E-COMMERCE SHIPPING BUSINESS

Final Project

ABSTRACT

Businesses are increasingly using sophisticated machine-learning techniques to extract insightful data from their customer databases in the dynamic world of global e-commerce.

For a business that specializes in electronic products and wants to comprehend consumer behavior and maximize operational elements like shipment effectiveness and customer satisfaction, this project is especially important.

The main goal is to use data-driven approaches to find important patterns and trends that can guide organizational strategy decisions.

Duong Thuy Le (RUID 220004531)

Big Data Analytics

E-Commerce Shipping Busines Report

December 4, 2023

Table of Content:

1 Introduction:

Businesses are increasingly using sophisticated machine-learning techniques to extract insightful data from their customer databases in the dynamic world of global e-commerce. For a business that specializes in electronic products and wants to comprehend consumer behavior and maximize operational elements like shipment effectiveness and customer satisfaction, this project is especially important.

This study examines a large dataset with 10,999 observations for 12 different variables. These variables cover a wide range of data, from shipment details to customer demographics. The main goal is to use data-driven approaches to find important patterns and trends that can guide organizational strategy decisions.

2 Business Overview:

Our subject company operates on a global scale, boasting a substantial warehousing infrastructure divided into distinct blocks (A, B, C, D, E). The modes of shipment—Ship, Flight, and Road—highlight the diversity in logistics, catering to the unique demands of a dynamic market. Customer care calls serve as a vital metric, offering insights into customer inquiries and the effectiveness of query resolutions.

Customer satisfaction, a cornerstone of business success, is evaluated through customer ratings ranging from 1 (lowest) to 5 (highest). The cost of the product, prior purchase history, product importance categorization (low, medium, high), and gender distribution further contribute to the multifaceted nature of our analysis.

Additionally, the dataset sheds light on the discount structures offered by the company, the weight of the products in grams, and a critical metric—the timely delivery of products, represented by the binary variable 'Reached on Time' (1 indicating a delay, 0 indicating on-time delivery).

As we embark on this data-driven journey, our aim is to not only scrutinize customer-centric factors but also to harness the power of machine learning to predict the timeliness of product deliveries. Through this exploration, we endeavor to equip our client with actionable insights, fostering a more agile and customer-focused business model.

The dataset used for model building contained 10999 observations of 12 variables. The data contains the following information:

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.

3 Data Preprocessing:

3.1 Import the needed package:

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

[2]:
```python
data = pd.read_csv('train.csv')
data.head(5)
```

[2]:
```
   ID Warehouse_block Mode_of_Shipment  Customer_care_calls  Customer_rating  \
0   1               D           Flight                    4                2
1   2               F           Flight                    4                5
2   3               A           Flight                    2                2
3   4               B           Flight                    3                3
4   5               C           Flight                    2                2

   Cost_of_the_Product  Prior_purchases Product_importance Gender  \
0                  177                3                low      F
1                  216                2                low      M
2                  183                4                low      M
3                  176                4             medium      M
4                  184                3             medium      F

   Discount_offered  Weight_in_gms  Reached.on.Time_Y.N
0                44           1233                    1
1                59           3088                    1
2                48           3374                    1
3                10           1177                    1
4                46           2484                    1
```

[3]:
```python
data.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
```

[4]:
```
print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))
```

```
Number of instances = 10999
Number of attributes = 12
```

3.2 Check missing values:

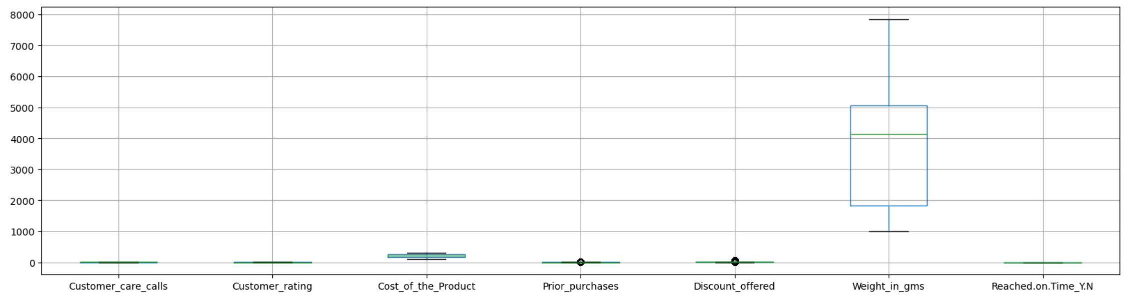
[5]:
```
print('Number of missing values:')
for col in data.columns:
    print('\t%s: %d' % (col,data[col].isna().sum()))
```

```
Number of missing values:
        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
```

3.3 Check outliners:

[6]:
```
data2 = data.
 ↪drop(['ID','Warehouse_block','Mode_of_Shipment','Product_importance','Gender'],axis=1)
data2.boxplot(figsize=(20,5))
```

[6]: <AxesSubplot: >



[7]:
```
Z = (data2-data2.mean())/data2.std()    # standardization --> (data-mean)/
 ↪(standard deviation)
Z[20:25]
```

[7]:
```
    Customer_care_calls  Customer_rating  Cost_of_the_Product  \
20            -0.923757         0.006689            -1.023585
21            -0.923757        -1.408135             0.453635
22            -1.799806         1.421513            -1.127614
23            -0.047709         0.006689             0.016711
24            -0.047709         1.421513             0.848947

    Prior_purchases  Discount_offered  Weight_in_gms  Reached.on.Time_Y.N
20        -1.029377          1.519653      -1.292067               0.8221
21         0.283941          2.321849      -0.449448               0.8221
22        -1.029377         -0.701811      -1.152038               0.8221
23        -0.372718         -0.084737       0.176096               0.8221
24        -1.029377          0.902580      -0.044648               0.8221
```

[8]:
```
print('Number of rows before discarding outliers = %d' % (Z.shape[0]))

Z2 = Z.loc[((Z > -3).sum(axis=1)==9) & ((Z <= 3).sum(axis=1)==9),:]    # only␣
 ↪take the rows (there are 9 of them)
                                                                        # whose␣
 ↪all attribute values are greater than -3
                                                                        # and␣
 ↪whose all attribute values are less than 3
print('Number of rows after discarding missing values = %d' % (Z2.shape[0]))
```

```
Number of rows before discarding outliers = 10999
Number of rows after discarding missing values = 0
```

[9]:
```
dups = data.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))   # counts the number of␣
 ↪True's
```

```
Number of duplicate rows = 0
```

Overall, it can be seen that the dataset has been cleaned and there is no outliner, no missing value, and no duplicated row. We can say that the quality of dataset is high and trustworthy.

4 Exploratory Data Analysis (EDA):

[10]:
```
data3 = data.drop('ID', axis = 1)
```

[11]:
```
data_des = data3.describe()
round(data_des,2)
```

[11]:

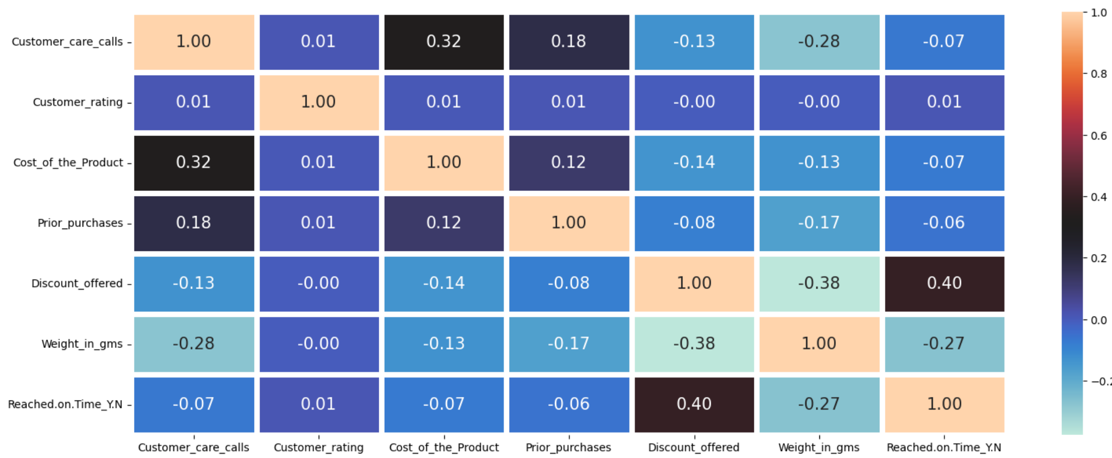
	Customer_care_calls	Customer_rating	Cost_of_the_Product	\
count	10999.00	10999.00	10999.00	
mean	4.05	2.99	210.20	
std	1.14	1.41	48.06	
min	2.00	1.00	96.00	
25%	3.00	2.00	169.00	
50%	4.00	3.00	214.00	
75%	5.00	4.00	251.00	
max	7.00	5.00	310.00	

	Prior_purchases	Discount_offered	Weight_in_gms	Reached.on.Time_Y.N
count	10999.00	10999.00	10999.00	10999.00
mean	3.57	13.37	3634.02	0.60
std	1.52	16.21	1635.38	0.49
min	2.00	1.00	1001.00	0.00
25%	3.00	4.00	1839.50	0.00
50%	3.00	7.00	4149.00	1.00
75%	4.00	10.00	5050.00	1.00
max	10.00	65.00	7846.00	1.00

4.1 Checking the correlation between the features and target column:

[12]:
```
plt.figure(figsize = (18, 7))
sns.heatmap(data3.corr(), annot = True, fmt = '0.2f', annot_kws = {'size' :␣
↪15}, linewidth = 5, cmap = 'icefire')
plt.show()
```

```
/var/folders/x8/vmspvpd557j0gjkqkwzhf08m0000gn/T/ipykernel_29922/2198535553.py:2
: FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only valid
columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(data3.corr(), annot = True, fmt = '0.2f', annot_kws = {'size' :
15}, linewidth = 5, cmap = 'icefire')
```



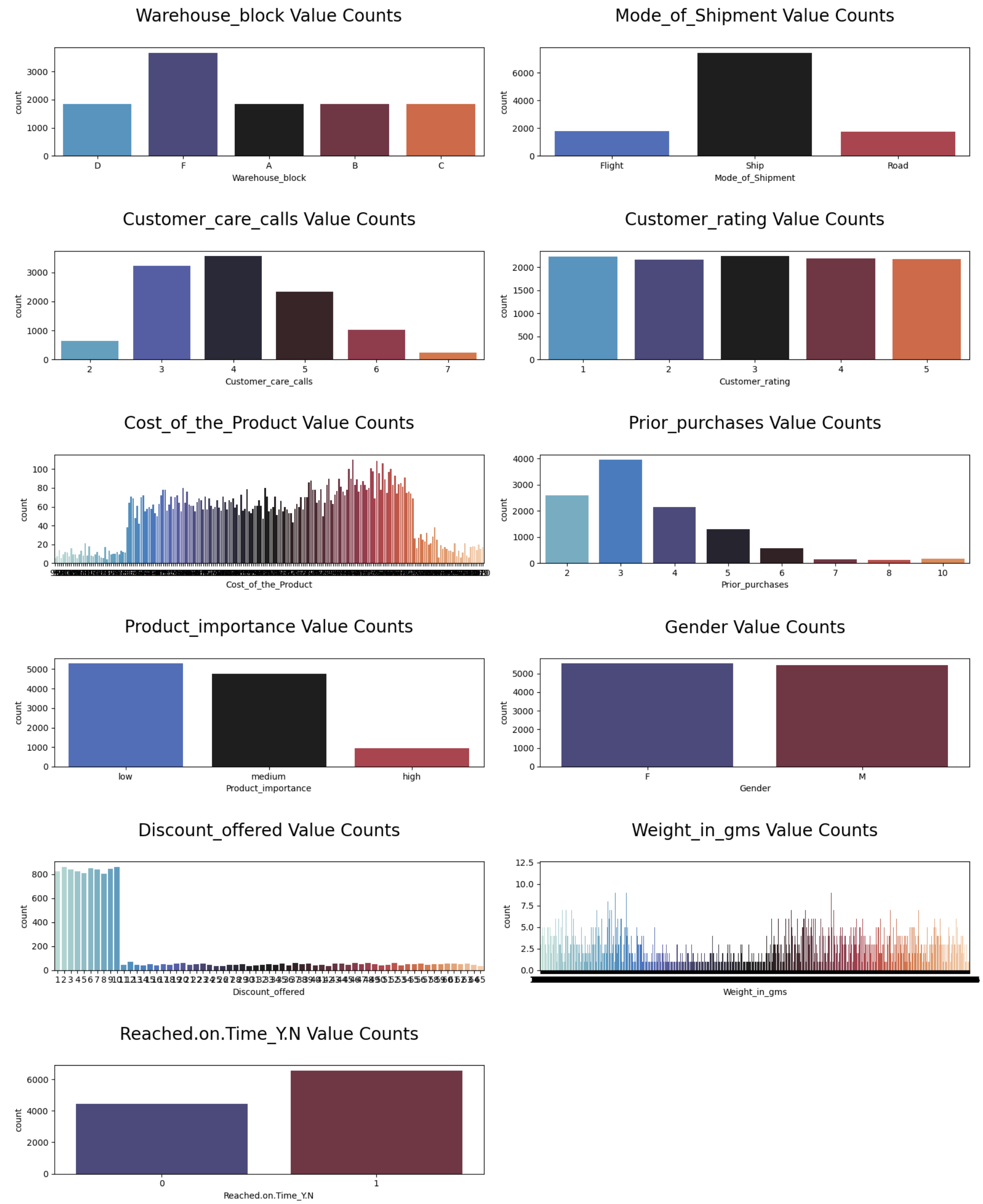
From Correlation matrix :

1) **Discount Offered** and **Weights in grams** have the 1st strongest negative correlation of -38%.
2) **Customer care calls** and **Weights in grams** have the 2nd strongest negative correlation with -28%.
3) **Weights in gram** and **Reached on Time or Not** have the 3rd strongest negative correlation of -27%.
4) **Discount Offered** and **Reached on Time or Not** have the 1st strongest positive correlation of 40%.
5) **Customer care calls** and **Cost of the product** have the 2nd strongest positive correlation of 32%.
6) **Customer care calls** and **Prior purchase** have the 3rd strongest positive correlation of 18%.

4.2 Checking the patterns in each variable:

[13]:
```
plt.figure(figsize=(16, 20))
plotnumber = 1

for i in range(len(data3.columns)):
    if plotnumber <= 11:
        ax = plt.subplot(6, 2, plotnumber)
        sns.countplot(x=data3.columns[i], data=data3, ax=ax, palette='icefire')
        plt.title(f"\n{data3.columns[i]} Value Counts\n", fontsize=20)

    plotnumber += 1

plt.tight_layout()
```

```
plt.show()
```



Based on the patterns:

1) Warehouse block F has high usage times than all other Warehouse blocks.
2) Ship delivers is the most common shipment of products to the customers.

3) Most of the customers calls 4 times to the customer care centers.
4) Most of the customers have 3 prior purchases, which is the good signal in brand's loyalty.
5) Most of the products are of low importance.
6) Gender, Customer Ratings doesn't have much variance.
7) It is more likely that products are not shipped on time.
8) Customers have a tendency to buy discount-offer products, with the most frequency from 1 to 10 products.

4.3 Ratio of each variable:

4.3.1 Warehouse distribution:

[14]:
```
warehouse = data3['Warehouse_block'].value_counts().reset_index().
↪rename(columns ={'index':'Warehouse','Warehouse_block':'Counts'})
warehouse
```

[14]:

	Warehouse	Counts
0	F	3666
1	D	1834
2	A	1833
3	B	1833
4	C	1833

[15]:
```
fig = px.pie(warehouse, names = 'Warehouse', values = 'Counts',
             color='Warehouse', width = 650, height = 400,
             hole = 0.5)
fig.update_traces(textinfo = 'percent+label')
```

[16]:
```
# Calculate counts
warehouse_counts = data3.groupby(['Warehouse_block', 'Reached.on.Time_Y.N']).
↪size().unstack(fill_value=0)

# Reset the index
warehouse_counts = warehouse_counts.reset_index()

# Create a stacked bar chart using Plotly Express
fig = px.bar(warehouse_counts, x='Warehouse_block', y=[0, 1],
             labels={'value': 'Frequency', 'variable': 'Reached.on.Time_Y.N'},
             title='Warehouse Performance - Reached Time Y/N',
             category_orders={'Warehouse_block': ['A', 'B', 'C', 'D', 'F']},
             barmode='stack')

# Show the plot
fig.show()
```

[17]:
```
warehouse_counts['Percentage_0'] = (warehouse_counts[0] / (warehouse_counts[0]␣
↪+ warehouse_counts[1])) * 100
```

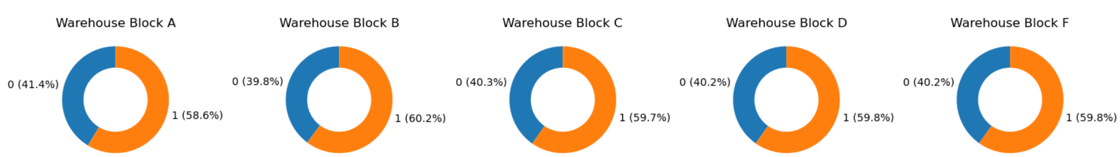
```
warehouse_counts['Percentage_1'] = (warehouse_counts[1] / (warehouse_counts[0]␣
↪+ warehouse_counts[1])) * 100
warehouse_counts
```

[17]:

Reached.on.Time_Y.N	Warehouse_block	0	1	Percentage_0	Percentage_1
0	A	758	1075	41.352973	58.647027
1	B	729	1104	39.770867	60.229133
2	C	739	1094	40.316421	59.683579
3	D	738	1096	40.239913	59.760087
4	F	1472	2194	40.152755	59.847245

[18]:

```
# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(warehouse_counts), figsize=(15, 5))

# Plotting side-by-side pie charts for each mode of shipment
for ax, (index, row) in zip(axes, warehouse_counts.iterrows()):
    ax.pie([row['Percentage_0'], row['Percentage_1']],
           labels=[f"0 ({row['Percentage_0']:.1f}%)", f"1 ({row['Percentage_1']:
↪.1f}%)"],
           startangle=90, autopct='', wedgeprops=dict(width=0.4))

    # Adding a title
    ax.set_title(f'Warehouse Block {row["Warehouse_block"]}')

# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the plot
plt.show()
```



The F block had a greater quantity of stored products than the other blocks. The remaining blocks have roughly equal quantities of stored products.

4.3.2 Shipment distribution:

[19]:

```
shipment = data3['Mode_of_Shipment'].value_counts().reset_index().
↪rename(columns ={'index':'Mode_of_Shipment','Mode_of_Shipment':'Counts'})
shipment
```

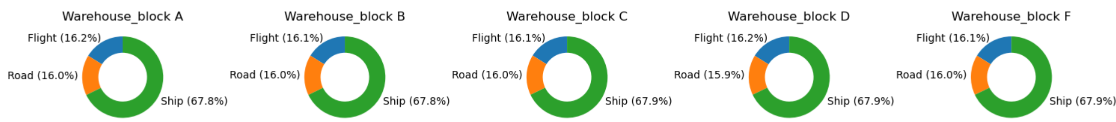
[19]:   Mode_of_Shipment Counts
```
  Mode_of_Shipment  Counts
0             Ship    7462
1           Flight    1777
2             Road    1760
```

[20]:
```
fig = px.pie(shipment, names = 'Mode_of_Shipment', values = 'Counts',
             color='Mode_of_Shipment', width = 650, height = 400,
             hole = 0.5)
fig.update_traces(textinfo = 'percent+label')
```

[21]:
```
# Calculate counts
warehouse_counts2 = data3.groupby(['Warehouse_block', 'Mode_of_Shipment']).
 ↪size().unstack(fill_value=0)

# Reset the index
warehouse_counts2 = warehouse_counts2.reset_index()

# Create a stacked bar chart using Plotly Express
fig = px.bar(warehouse_counts2, x='Warehouse_block', y=['Flight', 'Road',␣
 ↪'Ship'],

             labels={'value': 'Frequency', 'variable': 'Mode_of_Shipment'},
             title='Warehouse Performance - Mode of Shipment',
             category_orders={'Warehouse_block': ['A', 'B', 'C', 'D', 'F']},
             barmode='stack')

# Show the plot
fig.show()
```

[22]:
```
# Calculate the total counts for each warehouse block
warehouse_counts2['Total'] = warehouse_counts2[['Flight', 'Road', 'Ship']].
 ↪sum(axis=1)

# Calculate the percentages
warehouse_counts2['Percentage_Flight'] = (warehouse_counts2['Flight'] /␣
 ↪warehouse_counts2['Total']) * 100
warehouse_counts2['Percentage_Road'] = (warehouse_counts2['Road'] /␣
 ↪warehouse_counts2['Total']) * 100
warehouse_counts2['Percentage_Ship'] = (warehouse_counts2['Ship'] /␣
 ↪warehouse_counts2['Total']) * 100

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(warehouse_counts2), figsize=(15, 5))

# Plotting side-by-side pie charts for each mode of shipment
for ax, (index, row) in zip(axes, warehouse_counts2.iterrows()):
```

```
    ax.pie([row['Percentage_Flight'], row['Percentage_Road'], row['Percentage_Ship']],
           labels=[f"Flight ({row['Percentage_Flight']:.1f}%)", f"Road ({row['Percentage_Road']:.1f}%)",f"Ship ({row['Percentage_Ship']:.1f}%)"],
           startangle=90, autopct='', wedgeprops=dict(width=0.4))

    # Adding a title
    ax.set_title(f'Warehouse_block {row["Warehouse_block"]}')

# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the plot
plt.show()
```

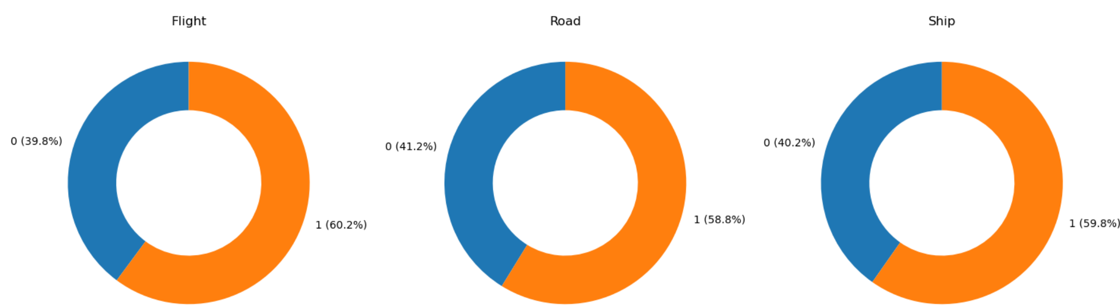


[23]:
```
# Calculate counts
shipment_perf = data3.groupby(['Mode_of_Shipment', 'Reached.on.Time_Y.N']).size().unstack(fill_value=0)

# Reset the index
shipment_perf = shipment_perf.reset_index()

# Calculate the total counts for each warehouse block
shipment_perf['Total'] = shipment_perf[[0,1]].sum(axis=1)

# Calculate the percentages
shipment_perf['Percentage_0'] = (shipment_perf[0] / shipment_perf['Total']) * 100
shipment_perf['Percentage_1'] = (shipment_perf[1] / shipment_perf['Total']) * 100
shipment_perf
```

[23]:
```
Reached.on.Time_Y.N Mode_of_Shipment     0     1  Total  Percentage_0  \
0                             Flight   708  1069   1777     39.842431
1                               Road   725  1035   1760     41.193182
2                               Ship  3003  4459   7462     40.243902

Reached.on.Time_Y.N  Percentage_1
0                       60.157569
1                       58.806818
```

```
2                          59.756098
```

[24]:
```
# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(shipment_perf), figsize=(15, 5))

# Plotting side-by-side pie charts for each mode of shipment
for ax, (index, row) in zip(axes, shipment_perf.iterrows()):
    ax.pie([row['Percentage_0'], row['Percentage_1']],
           labels=[f"0 ({row['Percentage_0']:.1f}%)", f"1 ({row['Percentage_1']:
↪.1f}%)"],
           startangle=90, autopct='', wedgeprops=dict(width=0.4))

    # Adding a title
    ax.set_title(f'{row["Mode_of_Shipment"]}')

# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the plot
plt.show()
```



The mode of shipment that was most commonly used to send the products was "Ship", with over 7000 shipments sent through it, wheresas Flight mode had a higher percentage of products that reached their destination on time compared to Ship and Road modes.

4.3.3 Gender:

[25]:
```
gender = data3['Gender'].value_counts().reset_index().rename(columns ={'index':
↪'Gender','Gender':'Counts'})
gender
```

[25]:

	Gender	Counts
0	F	5545
1	M	5454

[26]:
```
# Calculate counts
gender_counts = data3.groupby(['Gender', 'Customer_care_calls']).size().
↪unstack(fill_value=0)

# Reset the index
gender_counts = gender_counts.reset_index()

# Create a stacked bar chart using Plotly Express
fig = px.bar(gender_counts, x='Gender', y=gender_counts.columns[1:],
             labels={'value': 'Frequency', 'variable': 'Customer_care_calls'},
             title='Numbers of calls by Gender',
             category_orders={'Gender': ['F', 'M']},
             barmode='stack')

# Show the plot
fig.show()
```

[27]:
```
# Calculate counts
gender_counts2 = data3.groupby(['Gender', 'Prior_purchases']).size().
↪unstack(fill_value=0)

# Reset the index
gender_counts2 = gender_counts2.reset_index()

# Create a stacked bar chart using Plotly Express
fig = px.bar(gender_counts2, x='Gender', y=gender_counts2.columns[1:],
             labels={'value': 'Frequency', 'variable': 'Prior_purchases'},
             title='Numbers of calls by Gender',
             category_orders={'Gender': ['F', 'M']},
             barmode='stack')

# Show the plot
fig.show()
```

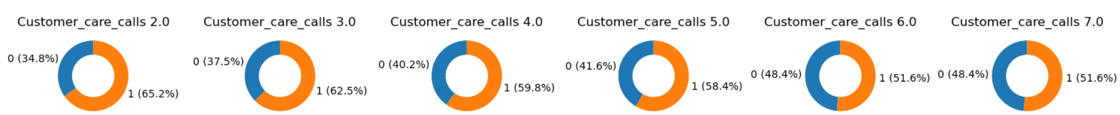
4.3.4 Customer Care Calls:

[28]:
```
# Calculate counts
customer_service = data3.groupby(['Customer_care_calls', 'Reached.on.Time_Y.
↪N']).size().unstack(fill_value=0)

# Reset the index
customer_service = customer_service.reset_index()

# Calculate the total counts for each warehouse block
customer_service['Total'] = customer_service[[0,1]].sum(axis=1)
```

```
customer_service['Percentage_0'] = (customer_service[0] /␣
 ↪customer_service['Total']) * 100
customer_service['Percentage_1'] = (customer_service[1] /␣
 ↪customer_service['Total']) * 100
customer_service
```

[28]:
```
Reached.on.Time_Y.N  Customer_care_calls     0     1  Total  Percentage_0  \
0                                      2   222   416    638     34.796238
1                                      3  1206  2011   3217     37.488343
2                                      4  1431  2126   3557     40.230531
3                                      5   968  1360   2328     41.580756
4                                      6   490   523   1013     48.371175
5                                      7   119   127    246     48.373984

Reached.on.Time_Y.N  Percentage_1
0                        65.203762
1                        62.511657
2                        59.769469
3                        58.419244
4                        51.628825
5                        51.626016
```

[29]:
```
# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(customer_service), figsize=(15, 5))

# Plotting side-by-side pie charts for each mode of shipment
for ax, (index, row) in zip(axes, customer_service.iterrows()):
    ax.pie([row['Percentage_0'], row['Percentage_1']],
           labels=[f"0 ({row['Percentage_0']:.1f}%)", f"1 ({row['Percentage_1']:
 ↪.1f}%)"],
           startangle=90, autopct='', wedgeprops=dict(width=0.4))

    # Adding a title
    ax.set_title(f'Customer_care_calls {row["Customer_care_calls"]}')

# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the plot
plt.show()
```



Customers calls were more when the product doesn't reach on time and when the product reaches at time then the calls were less.

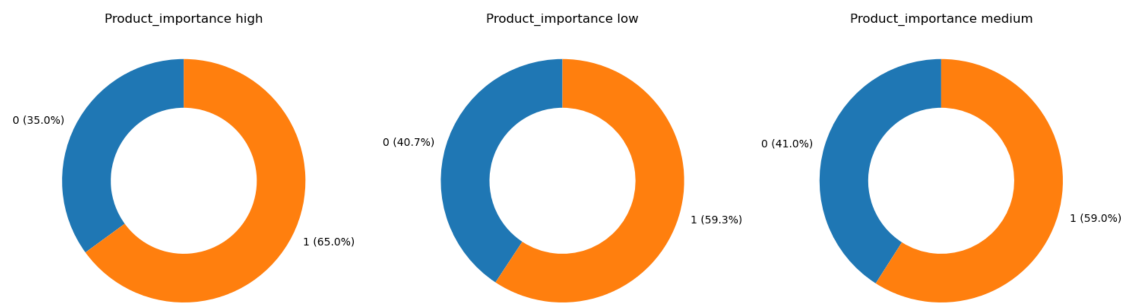
4.3.5 Product importance affection:

[30]:
```
# Calculate counts
important_service = data3.groupby(['Product_importance', 'Reached.on.Time_Y.
↪N']).size().unstack(fill_value=0)

# Reset the index
important_service = important_service.reset_index()

# Calculate the total counts for each warehouse block
important_service['Total'] = important_service[[0,1]].sum(axis=1)

important_service['Percentage_0'] = (important_service[0] /␣
↪important_service['Total']) * 100
important_service['Percentage_1'] = (important_service[1] /␣
↪important_service['Total']) * 100
important_service
```

[30]:
```
Reached.on.Time_Y.N Product_importance     0     1  Total  Percentage_0  \
0                                 high   332   616    948     35.021097
1                                  low  2157  3140   5297     40.721163
2                               medium  1947  2807   4754     40.954985

Reached.on.Time_Y.N  Percentage_1
0                       64.978903
1                       59.278837
2                       59.045015
```

[31]:
```
# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(important_service), figsize=(15, 5))

# Plotting side-by-side pie charts for each mode of shipment
for ax, (index, row) in zip(axes, important_service.iterrows()):
    ax.pie([row['Percentage_0'], row['Percentage_1']],
           labels=[f"0 ({row['Percentage_0']:.1f}%)", f"1 ({row['Percentage_1']:
↪.1f}%)"],
           startangle=90, autopct='', wedgeprops=dict(width=0.4))

    # Adding a title
    ax.set_title(f'Product_importance {row["Product_importance"]}')

# Adjust layout to prevent overlapping
plt.tight_layout()
```

```python
# Display the plot
plt.show()
```



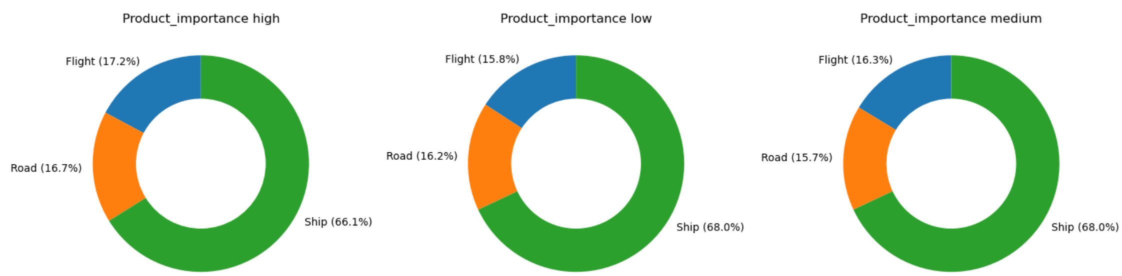
[32]:
```python
# Calculate counts
important_shipment = data3.groupby(['Product_importance', 'Mode_of_Shipment']).
↪size().unstack(fill_value=0)

# Reset the index
important_shipment = important_shipment.reset_index()
important_shipment
```

[32]:

Mode_of_Shipment	Product_importance	Flight	Road	Ship
0	high	163	158	627
1	low	838	857	3602
2	medium	776	745	3233

[33]:
```python
# Calculate the total counts for each warehouse block
important_shipment['Total'] = important_shipment[['Flight', 'Road', 'Ship']].
↪sum(axis=1)

# Calculate the percentages
important_shipment['Percentage_Flight'] = (important_shipment['Flight'] /␣
↪important_shipment['Total']) * 100
important_shipment['Percentage_Road'] = (important_shipment['Road'] /␣
↪important_shipment['Total']) * 100
important_shipment['Percentage_Ship'] = (important_shipment['Ship'] /␣
↪important_shipment['Total']) * 100

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=len(important_shipment), figsize=(15,␣
↪5))

# Plotting side-by-side pie charts for each mode of shipment
for ax, (index, row) in zip(axes, important_shipment.iterrows()):
```

```
    ax.pie([row['Percentage_Flight'], row['Percentage_Road'],␣
↪row['Percentage_Ship']],
           labels=[f"Flight ({row['Percentage_Flight']:.1f}%)", f"Road␣
↪({row['Percentage_Road']:.1f}%)",f"Ship ({row['Percentage_Ship']:.1f}%)"],
           startangle=90, autopct='', wedgeprops=dict(width=0.4))

    # Adding a title
    ax.set_title(f'Product_importance {row["Product_importance"]}')

# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the plot
plt.show()
```



5 Machine Learning Model Building:

5.1 Label Encoding:

[34]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# Encode the categorical variables
for col in data3.columns:
    if data3[col].dtype == 'object':
        data3[col] = le.fit_transform(data3[col])
```

[35]:
```
data3.apply(lambda x: x.unique())
```

[35]:
```
Warehouse_block                                          [3, 4, 0, 1, 2]
Mode_of_Shipment                                               [0, 2, 1]
Customer_care_calls                                   [4, 2, 3, 5, 6, 7]
Customer_rating                                          [2, 5, 3, 1, 4]
Cost_of_the_Product     [177, 216, 183, 176, 184, 162, 250, 233, 150, …
```

```
Prior_purchases                                        [3, 2, 4, 6, 5, 7, 10, 8]
Product_importance                                                    [1, 2, 0]
Gender                                                                   [0, 1]
Discount_offered        [44, 59, 48, 10, 46, 12, 3, 11, 29, 32, 1, 43,…
Weight_in_gms           [1233, 3088, 3374, 1177, 2484, 1417, 2371, 280…
Reached.on.Time_Y.N                                                      [1, 0]
dtype: object
```

[36]:

```
Y = data3['Reached.on.Time_Y.N']
X = data3.drop(['Reached.on.Time_Y.N'],axis=1)

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.7,␣
↪random_state=1)
```

5.2 Decision tree classifier:

Using gini index as the impurity measure with a maximum depth of 3.

5.2.1 Plot the resulting decision tree obtained after training the classifier:

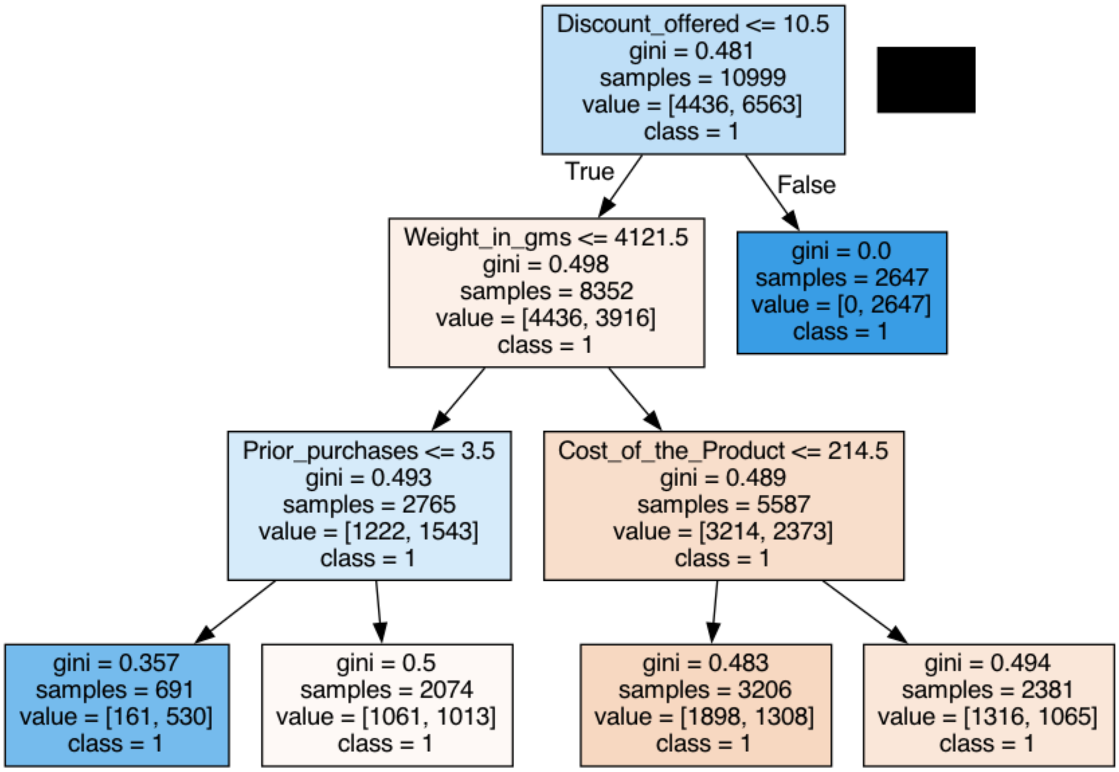
[37]:

```
from sklearn import tree

clf = tree.DecisionTreeClassifier(criterion='gini',max_depth=3).fit(X, Y)
```

[38]:

```
import pydotplus
from IPython.display import Image

class_names = data3['Reached.on.Time_Y.N'].astype(str)

dot_data = tree.export_graphviz(clf, feature_names=X.columns,␣
↪class_names=class_names, filled=True,
                                out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

[38]:



5.2.2 Compute the accuracy on the test set:

[39]:
```
Y_predict = clf.predict(X_test)

from sklearn.metrics import accuracy_score

print('Accuracy on test data is %.2f' % (accuracy_score(Y_test, Y_predict)))
```

```
Accuracy on test data is 0.68
```

[62]:
```
Y_predict1 = clf.predict(X_train)
df = pd.DataFrame({'Method': 'Decision Tree', 'Train Accuracy':␣
↪[accuracy_score(Y_train, Y_predict1)], 'Test Accuracy':␣
↪[accuracy_score(Y_test, Y_predict)]})
df
```

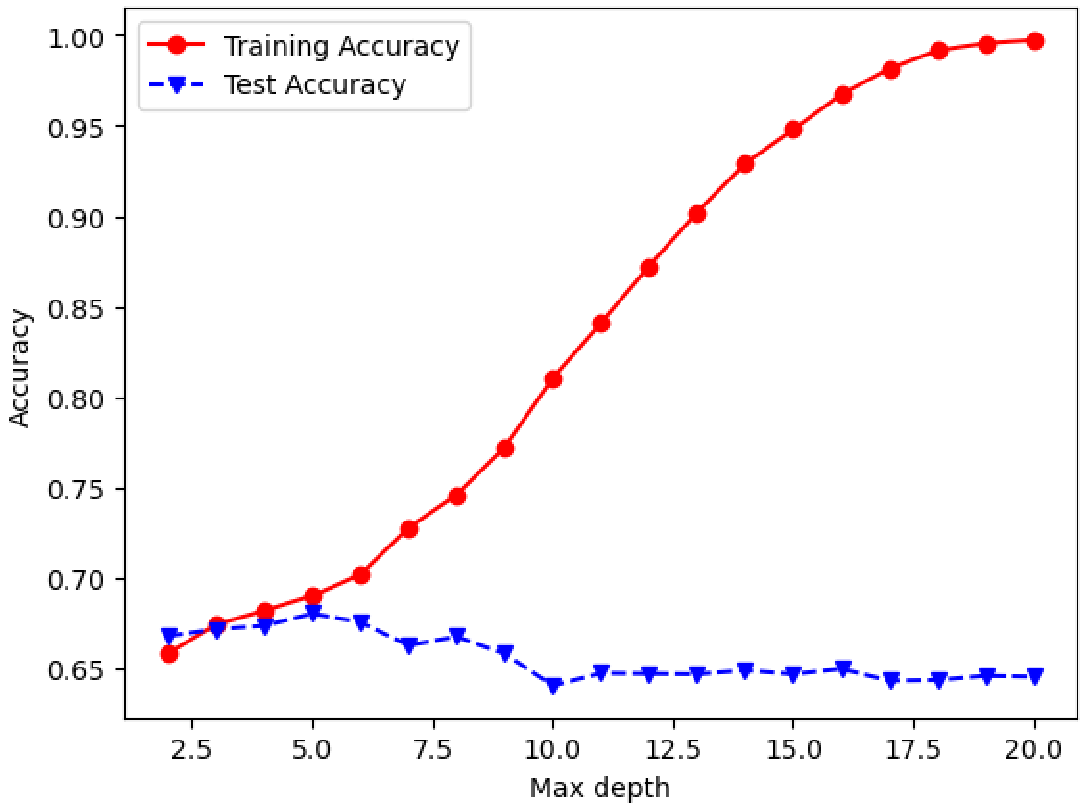
[62]:

	Method	Train Accuracy	Test Accuracy
0	Decision Tree	0.675659	0.678312

5.2.3 Plot the accuracy for different depth values (try values between 2 and 20) for both training and test:

[40]:
```
import matplotlib.pyplot as plt
%matplotlib inline

maxdepths = list(range(2,21))

trainAcc = np.zeros(len(maxdepths))
testAcc = np.zeros(len(maxdepths))

index = 0
for depth in maxdepths:
    clf2 = tree.DecisionTreeClassifier(max_depth=depth)
    clf2 = clf2.fit(X_train, Y_train)
    Y_predTrain = clf2.predict(X_train)
    Y_predTest = clf2.predict(X_test)
    trainAcc[index] = accuracy_score(Y_train, Y_predTrain)
    testAcc[index] = accuracy_score(Y_test, Y_predTest)
    index += 1


plt.plot(maxdepths,trainAcc,'ro-',maxdepths,testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
```

[40]:
```
Text(0, 0.5, 'Accuracy')
```



The plot above shows that training accuracy reaches 1 when the maximum depth is 10 onwards. Meanwhile, the test accuracy initially improves up to a maximum depth of 6, and then fluctuates in the same range even after increasing max depth.

Therefore, we could conclude that the model starts to overfit once depth increases from 6.

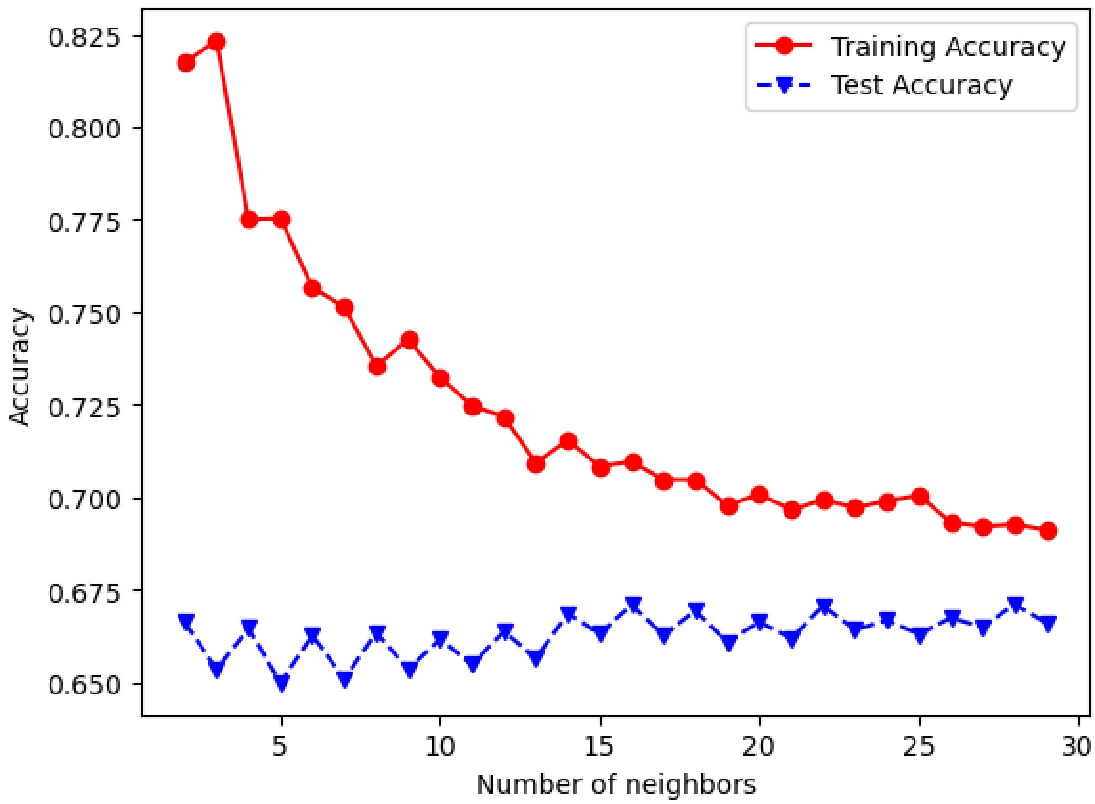
5.3 K-nn classifier:

For different number of nearest neighbors, Consider k (hyperparameter) values between 2 and 30. Plot the accuracies.

[41]:
```
from sklearn.neighbors import KNeighborsClassifier

numNeighbors = list(range(2,30))
trainAcc = []
testAcc = []

for k in numNeighbors:
    clf3 = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)
    clf3.fit(X_train, Y_train)
    Y_predTrain = clf3.predict(X_train)
    Y_predTest = clf3.predict(X_test)
```

```
    trainAcc.append(accuracy_score(Y_train, Y_predTrain))
    testAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(numNeighbors, trainAcc, 'ro-', numNeighbors, testAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
```

[41]: Text(0, 0.5, 'Accuracy')



[65]:
```
df1 = pd.DataFrame({'Method': 'K-Neighbors', 'Train Accuracy': [np.
↪mean(trainAcc)], 'Test Accuracy': [np.mean(testAcc)]})
df1
```

[65]:

	Method	Train Accuracy	Test Accuracy
0	K-Neighbors	0.87471	0.658139

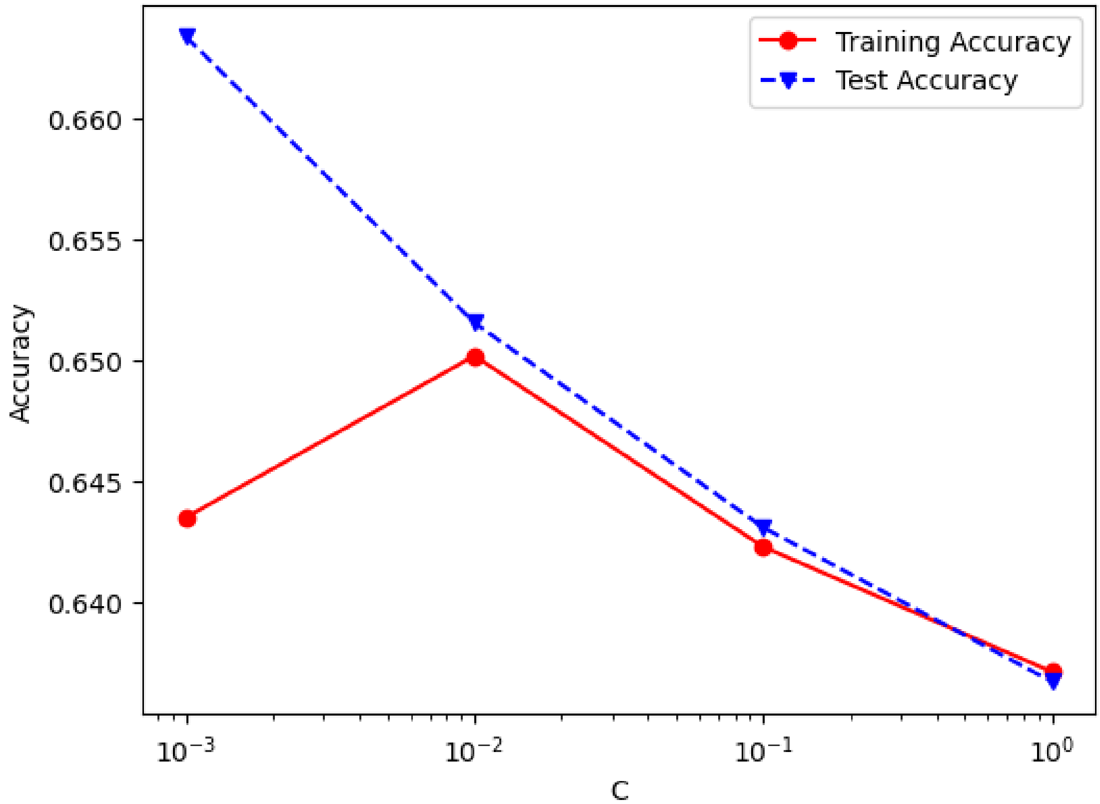
We see that the higher the numbers of k increase, the closer the gap between test accuracy and training accuracy has.

5.4 Support vector machine with a linear kernel:

Hyperparameter C can get values [0.001, 0.01, 0.1, 1]. Plot the accuracies for different C values.

[42]:
```
from sklearn.svm import SVC

C = [0.001, 0.01, 0.1, 1S]
SVMtrainAcc = []
SVMtestAcc = []

for param in C:
    clf4 = SVC(C=param,kernel='linear')
    clf4.fit(X_train, Y_train)
    Y_predTrain = clf4.predict(X_train)
    Y_predTest = clf4.predict(X_test)
    SVMtrainAcc.append(accuracy_score(Y_train, Y_predTrain))
    SVMtestAcc.append(accuracy_score(Y_test, Y_predTest))

plt.plot(C, SVMtrainAcc, 'ro-', C, SVMtestAcc,'bv--')
plt.legend(['Training Accuracy','Test Accuracy'])
plt.xlabel('C')
plt.xscale('log')
plt.ylabel('Accuracy')
```

[42]:
```
Text(0, 0.5, 'Accuracy')
```



[66]:
```
df2 = pd.DataFrame({'Method': 'SVM', 'Train Accuracy': [np.mean(SVMtrainAcc)],␣
↪'Test Accuracy': [np.mean(SVMtestAcc)]})
df2
```

[66]:

	Method	Train Accuracy	Test Accuracy
0	SVM	0.643301	0.648701

We can see that the accuracy on test set cannot be improved by using a linear kernel.

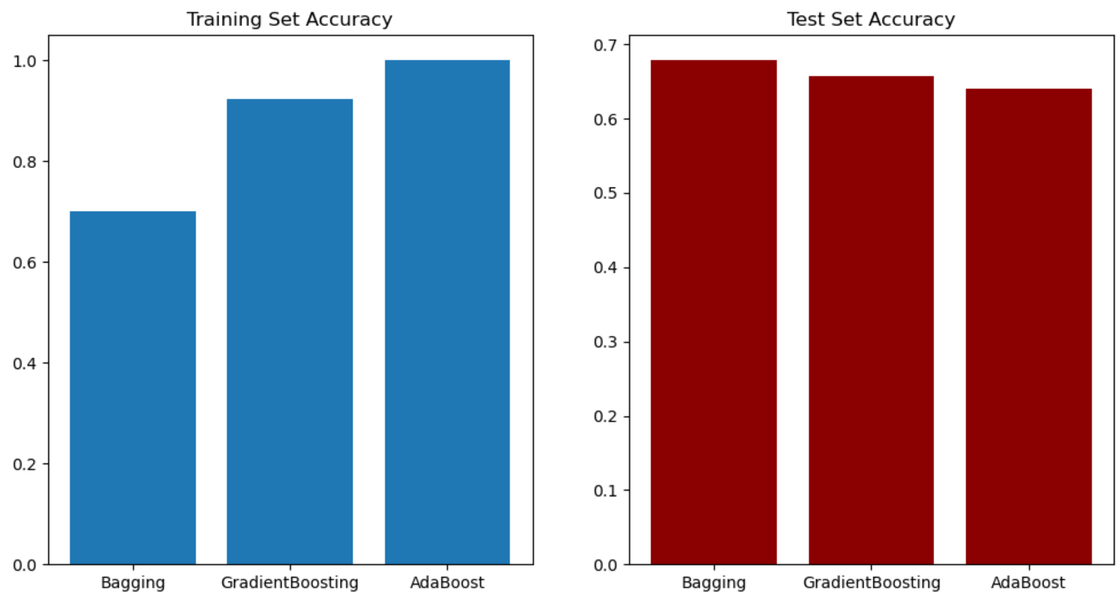
5.5 Ensemble classifiers, bagging, boosting, and adaboost:

With 150 base-classifiers with a maximum depth of 5. Plot the accuracies for both training and test.

[46]:
```
from sklearn import ensemble
from sklearn.tree import DecisionTreeClassifier

numBaseClassifiers = 150
maxdepth = 5
trainAcc = []
testAcc = []
```

```
clf5 = ensemble.
 ↪BaggingClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=numBaseClassifier
clf5.fit(X_train, Y_train)
Y_predTrain = clf5.predict(X_train)
Y_predTest = clf5.predict(X_test)
trainAcc.append(accuracy_score(Y_train, Y_predTrain))
testAcc.append(accuracy_score(Y_test, Y_predTest))

clf6 = ensemble.
 ↪GradientBoostingClassifier(n_estimators=numBaseClassifiers,max_depth=maxdepth)
clf6.fit(X_train, Y_train)
Y_predTrain = clf6.predict(X_train)
Y_predTest = clf6.predict(X_test)
trainAcc.append(accuracy_score(Y_train, Y_predTrain))
testAcc.append(accuracy_score(Y_test, Y_predTest))

clf7 = ensemble.
 ↪AdaBoostClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=numBaseClassifie
clf7.fit(X_train, Y_train)
Y_predTrain = clf7.predict(X_train)
Y_predTest = clf7.predict(X_test)
trainAcc.append(accuracy_score(Y_train, Y_predTrain))
testAcc.append(accuracy_score(Y_test, Y_predTest))

methods = ['Bagging', 'GradientBoosting', 'AdaBoost']
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,6))
ax1.bar([1.5,2.5,3.5], trainAcc)
ax1.set_xticks([1.5,2.5,3.5])
ax1.set_xticklabels(methods)
ax1.set_title("Training Set Accuracy")
ax2.bar([1.5,2.5,3.5], testAcc,color = 'darkred')
ax2.set_xticks([1.5,2.5,3.5])
ax2.set_xticklabels(methods)
ax2.set_title("Test Set Accuracy")
```

[46]: Text(0.5, 1.0, 'Test Set Accuracy')



[67]:
```
df3 = pd.DataFrame({'Method': methods, 'Train Accuracy': trainAcc, 'Test␣
 ↪Accuracy': testAcc})
df3
```

[67]:

	Method	Train Accuracy	Test Accuracy
0	Bagging	0.701728	0.678182
1	GradientBoosting	0.922401	0.656883
2	AdaBoost	1.000000	0.639351

From the plot above, we can understand that the AdaBoost is performing best on the training set, followed by Gradient Boosting Classifier, Bagging is not the ideal method for same situation. Whereas, the Bagging method could performed best in test set and AdaBoost is not the ideal method.

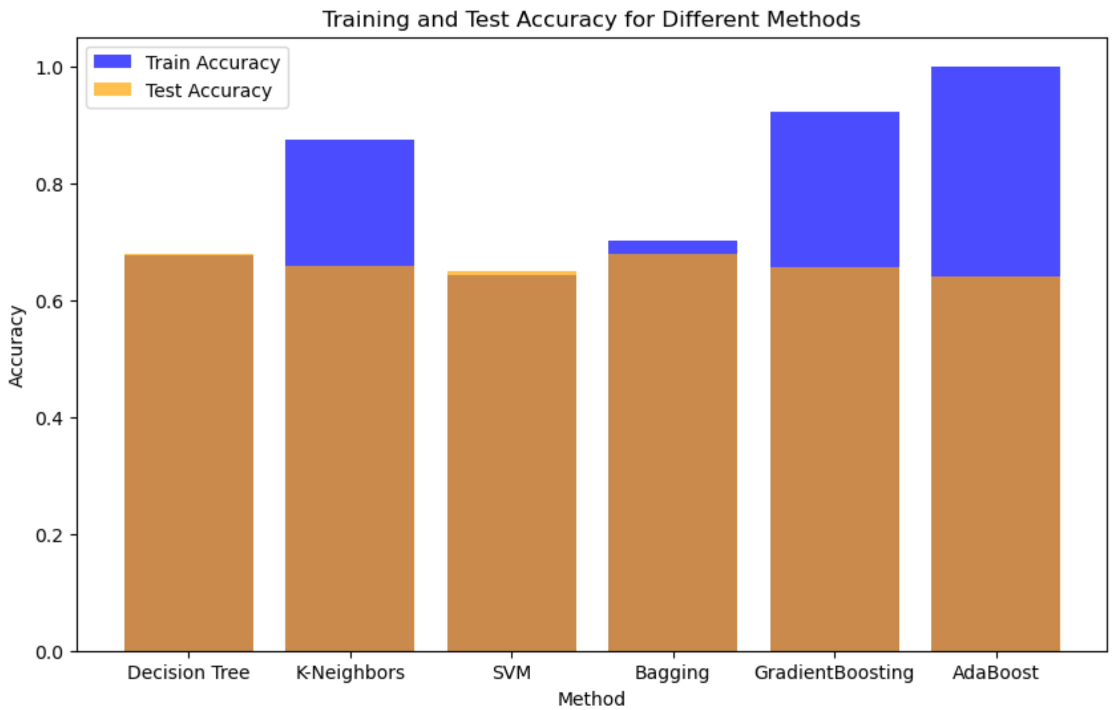
[69]:
```
# Concatenate DataFrames along rows
merged_df = pd.concat([df, df1, df2, df3], ignore_index=True)
merged_df
```

[69]:

	Method	Train Accuracy	Test Accuracy
0	Decision Tree	0.675659	0.678312
1	K-Neighbors	0.874710	0.658139
2	SVM	0.643301	0.648701
3	Bagging	0.701728	0.678182
4	GradientBoosting	0.922401	0.656883
5	AdaBoost	1.000000	0.639351

[79]:
```
# Plotting
plt.figure(figsize=(10, 6))
```

```
plt.bar(merged_df['Method'], merged_df['Train Accuracy'], color='blue', alpha=0.
  ↪7, label='Train Accuracy')
plt.bar(merged_df['Method'], merged_df['Test Accuracy'], color='orange',␣
  ↪alpha=0.7, label='Test Accuracy')
plt.xlabel('Method')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy for Different Methods')
plt.legend()
plt.show()
```



In this data:

- Decision Tree: Moderate test accuracy.
- K-Neighbors: Good test accuracy but check for overfitting.
- SVM: Moderate test accuracy.
- Bagging: Moderate test accuracy.
- GradientBoosting: High training accuracy but check for overfitting.
- AdaBoost: High training accuracy but check for overfitting.

5.6 Cluster:

Pick k and form k clusters by assigning each instance to its nearest centroid. We can choose k = 2 as 2 clusters, display the centroid for each of the two clusters.

[70]:
```
from sklearn import cluster

k_means = cluster.KMeans(n_clusters=2, max_iter=50, random_state=1)
k_means.fit(data3)
labels = k_means.labels_
pd.DataFrame(labels, columns=['Cluster ID'])
```

[70]:
```
       Cluster ID
0               0
1               0
2               1
3               0
4               0
…             …
10994           0
10995           0
10996           0
10997           0
10998           0

[10999 rows x 1 columns]
```

Compute the centroid of each cluster.

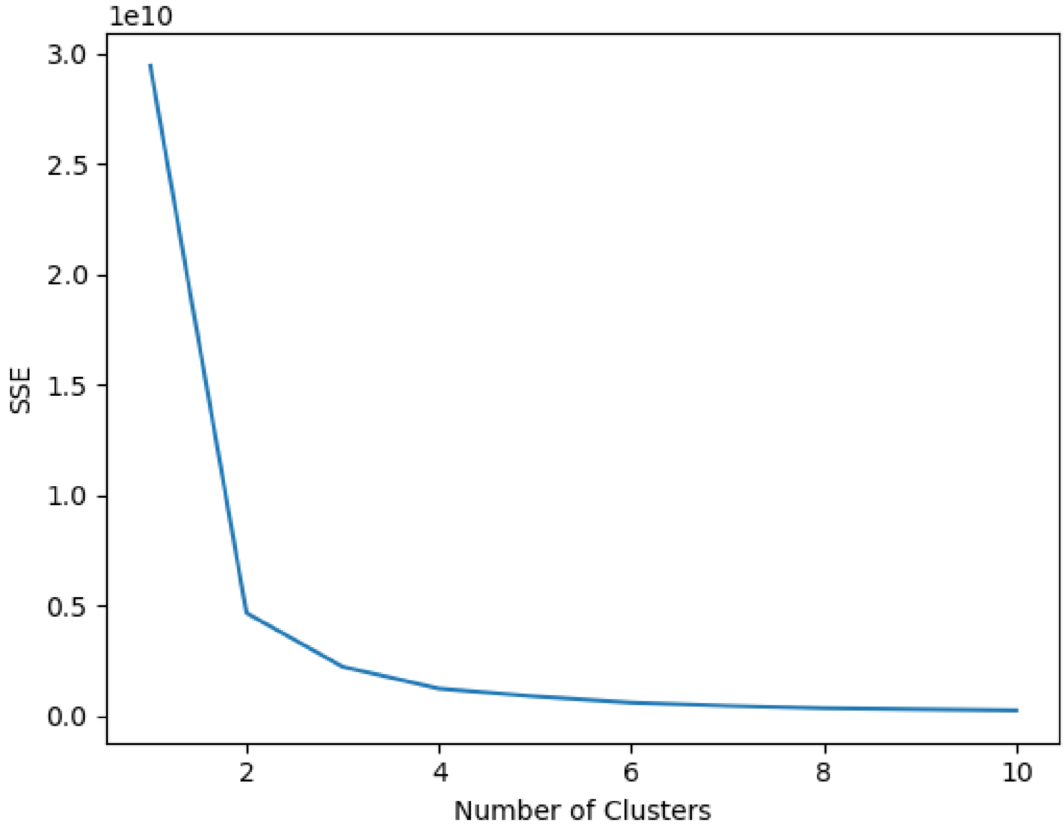
[72]:
```
centroids = k_means.cluster_centers_
pd.DataFrame(centroids,columns=data3.columns)
```

[72]:
```
   Warehouse_block  Mode_of_Shipment  Customer_care_calls  Customer_rating  \
0         2.330381          1.519755             4.417802         2.987965
1         2.335406          1.514936             3.811827         2.992267

   Cost_of_the_Product  Prior_purchases  Product_importance    Gender  \
0           218.177793         3.857175            1.342870  0.490009
1           204.867324         3.374223            1.348143  0.499773

   Discount_offered  Weight_in_gms  Reached.on.Time_Y.N
0         21.104905    1797.138283             0.762489
1          8.210159    4860.644882             0.485974
```

[73]:
```
numClusters = list(range(1,11))
SSE = []
for k in numClusters:
    k_means = cluster.KMeans(n_clusters=k)
    k_means.fit(data3)
    SSE.append(k_means.inertia_)

plt.plot(numClusters, SSE)
plt.xlabel('Number of Clusters')
```

```
plt.ylabel('SSE')
```

[73]: Text(0, 0.5, 'SSE')



The estimated clusters in the dataset should start from 2, since SSE got dramatic decline at this k=2. At the same time, we could see that the more clusters we assign for K, the better result it becomes.

6 Challenges & Limitations:

6.1 Challenges:

- **Complex Interdependencies:** The complex network of interdependencies between different variables was one of the main difficulties faced. It was tough to analyze the intricate relationships between Customer Care Calls, Customer Ratings, and Shipment Modes.
- **Managing Categorical Variables:** The dataset made it challenging to manage categorical variables well. Strategies for turning variables like Warehouse Block and Mode of Shipment into useful numerical representations must be carefully thought out to ensure understanding.
- **Unbalanced Data:** Predicting on-time deliveries in the face of imbalanced data presented difficulties for model training. To guarantee the robustness of the model, specific techniques

were needed due to the skewed distribution of the target variable “Reached on Time”.

- **Complexity of the Model:** Finding the ideal balance between model complexity and ease of use was complex. Iterative adjustments and careful analysis were required to avoid overfitting or underfitting while incorporating diverse features like Discount Offered and Product Importance.

6.2 Limitations of the Dataset and Model:

- **Limited Historical Data:** The dataset’s temporal scope is a limitation, as it offers a snapshot rather than a longitudinal view of customer behavior. Long-term trends and evolving patterns over time may not be fully captured.
- **Lack of External Factors:** External factors, such as market trends, economic conditions, or regional variations, were not included in the dataset. This limits the holistic understanding of the broader context in which customer interactions and deliveries take place.
- **Customer Feedback Dynamics:** The dataset’s reliance on Customer Ratings as a feedback metric may not capture the full spectrum of customer sentiments. Nuances in qualitative feedback or reasons behind specific ratings are not accounted for, limiting the depth of customer insight.
- **Model Generalization:** The model’s performance may be context-specific and might not generalize well to diverse e-commerce scenarios. External validation on a broader range of datasets would be essential to ensure its adaptability.

7 Conclusion:

In summary, the customer database analysis of the global e-commerce company has yielded valuable insights that have the potential to influence both customer satisfaction and operational efficiency greatly. Important conclusions highlight how crucial it is to match customer preferences for shipment modes, improve customer support tactics, and deal with the particular difficulties presented by high-importance products. These findings highlight the necessity of adopting a customer-centric strategy and staying current with changing market trends. Here are four recommendations to support the business moving forward:

- **Strategies Focused on the Customer:** Make customer-centric strategies your top priority by matching customer expectations and preferences with shipment methods, customer care procedures, and promotional activities.
- **Ongoing Observation and Modification:** Provide systems for tracking customer feedback continuously and modifying operational plans in response to changing consumer attitudes and trends.
- **Demand and Inventory Predictive Analytics:** Utilize predictive analytics models to anticipate demand trends and inventory requirements, enabling proactive steps to satisfy customer expectations.
- **Integration of Technology:** Use cutting-edge technologies, such as AI-driven customer service and real-time tracking, to improve responsiveness and transparency in the shipping process.

By putting the suggested strategies into practice, the business will be better positioned to maintain its competitive advantage and uphold its dedication to providing outstanding customer service in the ever-changing world of e-commerce.