Mid-Term Project

Rishabh Kaushick, NU ID: 002808996, kaushick.r@northeastern.edu

College of Engineering, Northeastern University Toronto, ON

In [1]:
```
# importing the required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

In [2]:
```
# loading the data
# structured data
scm_dataset_path = 'data/DataCoSupplyChainDataset.csv'
scm_df = pd.read_csv(scm_dataset_path, encoding='latin-1')

# unstructured data
scm_unstructured_path = 'data/tokenized_access_logs.csv'
scm_unstructured_df = pd.read_csv(scm_unstructured_path)
```

In [3]:
```
# Let's see what the first five rows data looks like
scm_df.head()
```

Out[3]:

	Type	Days for shipping (real)	Days for shipment (scheduled)	Benefit per order	Sales per customer	Delivery Status	Late_deli
0	DEBIT	3	4	91.250000	314.640015	Advance shipping	
1	TRANSFER	5	4	-249.089996	311.359985	Late delivery	
2	CASH	4	4	-247.779999	309.720001	Shipping on time	
3	DEBIT	3	4	22.860001	304.809998	Advance shipping	
4	PAYMENT	2	4	134.210007	298.250000	Advance shipping	

5 rows × 53 columns

In [4]:
```
# total number of rows in the above csv:
scm_df.count()
```

Out[4]:
```
Type                           180519
Days for shipping (real)       180519
Days for shipment (scheduled)  180519
Benefit per order              180519
Sales per customer             180519
Delivery Status                180519
Late_delivery_risk             180519
Category Id                    180519
Category Name                  180519
Customer City                  180519
Customer Country               180519
Customer Email                 180519
Customer Fname                 180519
Customer Id                    180519
Customer Lname                 180511
Customer Password              180519
Customer Segment               180519
Customer State                 180519
Customer Street                180519
Customer Zipcode               180516
Department Id                  180519
Department Name                180519
Latitude                       180519
Longitude                      180519
Market                         180519
Order City                     180519
Order Country                  180519
Order Customer Id              180519
order date (DateOrders)        180519
Order Id                       180519
Order Item Cardprod Id         180519
Order Item Discount            180519
Order Item Discount Rate       180519
Order Item Id                  180519
Order Item Product Price       180519
Order Item Profit Ratio        180519
Order Item Quantity            180519
Sales                          180519
Order Item Total               180519
Order Profit Per Order         180519
Order Region                   180519
Order State                    180519
Order Status                   180519
Order Zipcode                   24840
Product Card Id                180519
Product Category Id            180519
Product Description                 0
Product Image                  180519
Product Name                   180519
Product Price                  180519
Product Status                 180519
shipping date (DateOrders)     180519
Shipping Mode                  180519
dtype: int64
```

In [5]:
```
scm_unstructured_df.head()
```

Out[5]:

	Product	Category	Date	Month	Hour	Department	ip	
0	adidas Brazuca 2017 Official Match Ball	baseball & softball	9/1/2017 6:00	Sep	6	fitness	37.97.182.65	
1	The North Face Women's Recon Backpack	hunting & shooting	9/1/2017 6:00	Sep	6	fan shop	206.56.112.1	/de
2	adidas Kids' RG III Mid Football Cleat	featured shops	9/1/2017 6:00	Sep	6	apparel	215.143.180.0	
3	Under Armour Men's Compression EV SL Slide	electronics	9/1/2017 6:00	Sep	6	footwear	206.56.112.1	
4	Pelican Sunstream 100 Kayak	water sports	9/1/2017 6:01	Sep	6	fan shop	136.108.56.242	/

In [6]:
```
scm_unstructured_df.count()
```

Out[6]:
```
Product       469977
Category      469977
Date          469977
Month         469977
Hour          469977
Department    469977
ip            469977
url           469977
dtype: int64
```

In the main csv dataset, there are 53 columns and 180,519 rows. Therefore there are 53 X 180,519 = 9,567,507 data points.

However, from the unstructured dataset, we can see that there are 8 columns and 469,977 rows. Therefore there are 8 X 469,977 = 3,759,816 data points.

Since the columns in each of the datasets are different from each other, and since there is a lot more data in the structured dataframe, we will use the same and disregard the unstructured data in this project.

In [7]:
```
# understanding the number of distinct non null numbers are present in ea
scm_df.nunique()
```

Out[7]:
```
Type                              4
Days for shipping (real)          7
Days for shipment (scheduled)     4
Benefit per order             21998
Sales per customer             2927
Delivery Status                   4
Late_delivery_risk                2
Category Id                      51
Category Name                    50
Customer City                   563
Customer Country                  2
Customer Email                    1
Customer Fname                  782
Customer Id                   20652
Customer Lname                 1109
Customer Password                 1
Customer Segment                  3
Customer State                   46
Customer Street                7458
Customer Zipcode                995
Department Id                    11
Department Name                  11
Latitude                      11250
Longitude                      4487
Market                            5
Order City                     3597
Order Country                   164
Order Customer Id             20652
order date (DateOrders)       65752
Order Id                      65752
Order Item Cardprod Id          118
Order Item Discount            1017
Order Item Discount Rate         18
Order Item Id                180519
Order Item Product Price         75
Order Item Profit Ratio         162
Order Item Quantity               5
Sales                           193
Order Item Total               2927
Order Profit Per Order        21998
Order Region                     23
Order State                    1089
Order Status                      9
Order Zipcode                   609
Product Card Id                 118
Product Category Id              51
Product Description               0
Product Image                   118
Product Name                    118
Product Price                    75
Product Status                    1
shipping date (DateOrders)    63701
Shipping Mode                     4
dtype: int64
```

In [8]:
```
# checking which columns have null rows
scm_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180519 entries, 0 to 180518
Data columns (total 53 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   Type                           180519 non-null  object 
 1   Days for shipping (real)       180519 non-null  int64  
 2   Days for shipment (scheduled)  180519 non-null  int64  
 3   Benefit per order              180519 non-null  float64
 4   Sales per customer             180519 non-null  float64
 5   Delivery Status                180519 non-null  object 
 6   Late_delivery_risk             180519 non-null  int64  
 7   Category Id                    180519 non-null  int64  
 8   Category Name                  180519 non-null  object 
 9   Customer City                  180519 non-null  object 
 10  Customer Country               180519 non-null  object 
 11  Customer Email                 180519 non-null  object 
 12  Customer Fname                 180519 non-null  object 
 13  Customer Id                    180519 non-null  int64  
 14  Customer Lname                 180511 non-null  object 
 15  Customer Password              180519 non-null  object 
 16  Customer Segment               180519 non-null  object 
 17  Customer State                 180519 non-null  object 
 18  Customer Street                180519 non-null  object 
 19  Customer Zipcode               180516 non-null  float64
 20  Department Id                  180519 non-null  int64  
 21  Department Name                180519 non-null  object 
 22  Latitude                       180519 non-null  float64
 23  Longitude                      180519 non-null  float64
 24  Market                         180519 non-null  object 
 25  Order City                     180519 non-null  object 
 26  Order Country                  180519 non-null  object 
 27  Order Customer Id              180519 non-null  int64  
 28  order date (DateOrders)        180519 non-null  object 
 29  Order Id                       180519 non-null  int64  
 30  Order Item Cardprod Id         180519 non-null  int64  
 31  Order Item Discount            180519 non-null  float64
 32  Order Item Discount Rate       180519 non-null  float64
 33  Order Item Id                  180519 non-null  int64  
 34  Order Item Product Price       180519 non-null  float64
 35  Order Item Profit Ratio        180519 non-null  float64
 36  Order Item Quantity            180519 non-null  int64  
 37  Sales                          180519 non-null  float64
 38  Order Item Total               180519 non-null  float64
 39  Order Profit Per Order         180519 non-null  float64
 40  Order Region                   180519 non-null  object 
 41  Order State                    180519 non-null  object 
 42  Order Status                   180519 non-null  object 
 43  Order Zipcode                  24840 non-null   float64
 44  Product Card Id                180519 non-null  int64  
 45  Product Category Id            180519 non-null  int64  
 46  Product Description            0 non-null       float64
 47  Product Image                  180519 non-null  object 
 48  Product Name                   180519 non-null  object 
 49  Product Price                  180519 non-null  float64
 50  Product Status                 180519 non-null  int64  
 51  shipping date (DateOrders)     180519 non-null  object 
 52  Shipping Mode                  180519 non-null  object 
dtypes: float64(15), int64(14), object(24)
memory usage: 73.0+ MB
```

From the above - we can see that there are null values in the following columns

- Order Zipcode
- Project Description

In [9]:
```
scm_df.describe()
```

Out[9]:

	Days for shipping (real)	Days for shipment (scheduled)	Benefit per order	Sales per customer	Late_delivery
count	180519.000000	180519.000000	180519.000000	180519.000000	180519.00
mean	3.497654	2.931847	21.974989	183.107609	0.54
std	1.623722	1.374449	104.433526	120.043670	0.49
min	0.000000	0.000000	-4274.979980	7.490000	0.00
25%	2.000000	2.000000	7.000000	104.379997	0.00
50%	3.000000	4.000000	31.520000	163.990005	1.00
75%	5.000000	4.000000	64.800003	247.399994	1.00
max	6.000000	4.000000	911.799988	1939.989990	1.00

8 rows × 29 columns

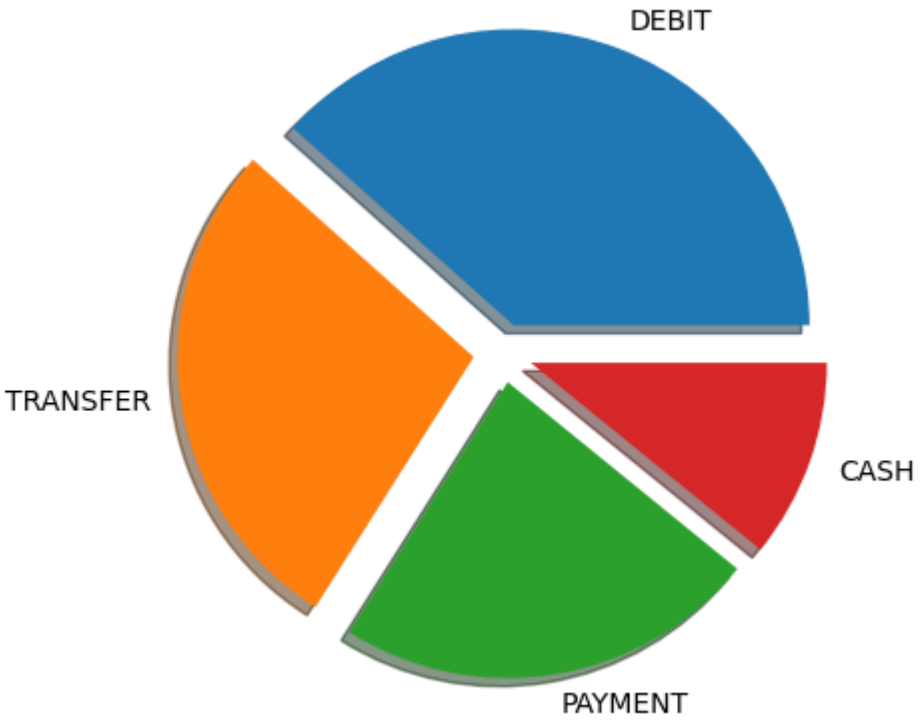
In [91]:
```
# Finding the correlation between the other columns.
```

In [10]:
```
# Let's see the type of data in the 'Type' Column
scm_df['Type'].value_counts()
```

Out[10]:
```
Type
DEBIT       69295
TRANSFER    49883
PAYMENT     41725
CASH        19616
Name: count, dtype: int64
```

In [11]:
```
# visualizing the same thing in a pie chart

# Trying to create a plot with each section exploded from the pie.
my_explode = [0.1, 0.1, 0.1, 0.1]
my_labels = np.array(['DEBIT', 'TRANSFER', 'PAYMENT', 'CASH'])
type_value_array = (scm_df['Type'].value_counts())
plt.pie(type_value_array, labels=my_labels, shadow=True, explode=my_explo
```

Out[11]:
```
([<matplotlib.patches.Wedge at 0x155ac6310>,
  <matplotlib.patches.Wedge at 0x155ac6460>,
  <matplotlib.patches.Wedge at 0x155acb0a0>,
  <matplotlib.patches.Wedge at 0x155acbb80>],
 [Text(0.428168266257398, 1.1210137982068438, 'DEBIT'),
  Text(-1.1885215498911712, -0.16557936297826523, 'TRANSFER'),
  Text(0.19342361628901053, -1.1843087877161436, 'PAYMENT'),
  Text(1.130752535999951, -0.40174457349126647, 'CASH')])
```

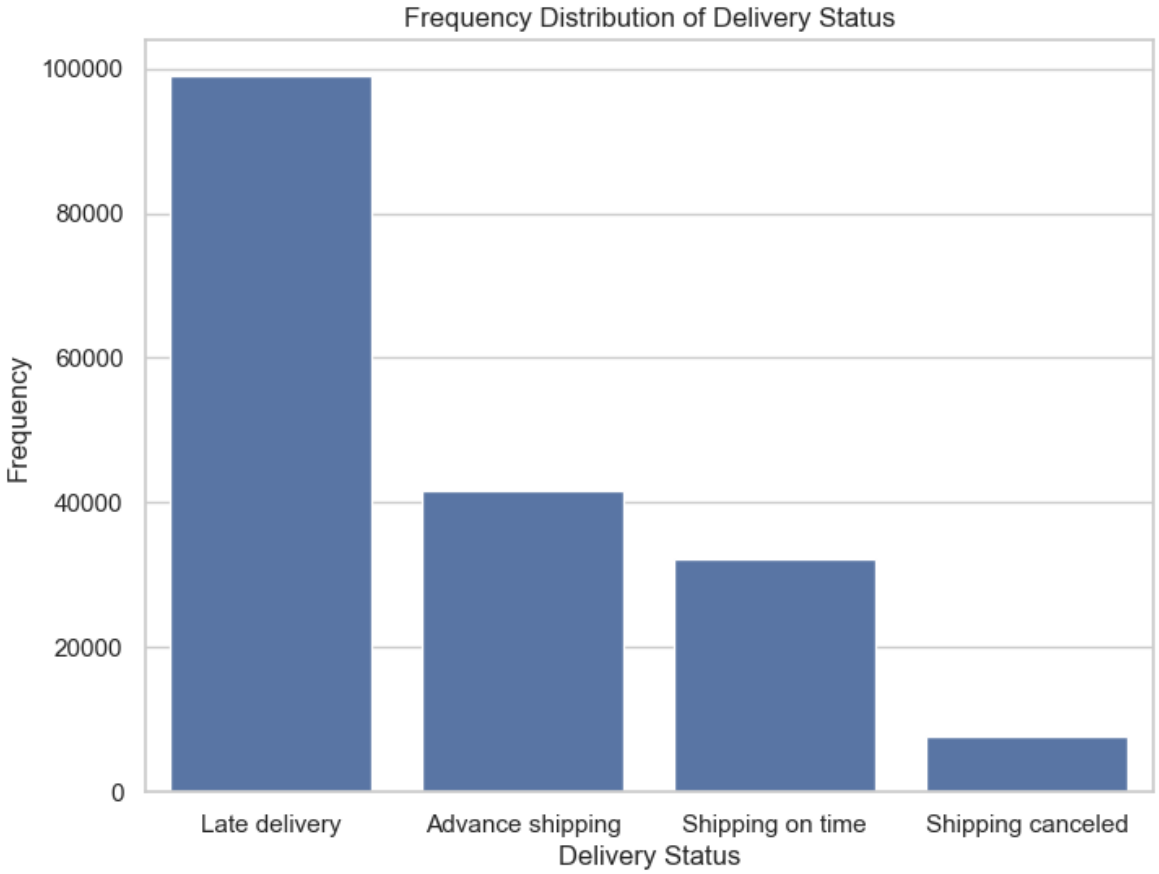


In [12]:
```
# Delivery Status
scm_df['Delivery Status'].value_counts()
```

Out[12]:
```
Delivery Status
Late delivery        98977
Advance shipping     41592
Shipping on time     32196
Shipping canceled     7754
Name: count, dtype: int64
```

In [14]:
```
# plotting it in a graph to visualize it better
sns.set(style="whitegrid")
plt.figure(figsize = (8,6))
sns.barplot(scm_df['Delivery Status'].value_counts())
plt.xlabel("Delivery Status")
plt.ylabel("Frequency")
plt.title("Frequency Distribution of Delivery Status")
```

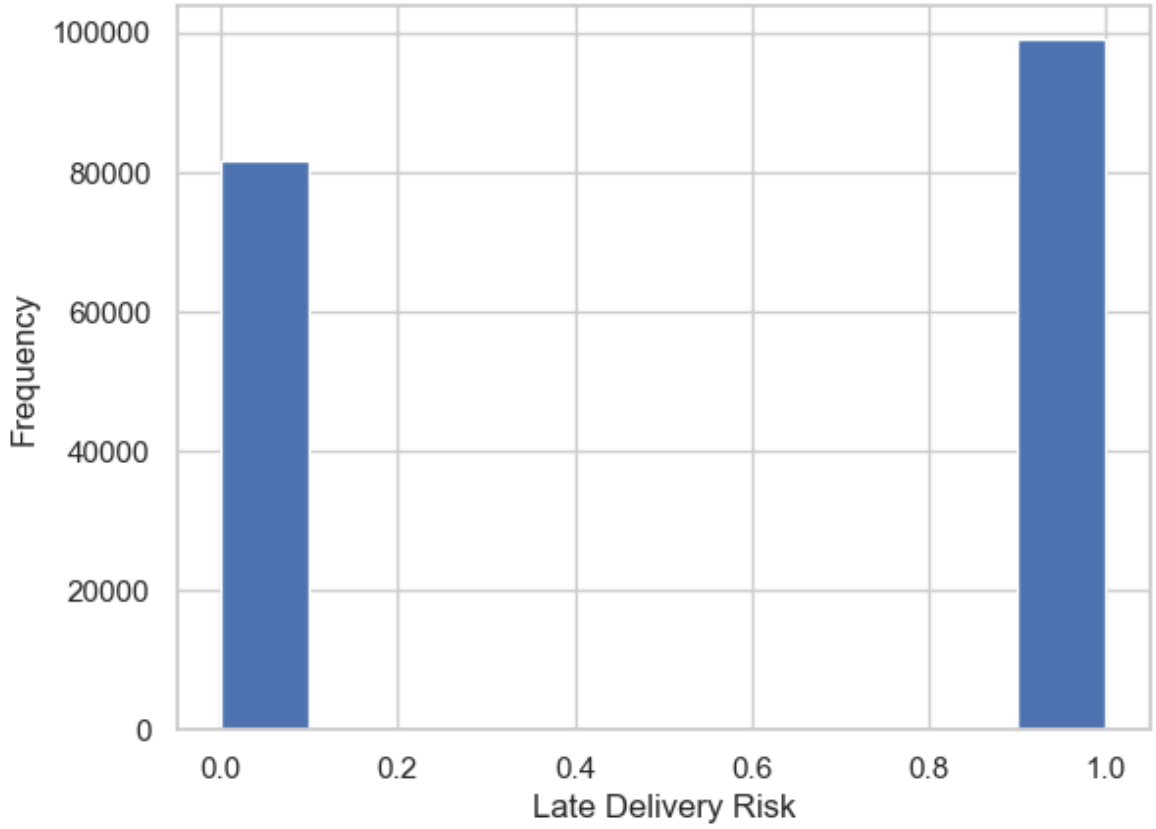
Out[14]:
```
Text(0.5, 1.0, 'Frequency Distribution of Delivery Status')
```



We can see that the Delivery Status for most of the deliveries in the dataset are 'Late delivery'

In [31]:
```
# late delivery risk column
# scm_df['Late_delivery_risk'].value_counts()
scm_df['Late_delivery_risk'].plot.hist(xlabel='Late Delivery Risk')
```

Out[31]:
```
<Axes: xlabel='Late Delivery Risk', ylabel='Frequency'>
```



In [32]:
```
# late delivery risk and delivery status
sns.countplot(scm_df, x='Delivery Status', hue='Late_delivery_risk')
```

Out[32]:
```
<Axes: xlabel='Delivery Status', ylabel='count'>
```

The above graph is counter-intuitive. Only on Late deliveries - we have 'Late delivery risk' as 1, and on all other Delivery statuses, the late delivery risk is 0.

Therefore, while training the model we can remove the Delivery Status column - since it is a redundant version of the 'Late delivery risk' column.

In [33]:
```
# checking which columns have null rows
scm_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180519 entries, 0 to 180518
Data columns (total 53 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   Type                           180519 non-null  object 
 1   Days for shipping (real)       180519 non-null  int64  
 2   Days for shipment (scheduled)  180519 non-null  int64  
 3   Benefit per order              180519 non-null  float64
 4   Sales per customer             180519 non-null  float64
 5   Delivery Status                180519 non-null  object 
 6   Late_delivery_risk             180519 non-null  int64  
 7   Category Id                    180519 non-null  int64  
 8   Category Name                  180519 non-null  object 
 9   Customer City                  180519 non-null  object 
 10  Customer Country               180519 non-null  object 
 11  Customer Email                 180519 non-null  object 
 12  Customer Fname                 180519 non-null  object 
 13  Customer Id                    180519 non-null  int64  
 14  Customer Lname                 180511 non-null  object 
 15  Customer Password              180519 non-null  object 
 16  Customer Segment               180519 non-null  object 
 17  Customer State                 180519 non-null  object 
 18  Customer Street                180519 non-null  object 
 19  Customer Zipcode               180516 non-null  float64
 20  Department Id                  180519 non-null  int64  
 21  Department Name                180519 non-null  object 
 22  Latitude                       180519 non-null  float64
 23  Longitude                      180519 non-null  float64
 24  Market                         180519 non-null  object 
 25  Order City                     180519 non-null  object 
 26  Order Country                  180519 non-null  object 
 27  Order Customer Id              180519 non-null  int64  
 28  order date (DateOrders)        180519 non-null  object 
 29  Order Id                       180519 non-null  int64  
 30  Order Item Cardprod Id         180519 non-null  int64  
 31  Order Item Discount            180519 non-null  float64
 32  Order Item Discount Rate       180519 non-null  float64
 33  Order Item Id                  180519 non-null  int64  
 34  Order Item Product Price       180519 non-null  float64
 35  Order Item Profit Ratio        180519 non-null  float64
 36  Order Item Quantity            180519 non-null  int64  
 37  Sales                          180519 non-null  float64
 38  Order Item Total               180519 non-null  float64
 39  Order Profit Per Order         180519 non-null  float64
 40  Order Region                   180519 non-null  object 
 41  Order State                    180519 non-null  object 
 42  Order Status                   180519 non-null  object 
 43  Order Zipcode                  24840 non-null   float64
 44  Product Card Id                180519 non-null  int64  
 45  Product Category Id            180519 non-null  int64  
 46  Product Description            0 non-null       float64
 47  Product Image                  180519 non-null  object 
 48  Product Name                   180519 non-null  object 
 49  Product Price                  180519 non-null  float64
 50  Product Status                 180519 non-null  int64  
 51  shipping date (DateOrders)     180519 non-null  object 
 52  Shipping Mode                  180519 non-null  object 
dtypes: float64(15), int64(14), object(24)
memory usage: 73.0+ MB
```

Let us try to find out more about location data of the customers which result in late delivery.

In [34]:
```
# Plotting geographic data of the customer locations which resulted in La
sns.countplot(scm_df, x='Order Region', hue='Delivery Status')
```

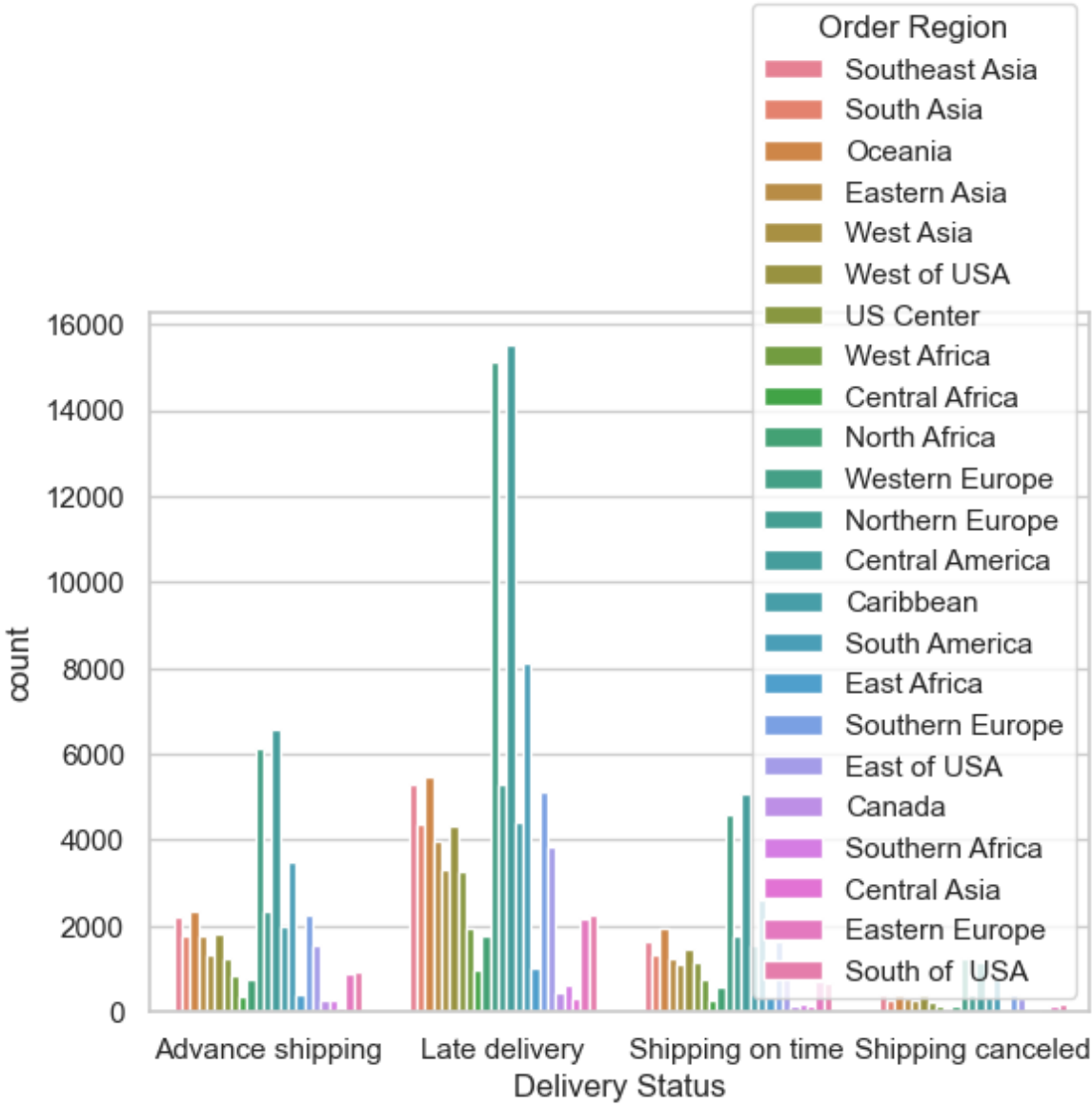
Out[34]:
```
<Axes: xlabel='Order Region', ylabel='count'>
```

The above plot is not useful since we are unable to see the regions on the x-axis

However, we can plot the same data but instead of order region in the x axis & different colors for delivery status, let us consider delivery status on the x-axis and regions with different colors

In [35]:
```
# trying to plot the same figure in a different way
sns.countplot(scm_df, x='Delivery Status', hue='Order Region')
```

Out[35]:
```
<Axes: xlabel='Delivery Status', ylabel='count'>
```



This is better, but it is still hard to read the graph to understand exactly which region has most delivery statuses as 'Late delievry'

Therefore, using the plotly library as shown below in order to have an interactive graph to plot the same data.

In [36]:
```
# displaying the same data using plotly library
data_delivery_status_region=scm_df.groupby(['Delivery Status', 'Order Reg
px.bar(data_delivery_status_region, x='Delivery Status', y='Number of Ord
       )
```

In [37]:
```
# late delivery risk vs order region
data_delivery_status_region=scm_df.groupby(['Late_delivery_risk', 'Order
px.bar(data_delivery_status_region, x='Late_delivery_risk', y='Number of
       )
```

In [38]:
```
# Customer Segment
scm_df['Customer Segment'].value_counts()
```

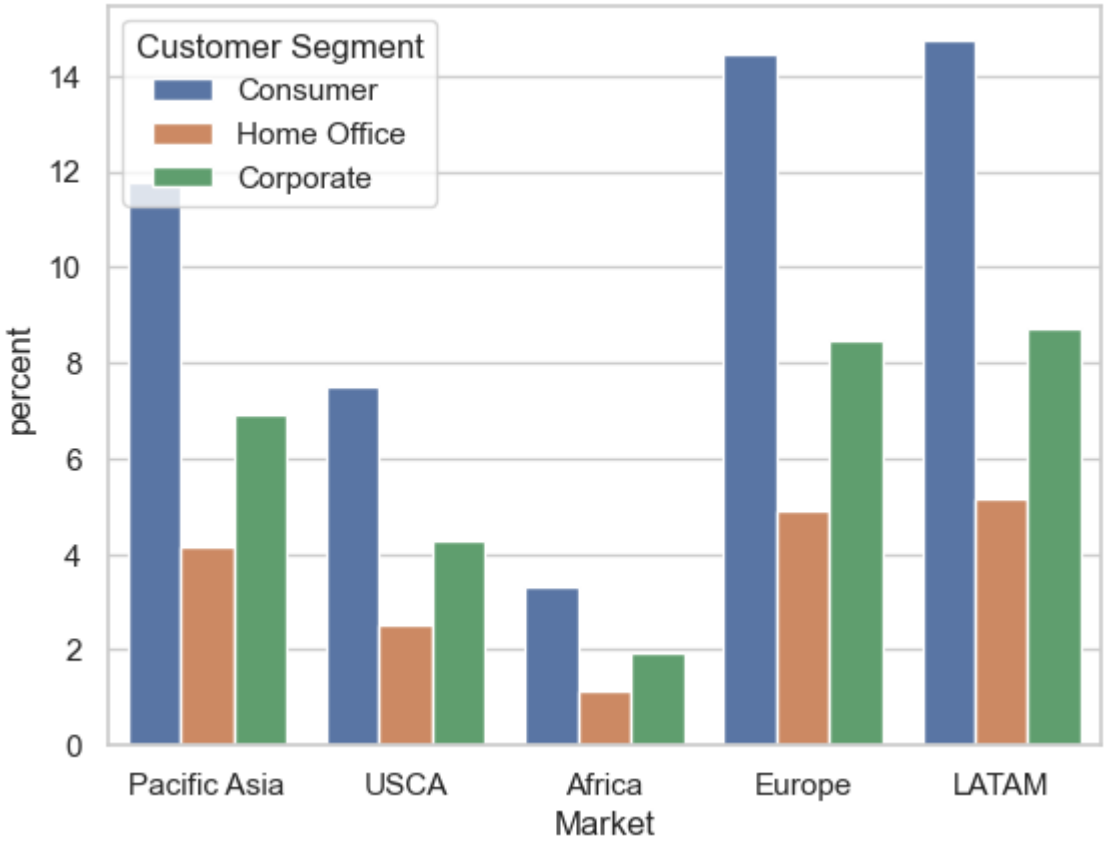
Out[38]:
```
Customer Segment
Consumer       93504
Corporate      54789
Home Office    32226
Name: count, dtype: int64
```

In [39]:
```
# Market
scm_df['Market'].value_counts()
```

Out[39]:
```
Market
LATAM           51594
Europe          50252
Pacific Asia    41260
USCA            25799
Africa          11614
Name: count, dtype: int64
```

In [40]:
```
# In each market - finding the customer segment (Consumer, Home Office an
sns.countplot(scm_df, x="Market", hue="Customer Segment", stat="percent",
```

Out[40]:
```
<Axes: xlabel='Market', ylabel='percent'>
```

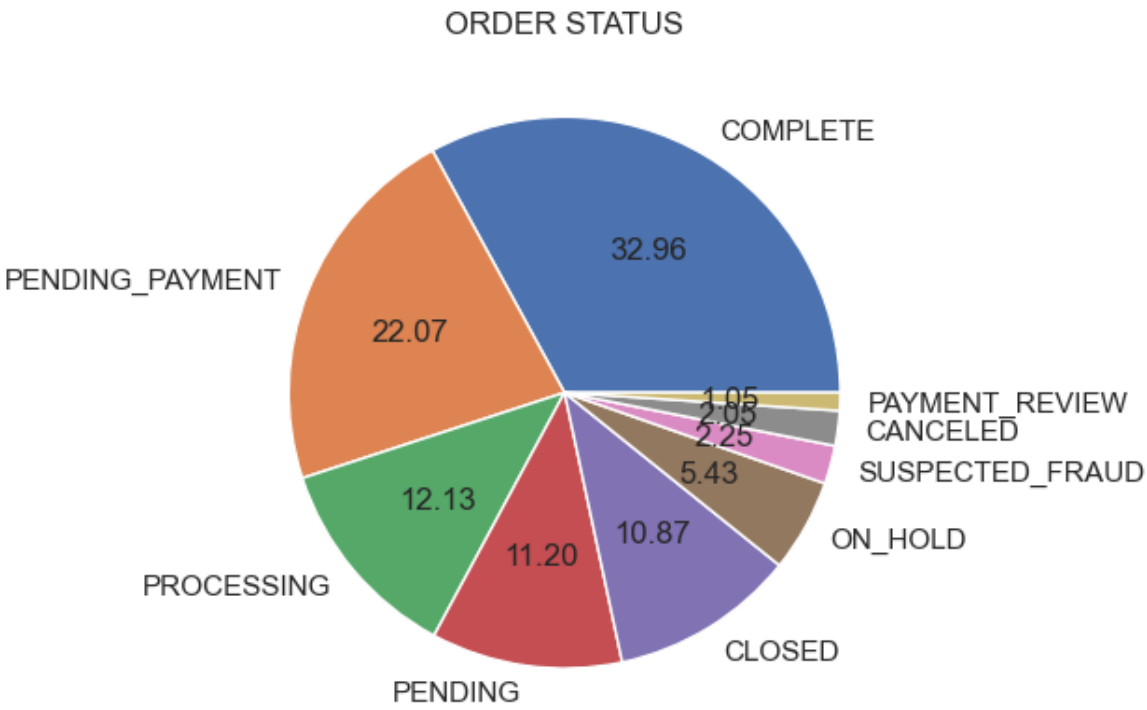


In [41]:
```
# Order Status
scm_df['Order Status'].value_counts()
```

Out[41]:
```
Order Status
COMPLETE           59491
PENDING_PAYMENT    39832
PROCESSING         21902
PENDING            20227
CLOSED             19616
ON_HOLD             9804
SUSPECTED_FRAUD     4062
CANCELED            3692
PAYMENT_REVIEW      1893
Name: count, dtype: int64
```

In [42]:
```
# again visualizing the order status with pie charts
scm_df['Order Status'].value_counts().plot.pie(title='ORDER STATUS',autop
```

Out[42]:
```
<Axes: title={'center': 'ORDER STATUS'}>
```



In [43]:
```
# Product Status
scm_df['Product Status'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 180519 entries, 0 to 180518
Series name: Product Status
Non-Null Count   Dtype
--------------   -----
180519 non-null  int64
dtypes: int64(1)
memory usage: 1.4 MB
```

In [44]:
```
scm_df['Product Status'].value_counts()
```

Out[44]:
```
Product Status
0    180519
Name: count, dtype: int64
```

Product Status can be 1 - not available and 0 available. In this dataset, we can see that all the values are 0 (available).

Therefore this data is not useful to us, and we can drop this column.

In [45]:
```
# Shipping Mode
scm_df['Shipping Mode'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 180519 entries, 0 to 180518
Series name: Shipping Mode
Non-Null Count   Dtype
--------------   -----
180519 non-null  object
dtypes: object(1)
memory usage: 1.4+ MB
```

In [46]:
```
scm_df['Shipping Mode'].describe()
```

Out[46]:
```
count             180519
unique                 4
top       Standard Class
freq              107752
Name: Shipping Mode, dtype: object
```

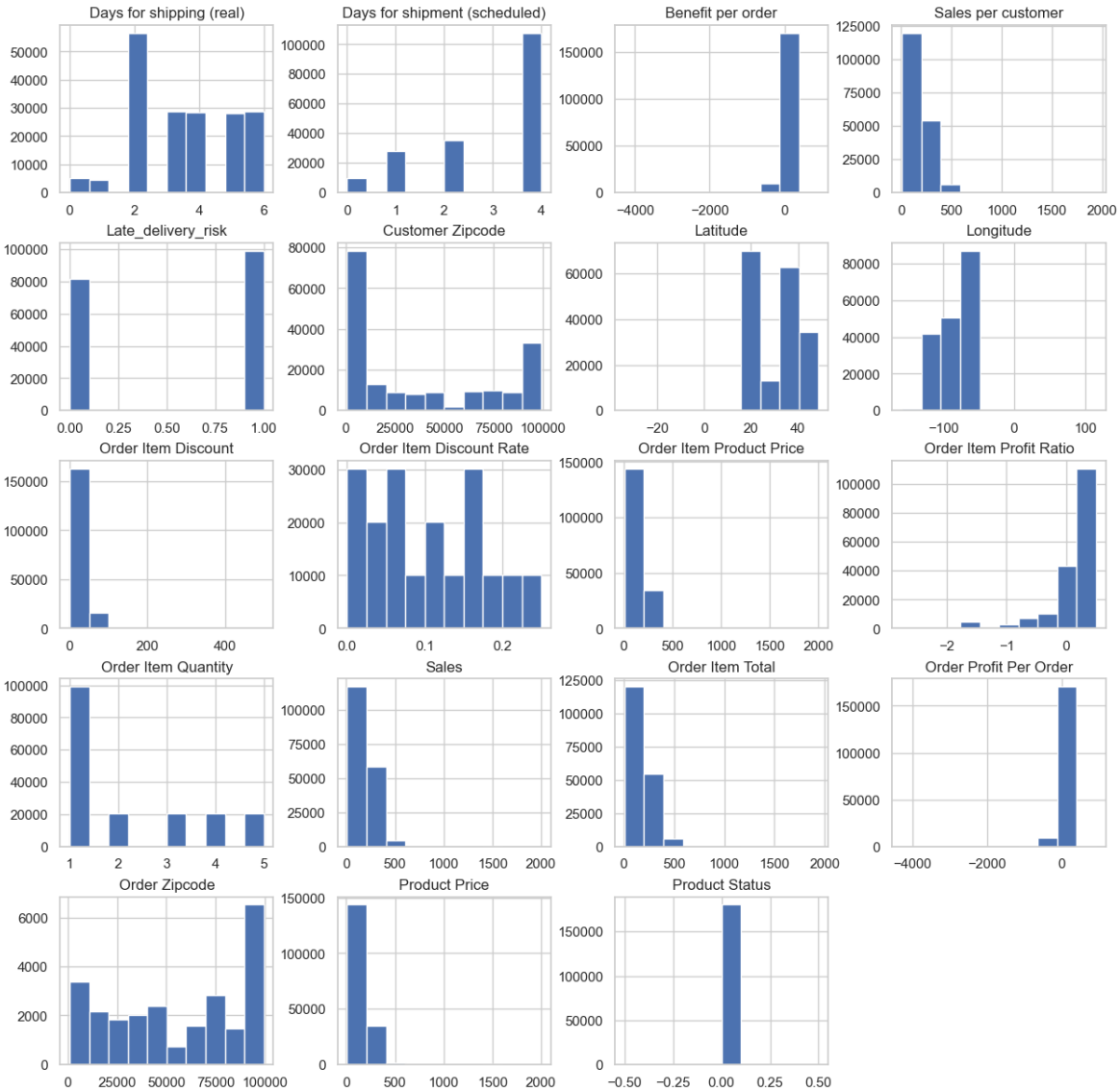
In [47]:
```
scm_df['Shipping Mode'].value_counts()
```

Out[47]:
```
Shipping Mode
Standard Class    107752
Second Class       35216
First Class        27814
Same Day            9737
Name: count, dtype: int64
```

In [ ]:
```
# Let us try to ...
```

In [48]:
```
# plotting a histogram of all the columns

# creating a temporary dataframe for visualizing only those columns which
scm_df_temp_visualization = scm_df.drop(columns=['Category Id',
                                                 'Customer Id',
                                                 'Department Id',
                                                 'Order Customer Id',
                                                 'Order Id',
                                                 'Order Item Cardprod Id'
                                                 'Order Item Id',
                                                 'Product Card Id',
                                                 'Product Category Id',
                                                 'Product Description'])
scm_df_temp_visualization.hist(figsize=(15,15))
```

Out[48]:
```
array([[<Axes: title={'center': 'Days for shipping (real)'}>,
        <Axes: title={'center': 'Days for shipment (scheduled)'}>,
        <Axes: title={'center': 'Benefit per order'}>,
        <Axes: title={'center': 'Sales per customer'}>],
       [<Axes: title={'center': 'Late_delivery_risk'}>,
        <Axes: title={'center': 'Customer Zipcode'}>,
        <Axes: title={'center': 'Latitude'}>,
        <Axes: title={'center': 'Longitude'}>],
       [<Axes: title={'center': 'Order Item Discount'}>,
        <Axes: title={'center': 'Order Item Discount Rate'}>,
        <Axes: title={'center': 'Order Item Product Price'}>,
        <Axes: title={'center': 'Order Item Profit Ratio'}>],
       [<Axes: title={'center': 'Order Item Quantity'}>,
        <Axes: title={'center': 'Sales'}>,
        <Axes: title={'center': 'Order Item Total'}>,
        <Axes: title={'center': 'Order Profit Per Order'}>],
       [<Axes: title={'center': 'Order Zipcode'}>,
        <Axes: title={'center': 'Product Price'}>,
        <Axes: title={'center': 'Product Status'}>, <Axes: >]],
      dtype=object)
```



In [65]:
```
print(*scm_df['Category Name'].unique(), sep='\n')
```

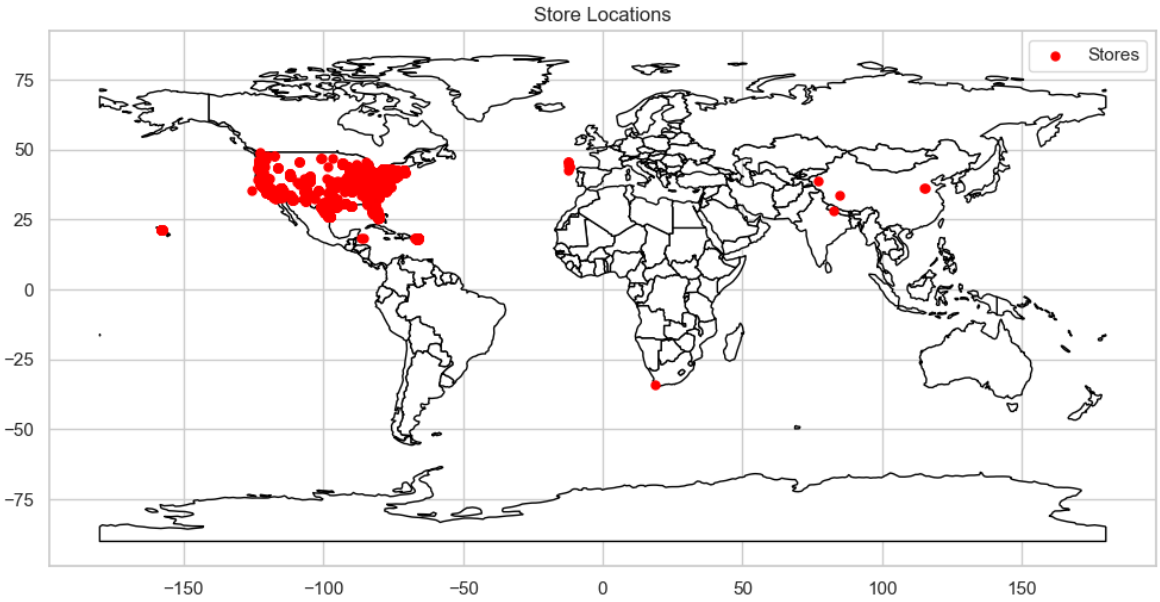
```
Sporting Goods
Cleats
Shop By Sport
Women's Apparel
Electronics
Boxing & MMA
Cardio Equipment
Trade-In
Kids' Golf Clubs
Hunting & Shooting
Baseball & Softball
Men's Footwear
Camping & Hiking
Consumer Electronics
Cameras
Computers
Basketball
Soccer
Girls' Apparel
Accessories
Women's Clothing
Crafts
Men's Clothing
Tennis & Racquet
Fitness Accessories
As Seen on  TV!
Golf Balls
Strength Training
Children's Clothing
Lacrosse
Baby
Fishing
Books
DVDs
CDs
Garden
Hockey
Pet Supplies
Health and Beauty
Music
Video Games
Golf Gloves
Golf Bags & Carts
Golf Shoes
Golf Apparel
Women's Golf Clubs
Men's Golf Clubs
Toys
Water Sports
Indoor/Outdoor Games
```

In [71]:
```
# using the latitude & longitude values from store to plot it on the worl
import geopandas as gpd

# dropping the duplicates and null rows
df_stores_location_lat_n_long = scm_df[['Latitude', 'Longitude']].drop_du

df_geo = gpd.GeoDataFrame(
    df_stores_location_lat_n_long,
    geometry=gpd.points_from_xy(df_stores_location_lat_n_long['Longitude'
```

```
)

#get the map image
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))

#plot the world map
ax = world.plot(figsize=(12, 8), color='white', edgecolor='black')

#plot the store locations as dots
df_geo.plot(ax=ax, marker='o', color='red', markersize=25, label='Stores'

plt.title('Store Locations')
plt.legend(loc='upper right')
```

```
/var/folders/3m/tt_kbz7d2tx3f5cx3v4nd4940000gn/T/ipykernel_11768/282516111
0.py:13: FutureWarning:

The geopandas.dataset module is deprecated and will be removed in GeoPanda
s 1.0. You can get the original 'naturalearth_lowres' data from https://ww
w.naturalearthdata.com/downloads/110m-cultural-vectors/.
```

Out[71]: <matplotlib.legend.Legend at 0x1635320a0>



Model Preparation

Let us start with removing the following columns

1. ID Columns:
    - 'Category Id',
    - 'Customer Id',
    - 'Department Id',
    - 'Order Customer Id',
    - 'Order Id',
    - 'Order Item Cardprod Id',
    - 'Order Item Id',
    - 'Product Card Id',
    - 'Product Category Id',

2. Irrelevant Non-numeric Columns
    - 'Product Description'
    - 'Customer Email'
    - 'Customer Password'
    - 'Customer Fname'
    - 'Customer Lname'
    - 'Customer State'
    - 'Customer Street'
    - 'Customer Zipcode'
    - Order Region
    - Order State
    - Order Status
    - Order Zipcode
    - Product Image
3. Redundant Columns:
    - 'Delivery Status'
    - 'Product Status'
    - 'Customer City'
    - 'Customer Country' (since we already have Order City and Order Country)

In [120…
```
from sklearn import svm, metrics, tree, preprocessing, linear_model
```

In [117…
```
scm_clean_df = scm_df.drop(columns=['Category Id',
                                    'Customer Id',
                                    'Department Id',
                                    'Order Customer Id',
                                    'Order Id',
                                    'Order Item Cardprod Id',
                                    'Order Item Id',
                                    'Product Card Id',
                                    'Product Category Id',
                                    'Product Description',
                                    'Delivery Status',
                                    'Customer Email',
                                    'Customer Password',
                                    'Product Status',
                                    'Customer Fname',
                                    'Customer Lname',
                                    'Customer State',
                                    'Customer Street',
                                    'Customer Zipcode',
                                    'Customer City',
                                    'Customer Country',
                                    'Order Region',
                                    'Order State',
                                    'Order Status',
                                    'Order Zipcode',
                                    'Product Image'])
```

Feature Engineering

Latitude & Longitude for Customer Locations

From visualizing the data, we can see that there are latitude and longitude values for the stores, however, there are no latitude and longitude values for the customer locations.

However, due to complexities and time restrictions this feature engineering is out of scope for this mid-term project, and will be explored for the final project.

In [112…
```
# len(scm_clean_df['Latitude'].unique())
```

In [113…
```
# len(scm_clean_df['Longitude'].unique())
```

In [114…
```
# len(scm_clean_df['Order City'].unique())
```

In [115…
```
# len(scm_clean_df['Order Country'].unique())
```

In [97]:
```
# from geopy.geocoders import Nominatim

# scm_clean_df[['Order Country', 'Order City']].drop_duplicates(inplace=T
# scm_clean_df[['Order Country', 'Order City']].dropna(inplace=True)
# scm_clean_df[['Order Country', 'Order City']].reset_index(inplace=True,

# geolocator = Nominatim(user_agent="city_location") #city and country
# geolocator_country = Nominatim(user_agent="YourAppNameForCountryGeocodi
```

```
/var/folders/3m/tt_kbz7d2tx3f5cx3v4nd4940000gn/T/ipykernel_11768/189440105
5.py:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/var/folders/3m/tt_kbz7d2tx3f5cx3v4nd4940000gn/T/ipykernel_11768/189440105
5.py:4: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [76]:
```
# df_delivery_location_lat_n_long.loc[:,'Delivery Latitude'] = None
# df_delivery_location_lat_n_long.loc[:,'Delivery Longitude'] = None

# for index,row in df_delivery_location_lat_n_long.iterrows():
#     city = row['Order City']
#     country = row['Order Country']

#     if index%500 == 0:
#         print(index)

#     try:
#         location = geolocator.geocode(f"{city}, {country}")

#         df_delivery_location_lat_n_long.loc[index,'Delivery Latitude']
```

```
#         df_delivery_location_lat_n_long.loc[index,'Delivery Longitude']

#     except AttributeError:

#         location = geolocator_country.geocode(country)

#         df_delivery_location_lat_n_long.loc[index,'Delivery Latitude']
#         df_delivery_location_lat_n_long.loc[index,'Delivery Longitude']
```

```
0
500
1000
1500
2000
```

```
---------------------------------------------------------------------------
_
timeout                                   Traceback (most recent call las
t)
File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/url
lib3/connectionpool.py:466, in HTTPConnectionPool._make_request(self, con
n, method, url, timeout, chunked, **httplib_request_kw)
    462         except BaseException as e:
    463             # Remove the TypeError from the exception chain in
    464             # Python 3 (including for exceptions like SystemExit).
    465             # Otherwise it looks like a bug in the code.
--> 466             six.raise_from(e, None)
    467 except (SocketTimeout, BaseSSLError, SocketError) as e:

File <string>:3, in raise_from(value, from_value)

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/url
lib3/connectionpool.py:461, in HTTPConnectionPool._make_request(self, con
n, method, url, timeout, chunked, **httplib_request_kw)
    460 try:
--> 461     httplib_response = conn.getresponse()
    462 except BaseException as e:
    463     # Remove the TypeError from the exception chain in
    464     # Python 3 (including for exceptions like SystemExit).
    465     # Otherwise it looks like a bug in the code.

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/http/client.py:13
48, in HTTPConnection.getresponse(self)
   1347 try:
-> 1348     response.begin()
   1349 except ConnectionError:

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/http/client.py:31
6, in HTTPResponse.begin(self)
    315 while True:
--> 316     version, status, reason = self._read_status()
    317     if status != CONTINUE:

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/http/client.py:27
7, in HTTPResponse._read_status(self)
    276 def _read_status(self):
--> 277     line = str(self.fp.readline(_MAXLINE + 1), "iso-8859-1")
    278     if len(line) > _MAXLINE:

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/socket.py:669, in
SocketIO.readinto(self, b)
    668 try:
--> 669     return self._sock.recv_into(b)
    670 except timeout:

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/ssl.py:1241, in S
SLSocket.recv_into(self, buffer, nbytes, flags)
   1238         raise ValueError(
   1239           "non-zero flags not allowed in calls to recv_into() on %
s" %
   1240           self.__class__)
-> 1241     return self.read(nbytes, buffer)
   1242 else:

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/ssl.py:1099, in S
```

```
SLSocket.read(self, len, buffer)
   1098 if buffer is not None:
-> 1099     return self._sslobj.read(len, buffer)
   1100 else:

timeout: The read operation timed out

During handling of the above exception, another exception occurred:

ReadTimeoutError                          Traceback (most recent call las
t)
File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/url
lib3/connectionpool.py:714, in HTTPConnectionPool.urlopen(self, method, ur
l, body, headers, retries, redirect, assert_same_host, timeout, pool_timeo
ut, release_conn, chunked, body_pos, **response_kw)
    713 # Make the request on the httplib connection object.
--> 714 httplib_response = self._make_request(
    715     conn,
    716     method,
    717     url,
    718     timeout=timeout_obj,
    719     body=body,
    720     headers=headers,
    721     chunked=chunked,
    722 )
    724 # If we're going to release the connection in ``finally:``, then
    725 # the response doesn't need to know about the connection. Otherwis
e
    726 # it will also try to release it and we'll have a double-release
    727 # mess.

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/url
lib3/connectionpool.py:468, in HTTPConnectionPool._make_request(self, con
n, method, url, timeout, chunked, **httplib_request_kw)
    467 except (SocketTimeout, BaseSSLError, SocketError) as e:
--> 468     self._raise_timeout(err=e, url=url, timeout_value=read_timeou
t)
    469     raise

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/url
lib3/connectionpool.py:357, in HTTPConnectionPool._raise_timeout(self, er
r, url, timeout_value)
    356 if isinstance(err, SocketTimeout):
--> 357     raise ReadTimeoutError(
    358         self, url, "Read timed out. (read timeout=%s)" % timeout_v
alue
    359     )
    361 # See the above comment about EAGAIN in Python 3. In Python 2 we h
ave
    362 # to specifically catch it and throw the timeout error

ReadTimeoutError: HTTPSConnectionPool(host='nominatim.openstreetmap.org',
port=443): Read timed out. (read timeout=1)

During handling of the above exception, another exception occurred:

MaxRetryError                             Traceback (most recent call las
t)
File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/req
uests/adapters.py:486, in HTTPAdapter.send(self, request, stream, timeout,
```

```
verify, cert, proxies)
    485 try:
--> 486     resp = conn.urlopen(
    487         method=request.method,
    488         url=url,
    489         body=request.body,
    490         headers=request.headers,
    491         redirect=False,
    492         assert_same_host=False,
    493         preload_content=False,
    494         decode_content=False,
    495         retries=self.max_retries,
    496         timeout=timeout,
    497         chunked=chunked,
    498     )
    500 except (ProtocolError, OSError) as err:

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/url
lib3/connectionpool.py:826, in HTTPConnectionPool.urlopen(self, method, ur
l, body, headers, retries, redirect, assert_same_host, timeout, pool_timeo
ut, release_conn, chunked, body_pos, **response_kw)
    823     log.warning(
    824         "Retrying (%r) after connection broken by '%r': %s", retri
es, err, url
    825     )
--> 826     return self.urlopen(
    827         method,
    828         url,
    829         body,
    830         headers,
    831         retries,
    832         redirect,
    833         assert_same_host,
    834         timeout=timeout,
    835         pool_timeout=pool_timeout,
    836         release_conn=release_conn,
    837         chunked=chunked,
    838         body_pos=body_pos,
    839         **response_kw
    840     )
    842 # Handle redirect?

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/url
lib3/connectionpool.py:826, in HTTPConnectionPool.urlopen(self, method, ur
l, body, headers, retries, redirect, assert_same_host, timeout, pool_timeo
ut, release_conn, chunked, body_pos, **response_kw)
    823     log.warning(
    824         "Retrying (%r) after connection broken by '%r': %s", retri
es, err, url
    825     )
--> 826     return self.urlopen(
    827         method,
    828         url,
    829         body,
    830         headers,
    831         retries,
    832         redirect,
    833         assert_same_host,
    834         timeout=timeout,
    835         pool_timeout=pool_timeout,
```

```
    836         release_conn=release_conn,
    837         chunked=chunked,
    838         body_pos=body_pos,
    839         **response_kw
    840     )
    842 # Handle redirect?

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/url
lib3/connectionpool.py:798, in HTTPConnectionPool.urlopen(self, method, ur
l, body, headers, retries, redirect, assert_same_host, timeout, pool_timeo
ut, release_conn, chunked, body_pos, **response_kw)
    796     e = ProtocolError("Connection aborted.", e)
--> 798 retries = retries.increment(
    799     method, url, error=e, _pool=self, _stacktrace=sys.exc_info()
[2]
    800 )
    801 retries.sleep()

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/url
lib3/util/retry.py:592, in Retry.increment(self, method, url, response, er
ror, _pool, _stacktrace)
    591 if new_retry.is_exhausted():
--> 592     raise MaxRetryError(_pool, url, error or ResponseError(cause))
    594 log.debug("Incremented Retry for (url='%s'): %r", url, new_retry)

MaxRetryError: HTTPSConnectionPool(host='nominatim.openstreetmap.org', por
t=443): Max retries exceeded with url: /search?q=Duque+de+Caxias%2C+Brasil
&format=json&limit=1 (Caused by ReadTimeoutError("HTTPSConnectionPool(host
='nominatim.openstreetmap.org', port=443): Read timed out. (read timeout=
1)"))

During handling of the above exception, another exception occurred:

ConnectionError                           Traceback (most recent call las
t)
File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/geo
py/adapters.py:482, in RequestsAdapter._request(self, url, timeout, header
s)
    481 try:
--> 482     resp = self.session.get(url, timeout=timeout, headers=headers)
    483 except Exception as error:

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/req
uests/sessions.py:602, in Session.get(self, url, **kwargs)
    601 kwargs.setdefault("allow_redirects", True)
--> 602 return self.request("GET", url, **kwargs)

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/req
uests/sessions.py:589, in Session.request(self, method, url, params, data,
headers, cookies, files, auth, timeout, allow_redirects, proxies, hooks, s
tream, verify, cert, json)
    588 send_kwargs.update(settings)
--> 589 resp = self.send(prep, **send_kwargs)
    591 return resp

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/req
uests/sessions.py:703, in Session.send(self, request, **kwargs)
    702 # Send the request
--> 703 r = adapter.send(request, **kwargs)
    705 # Total elapsed time of the request (approximately)
```

```
File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/requests/adapters.py:519, in HTTPAdapter.send(self, request, stream, timeout, verify, cert, proxies)
    517         raise SSLError(e, request=request)
--> 519     raise ConnectionError(e, request=request)
    521 except ClosedPoolError as e:

ConnectionError: HTTPSConnectionPool(host='nominatim.openstreetmap.org', port=443): Max retries exceeded with url: /search?q=Duque+de+Caxias%2C+Brasil&format=json&limit=1 (Caused by ReadTimeoutError("HTTPSConnectionPool(host='nominatim.openstreetmap.org', port=443): Read timed out. (read timeout=1)"))

During handling of the above exception, another exception occurred:

GeocoderUnavailable                       Traceback (most recent call last)
/Users/rishabhkaushick/Documents/Northeastern/INFO6105/MidTermProject/RishLK_MidTermProject.ipynb Cell 52 line 1
      <a href='vscode-notebook-cell:/Users/rishabhkaushick/Documents/Northeastern/INFO6105/MidTermProject/RishLK_MidTermProject.ipynb#Y123sZmlsZQ%3D%3D?line=8'>9</a>     print(index)
     <a href='vscode-notebook-cell:/Users/rishabhkaushick/Documents/Northeastern/INFO6105/MidTermProject/RishLK_MidTermProject.ipynb#Y123sZmlsZQ%3D%3D?line=10'>11</a> try:
---> <a href='vscode-notebook-cell:/Users/rishabhkaushick/Documents/Northeastern/INFO6105/MidTermProject/RishLK_MidTermProject.ipynb#Y123sZmlsZQ%3D%3D?line=11'>12</a>     location = geolocator.geocode(f"{city}, {country}")
     <a href='vscode-notebook-cell:/Users/rishabhkaushick/Documents/Northeastern/INFO6105/MidTermProject/RishLK_MidTermProject.ipynb#Y123sZmlsZQ%3D%3D?line=13'>14</a>     df_delivery_location_lat_n_long.loc[index,'Delivery Latitude'] = location.latitude
     <a href='vscode-notebook-cell:/Users/rishabhkaushick/Documents/Northeastern/INFO6105/MidTermProject/RishLK_MidTermProject.ipynb#Y123sZmlsZQ%3D%3D?line=14'>15</a>     df_delivery_location_lat_n_long.loc[index,'Delivery Longitude']= location.longitude

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/geopy/geocoders/nominatim.py:297, in Nominatim.geocode(self, query, exactly_one, timeout, limit, addressdetails, language, geometry, extratags, country_codes, viewbox, bounded, featuretype, namedetails)
    295 logger.debug("%s.geocode: %s", self.__class__.__name__, url)
    296 callback = partial(self._parse_json, exactly_one=exactly_one)
--> 297 return self._call_geocoder(url, callback, timeout=timeout)

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/geopy/geocoders/base.py:368, in Geocoder._call_geocoder(self, url, callback, timeout, is_json, headers)
    366 try:
    367     if is_json:
--> 368         result = self.adapter.get_json(url, timeout=timeout, headers=req_headers)
    369     else:
    370         result = self.adapter.get_text(url, timeout=timeout, headers=req_headers)

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/geopy/adapters.py:472, in RequestsAdapter.get_json(self, url, timeout, headers)
```

```
    471 def get_json(self, url, *, timeout, headers):
--> 472     resp = self._request(url, timeout=timeout, headers=headers)
    473     try:
    474         return resp.json()

File /opt/homebrew/anaconda3/envs/info6105/lib/python3.8/site-packages/geopy/adapters.py:494, in RequestsAdapter._request(self, url, timeout, headers)
    492         raise GeocoderServiceError(message)
    493     else:
--> 494         raise GeocoderUnavailable(message)
    495 elif isinstance(error, requests.Timeout):
    496     raise GeocoderTimedOut("Service timed out")

GeocoderUnavailable: HTTPSConnectionPool(host='nominatim.openstreetmap.org', port=443): Max retries exceeded with url: /search?q=Duque+de+Caxias%2C+Brasil&format=json&limit=1 (Caused by ReadTimeoutError("HTTPSConnectionPool(host='nominatim.openstreetmap.org', port=443): Read timed out. (read timeout=1)"))
```

In [118…

```
scm_clean_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180519 entries, 0 to 180518
Data columns (total 27 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   Type                            180519 non-null  object 
 1   Days for shipping (real)        180519 non-null  int64  
 2   Days for shipment (scheduled)   180519 non-null  int64  
 3   Benefit per order               180519 non-null  float64
 4   Sales per customer              180519 non-null  float64
 5   Late_delivery_risk              180519 non-null  int64  
 6   Category Name                   180519 non-null  object 
 7   Customer Segment                180519 non-null  object 
 8   Department Name                 180519 non-null  object 
 9   Latitude                        180519 non-null  float64
 10  Longitude                       180519 non-null  float64
 11  Market                          180519 non-null  object 
 12  Order City                      180519 non-null  object 
 13  Order Country                   180519 non-null  object 
 14  order date (DateOrders)         180519 non-null  object 
 15  Order Item Discount             180519 non-null  float64
 16  Order Item Discount Rate        180519 non-null  float64
 17  Order Item Product Price        180519 non-null  float64
 18  Order Item Profit Ratio         180519 non-null  float64
 19  Order Item Quantity             180519 non-null  int64  
 20  Sales                           180519 non-null  float64
 21  Order Item Total                180519 non-null  float64
 22  Order Profit Per Order          180519 non-null  float64
 23  Product Name                    180519 non-null  object 
 24  Product Price                   180519 non-null  float64
 25  shipping date (DateOrders)      180519 non-null  object 
 26  Shipping Mode                   180519 non-null  object 
dtypes: float64(12), int64(4), object(11)
memory usage: 37.2+ MB
```

Text Columns with Categorical Data

- Type
- Category Name
- Customer Segment
- Department Name
- Market
- Order City
- Order Country
- order date (DateOrders)
- Product Name
- shipping date (DateOrders)
- Shipping Mode

In [121…
```
# need to use LabelEncoders to convert the categorical text values

label_encoder = preprocessing.LabelEncoder()
```

In [122…
```
scm_clean_df['Type'] = label_encoder.fit_transform(scm_clean_df['Type'])
scm_clean_df['Category Name'] = label_encoder.fit_transform(scm_clean_df[
scm_clean_df['Customer Segment'] = label_encoder.fit_transform(scm_clean_
scm_clean_df['Department Name'] = label_encoder.fit_transform(scm_clean_d
scm_clean_df['Market'] = label_encoder.fit_transform(scm_clean_df['Market
scm_clean_df['Order City'] = label_encoder.fit_transform(scm_clean_df['Or
scm_clean_df['Order Country'] = label_encoder.fit_transform(scm_clean_df[
scm_clean_df['order date (DateOrders)'] = label_encoder.fit_transform(scm
scm_clean_df['Product Name'] = label_encoder.fit_transform(scm_clean_df['
scm_clean_df['shipping date (DateOrders)'] = label_encoder.fit_transform(
scm_clean_df['Shipping Mode'] = label_encoder.fit_transform(scm_clean_df[
```

In [123…
```
scm_clean_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180519 entries, 0 to 180518
Data columns (total 27 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   Type                            180519 non-null  int64  
 1   Days for shipping (real)        180519 non-null  int64  
 2   Days for shipment (scheduled)   180519 non-null  int64  
 3   Benefit per order               180519 non-null  float64
 4   Sales per customer              180519 non-null  float64
 5   Late_delivery_risk              180519 non-null  int64  
 6   Category Name                   180519 non-null  int64  
 7   Customer Segment                180519 non-null  int64  
 8   Department Name                 180519 non-null  int64  
 9   Latitude                        180519 non-null  float64
 10  Longitude                       180519 non-null  float64
 11  Market                          180519 non-null  int64  
 12  Order City                      180519 non-null  int64  
 13  Order Country                   180519 non-null  int64  
 14  order date (DateOrders)         180519 non-null  int64  
 15  Order Item Discount             180519 non-null  float64
 16  Order Item Discount Rate        180519 non-null  float64
 17  Order Item Product Price        180519 non-null  float64
 18  Order Item Profit Ratio         180519 non-null  float64
 19  Order Item Quantity             180519 non-null  int64  
 20  Sales                           180519 non-null  float64
 21  Order Item Total                180519 non-null  float64
 22  Order Profit Per Order          180519 non-null  float64
 23  Product Name                    180519 non-null  int64  
 24  Product Price                   180519 non-null  float64
 25  shipping date (DateOrders)      180519 non-null  int64  
 26  Shipping Mode                   180519 non-null  int64  
dtypes: float64(12), int64(15)
memory usage: 37.2 MB
```

Now all the values are in either int or float.

In [125…
```
# Next creating the train and test set

# creating a function which splits the data randomly
def split_train_test(data, test_ratio):
    np.random.seed(23) # setting the random number generator's seed will
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

In [126…
```
# here we are splitting the data into 80% training and 20% testing sets.
scm_trainset, scm_testset = split_train_test(scm_clean_df, 0.2)
```

In [127…
```
len(scm_trainset)
```

Out[127… 144416

In [128…
```
len(scm_testset)
```

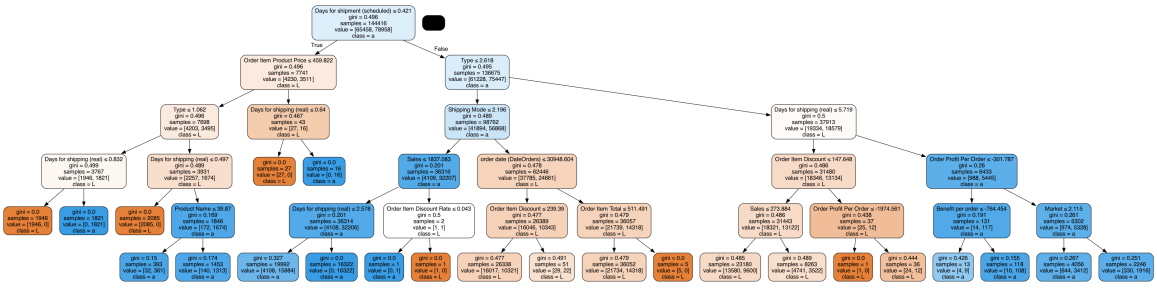
Out[128… 36103

In [133…
```
from sklearn.tree import DecisionTreeClassifier
y = scm_trainset["Late_delivery_risk"]
X = scm_trainset.drop(columns="Late_delivery_risk")

tree_clf = DecisionTreeClassifier(max_depth=5, max_features="sqrt", rando
tree_clf.fit(X, y)
```

Out[133…
```
DecisionTreeClassifier(max_depth=5, max_features='sqrt', random_s
tate=23,
                       splitter='random')
```

In [134…
```
from sklearn.tree import export_graphviz
import pydotplus
# from sklearn.externals.six import StringIO
from six import StringIO
from IPython.display import Image

dot_data = StringIO()
export_graphviz(
        tree_clf,
        out_file=dot_data,
        feature_names= X.columns.values.tolist(),
        class_names= "Late_delivery_risk",
        rounded=True,
        filled=True,
        special_characters=True
    )
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('decision_tree_model_1.png')
Image(graph.create_png())
```

Out[134…



In [135…
```
# The decision tree model has been trained. We can now test it using the

predictions = tree_clf.predict(X)
predictions
```

Out[135…
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [136…
```
# Using mean sequared error to calculate the error percentage
from sklearn.metrics import mean_squared_error
tree_mean_sq_error = mean_squared_error(y, predictions)
tree_root_mean_sq_err = np.sqrt(tree_mean_sq_error)
tree_root_mean_sq_err
```

Out[136…
```
0.5460651806110116
```

Looks like the model is not able to predict the values well. The % of error is 54.6%, which means the train accuracy is 46.4%.

In [138…
```
# Training the a second model on different parameters

tree_clf = DecisionTreeClassifier(max_depth=10, max_features="log2", rand
tree_clf.fit(X, y)
```

Out[138…
```
DecisionTreeClassifier(max_depth=10, max_features='log2', random_
state=5,
                       splitter='random')
```

In [139…
```
from sklearn.tree import export_graphviz
import pydotplus
# from sklearn.externals.six import StringIO
from six import StringIO
from IPython.display import Image

dot_data = StringIO()
export_graphviz(
        tree_clf,
        out_file=dot_data,
        feature_names= X.columns.values.tolist(),
        class_names= "Late_delivery_risk",
        rounded=True,
        filled=True,
        special_characters=True
    )
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('decision_tree_model_2.png')
Image(graph.create_png())
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.658726 to
fit

dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.658726 to
fit
```

Out[139…


In [140…
```
predictions = tree_clf.predict(X)
predictions
```

Out[140…
```
array([0, 0, 0, ..., 0, 1, 0])
```

In [141…
```
tree_mean_sq_error = mean_squared_error(y, predictions)
tree_root_mean_sq_err = np.sqrt(tree_mean_sq_error)
tree_root_mean_sq_err
```

Out[141…
```
0.48176791227108456
```

This has helped us improve the model from 46% to 52%.

In [142…
```
y_test = scm_testset['Late_delivery_risk']
X_test = scm_testset.drop(columns=['Late_delivery_risk'])
test_predictions = tree_clf.predict(X_test)
test_predictions
```

Out[142…
```
array([1, 1, 0, ..., 0, 1, 0])
```

In [144…
```
tree_mean_sq_error = mean_squared_error(y_test, test_predictions)
tree_root_mean_sq_err = np.sqrt(tree_mean_sq_error)
tree_root_mean_sq_err
```

Out[144…
```
0.48137915965998845
```

We can see both train and test accuracy is around 52%.

In [ ]: