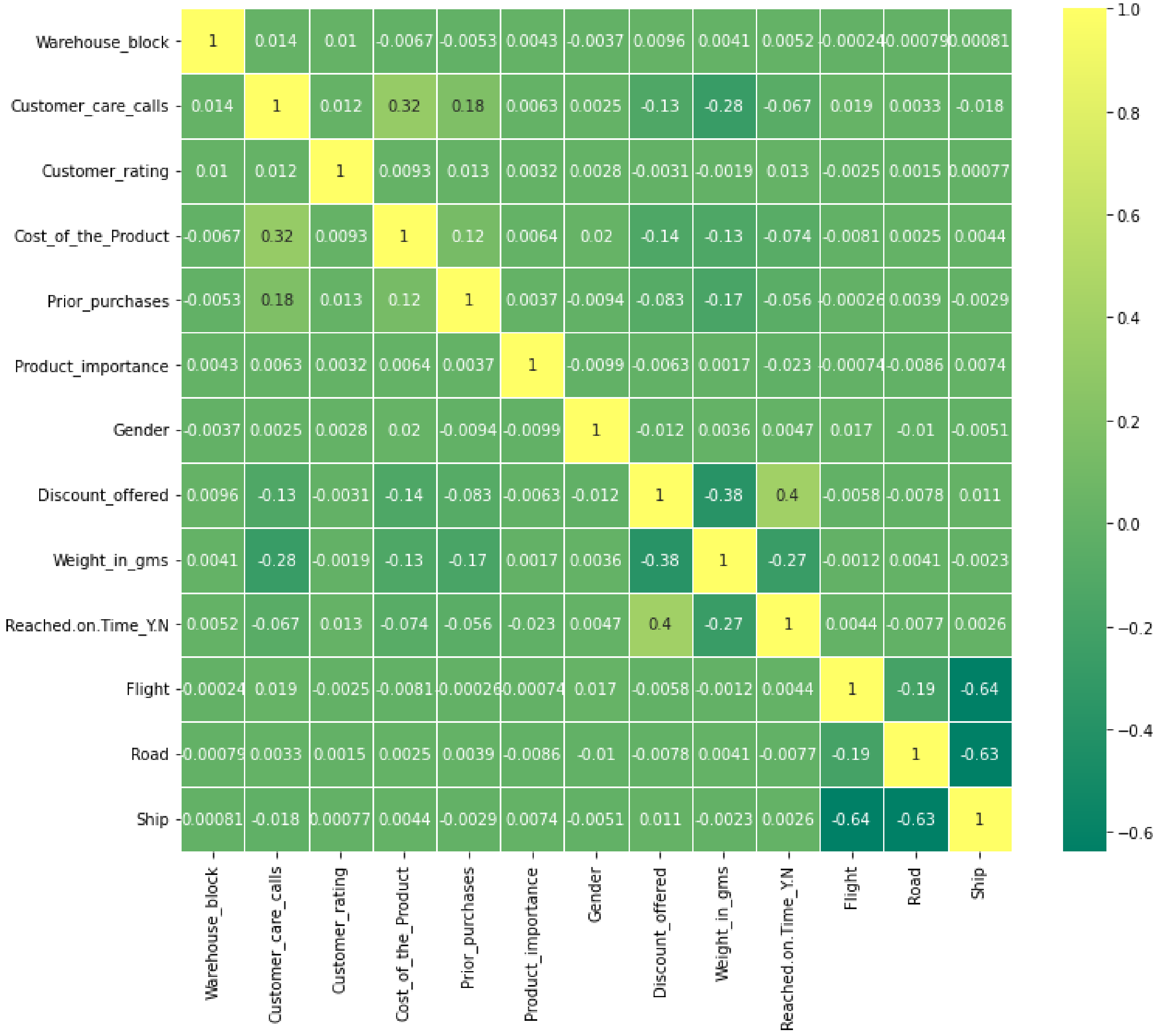
```
# Heatmap
```

In [50]:

```
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(), annot = True, linewidth = 0.5, cmap = 'summer')
```

Out[50]:

```
<Axes: >
```



In [61]:
```
# Train Test Split
```

In [62]:
```
from sklearn.model_selection import train_test_split
```

In [63]:
```
df.head()
```

Out[63]:

	Warehouse_block	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_imp
0	3	4	2	177	3	
1	4	4	5	216	2	
2	0	2	2	183	4	
3	1	3	3	176	4	
4	2	2	2	184	3	

In [64]:
```
y = df['Reached.on.Time_Y.N']
```

In [65]:
```
x = df.drop('Reached.on.Time_Y.N', axis = 1)
```

In [68]:
```
x
```

Out[68]:

	Warehouse_block	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product
0	3	4	2	177	3	
1	4	4	5	216	2	
2	0	2	2	183	4	
3	1	3	3	176	4	
4	2	2	2	184	3	
...	...	...	...	...	...	
10994	0	4	1	252	5	
10995	1	4	1	232	5	
10996	2	5	4	242	5	
10997	4	5	2	223	6	
10998	3	2	5	155	5	

10999 rows × 12 columns

In [69]:

```
y
```

Out[69]:

```
0        1
1        1
2        1
3        1
4        1
        ..
10994    1
10995    0
10996    0
10997    0
10998    0
Name: Reached.on.Time_Y.N, Length: 10999, dtype: int64
```

In [71]:

```
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size = 0.3, random_state = 4
```

In [72]:

```
print(x_train.shape, y_train.shape)
```

```
(7699, 12) (7699,)
```

In [73]:

```
print(x_test.shape, y_test.shape)
```

```
(3300, 12) (3300,)
```

In [75]:

```
# Standardization
```

In [76]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_x_train  = scaler.fit_transform(x_train)
scaled_x_test = scaler.transform(x_test)
```

In [77]:

```
scaled_x_train
```

Out[77]:

```
array([[-8.94636324e-01,  2.54637537e+00,  7.09616774e-01, ...,
        -4.41310334e-01, -4.35203552e-01,  6.89542985e-01],
       [ 1.11777263e+00, -5.87642437e-02, -1.41619628e+00, ...,
         2.26597912e+00, -4.35203552e-01, -1.45023591e+00],
       [-2.23833338e-01, -5.87642437e-02, -1.41619628e+00, ...,
        -4.41310334e-01, -4.35203552e-01,  6.89542985e-01],
       ...,
       [-1.56543931e+00, -5.87642437e-02, -7.07591927e-01, ...,
        -4.41310334e-01, -4.35203552e-01,  6.89542985e-01],
       [-1.56543931e+00,  8.09615626e-01,  1.01242341e-03, ...,
        -4.41310334e-01, -4.35203552e-01,  6.89542985e-01],
       [-2.23833338e-01, -5.87642437e-02,  1.01242341e-03, ...,
         2.26597912e+00, -4.35203552e-01, -1.45023591e+00]])
```

In [74]:
```
# Logistic Regression
```

In [78]:
```
from sklearn.linear_model import LogisticRegression
```

In [85]:
```
log = LogisticRegression()
log.fit(scaled_x_train, y_train)
y_pred  = log.predict(scaled_x_test)
```

In [86]:
```
from sklearn.metrics import r2_score, accuracy_score, confusion_matrix
```

In [87]:
```
ac = accuracy_score(y_test, y_pred)
cm =  confusion_matrix(y_test, y_pred)
r2  = r2_score(y_test, y_pred)
print('accuracy score:', ac)
print('r2 score of model:', r2)
```

```
accuracy score: 0.6348484848484849
r2 score of model: -0.5245819428767728
```

In [88]:
```
print(cm)
```

```
[[ 749  563]
 [ 642 1346]]
```

In [89]:
```
# Random Forest Classifier
```

In [90]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300, 400, 500, 800, 1000],
    'max_depth': [None, 5,10, 15,20],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1,3, 5,8,10]
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, ve
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
```

```
Fitting 5 folds for each of 875 candidates, totalling 4375 fits
```

In [93]:
```
grid_search.best_estimator_
```

Out[93]:

```
□                         RandomForestClassifier
RandomForestClassifier(max_depth=5, min_samples_split=15, n_estimators=200,
                       random_state=42)
```

In [94]:
```
rfc = RandomForestClassifier(max_depth=5, min_samples_split=15, n_estimators=200, random
rfc.fit(x_train, y_train)
y_pred = rfc.predict(x_test)
```

In [95]:
```
acc = accuracy_score(y_test,y_pred)
r2 = r2_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
```

```
print('accuracy score :', acc)
print('r2_score:',r2)
```

```
accuracy score : 0.6827272727272727
r2_score: -0.3246782524414782
```

In [96]:

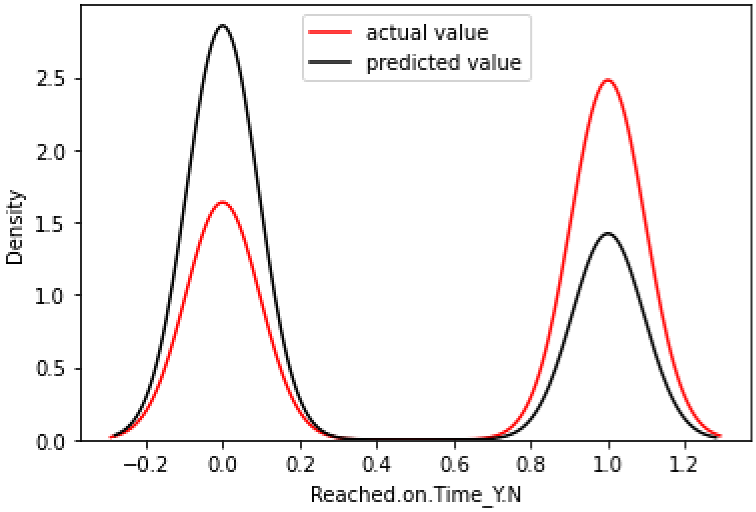
```
print(cm)
```

```
[[1234   78]
 [ 969 1019]]
```

In [97]:

```
sns.distplot(y_test, label = 'actual value', color = 'red', hist = False)
sns.distplot(y_pred, label = 'predicted value', color= 'black', hist = False)
plt.legend()
plt.show()
```

```
C:\Users\amits\AppData\Local\Temp/ipykernel_23648/2942436175.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test, label = 'actual value', color = 'red', hist = False)
C:\Users\amits\AppData\Local\Temp/ipykernel_23648/2942436175.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_pred, label = 'predicted value', color= 'black', hist = False)
```



In [98]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()

param_grid = {
    'max_depth': [None, 5,10, 15,20],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1,3, 5,8,10, 15]

}
```

```
grid_search = GridSearchCV(dtree, param_grid, cv=5, verbose=1, n_jobs = -1, scoring='neg

grid_search.fit(x_train, y_train)

grid_search.best_params_
```

```
Fitting 5 folds for each of 150 candidates, totalling 750 fits
```

Out[98]:

```
{'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

In [101…

```
grid_search.best_estimator_
```

Out[101]:

```
□       DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

In [102…

```
dtree = DecisionTreeClassifier(max_depth=5, min_samples_leaf=1, min_samples_split= 2 )
dtree.fit(x_train,y_train)
y_pred = dtree.predict(x_test)
```

In [103…

```
acc = accuracy_score(y_test,y_pred)
r2 = r2_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print('accuracy score :', acc)
print('r2_score:',r2)
```

```
accuracy score : 0.6848484848484848
r2_score: -0.31582175982725613
```

In [104…

```
print(cm)
```

```
[[1203  109]
 [ 931 1057]]
```

In [106…

```
from xgboost import XGBClassifier
xgb = XGBClassifier()

param_grid = {
    'n_estimators': [100, 200, 300, 500, 700, 800,1000],
    'max_depth': [3, 4, 5, 8, 10, 15],
    'learning_rate': [0.1, 0.01, 0.001]
}
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5, scoring='accuracy
grid_search.fit(x_train, y_train)

grid_search.best_params_
```

```
Fitting 5 folds for each of 126 candidates, totalling 630 fits
```

```
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:312: FutureWarning: is_sparse
is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Spars
eDtype)` instead.
  if is_sparse(dtype):
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:314: FutureWarning: is_catego
rical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype,
CategoricalDtype) instead
  elif is_categorical_dtype(dtype) and enable_categorical:
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:345: FutureWarning: is_catego
rical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype,
CategoricalDtype) instead
  if is_categorical_dtype(dtype)
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:336: FutureWarning: is_catego
rical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype,
CategoricalDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
```

```
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:440: FutureWarning: is_sparse
is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Spars
eDtype)` instead.
  if is_sparse(data):
```

Out[106]:

```
{'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}
```

In [107…

```
grid_search.best_estimator_
```

Out[107]:

□ XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [108…

```
xgb = XGBClassifier(learning_rate = 0.01, max_depth = 3, n_estimators = 100 )
xgb.fit(x_train,y_train)
y_pred = xgb.predict(x_test)
```

```
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:312: FutureWarning: is_sparse
is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Spars
eDtype)` instead.
  if is_sparse(dtype):
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:314: FutureWarning: is_catego
rical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype,
CategoricalDtype) instead
  elif is_categorical_dtype(dtype) and enable_categorical:
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:345: FutureWarning: is_catego
rical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype,
CategoricalDtype) instead
  if is_categorical_dtype(dtype)
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:336: FutureWarning: is_catego
rical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype,
CategoricalDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:440: FutureWarning: is_sparse
is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Spars
eDtype)` instead.
  if is_sparse(data):
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:312: FutureWarning: is_sparse
is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Spars
eDtype)` instead.
  if is_sparse(dtype):
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:314: FutureWarning: is_catego
rical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype,
CategoricalDtype) instead
  elif is_categorical_dtype(dtype) and enable_categorical:
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:345: FutureWarning: is_catego
rical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype,
CategoricalDtype) instead
  if is_categorical_dtype(dtype)
C:\Users\amits\anaconda3\lib\site-packages\xgboost\data.py:336: FutureWarning: is_catego
rical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype,
CategoricalDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
```

```
In [ ]:
```