ADS 505 - Final Team Project

George Garcia, Summer Purschke, Vannesa Salazar - ADS - 505

In [1]:
```
__author__ = 'George Garcia, Summer Purschke, Vannesa Salazar'
__email__ = 'ggarcia@sandiego.edu, spurschke@sandiego.edu, vsalazar@sandiego.edu'
__version__ = '1.0'
__date__ = 'October 2022'
```

Problem Statement

Due to the Covid-19 pandemic and supply-chain related delays, profit and customer analysis is critical in the realm of E-commerce. The Global Superstore dataset contains 51290 rows of transaction data including customer, shipping, order, profit, and location information. Our goal, at a high level, was to explore this information and recommend actionable changes to improve aspects of the company. More specifically, our team focused on profit associated with products. With a dataset this complex, correlation between product and profit is not as straightforward as one may assume. Products have differing profit margins, in addition to purchase frequency, holding cost, shipping options, and wholesale order reliability. Our team took a machine learning approach to this problem by training predictive models to predict profitability of products in relation to the details surrounding each order. These models, which will be described in detail throughout this report, give Global Superstore the ability to choose which orders to fulfill to maximize their profit margins.

Setup

In [2]:
```
# pip install --upgrade anndata scanpy numpy scipy scikit-learn pandas matplotlib umap-learn h5py statsmodels
```

In [3]:
```
pip install tk
```

```
Requirement already satisfied: tk in /Users/vannesasalazar/opt/anaconda3/lib/python3.9/site-packages (0.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```python
# Basics
import pandas as pd
import numpy as np
import seaborn as sns
import scipy

# Visualization
import matplotlib.pylab as plt
%matplotlib inline

# Modeling
import statsmodels.formula.api as sm
import statsmodels.tools.tools as stattools
from scipy.stats import skew

from dmba import regressionSummary, exhaustive_search
from dmba import backward_elimination, forward_selection, stepwise_selection
from dmba import adjusted_r2_score, AIC_score, BIC_score
from dmba import plotDecisionTree, gainsChart, liftChart
from dmba import classificationSummary, regressionSummary

from sklearn.linear_model import LinearRegression, Lasso, Ridge, LassoCV, BayesianRidge, SGDRegressor
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.svm import SVR
from sklearn.impute import SimpleImputer
from sklearn import tree
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, KFold, cross_val_score
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz, plot_tree
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score, confusion_matrix,r2_score
from sklearn.preprocessing import OrdinalEncoder, StandardScaler, MinMaxScaler
from sklearn import metrics

from wsgiref.simple_server import WSGIRequestHandler

# Set basic options for consistent output
PRECISION = 2
np.set_printoptions(precision = PRECISION)
pd.set_option('display.float_format', lambda x: '%.2f' % x)
pd.set_option('display.precision', PRECISION)
pd.set_option('display.width', 1000)
pd.set_option('display.colheader_justify', 'center')

# Set Matplotlib defaults for consistent visualization look 'n' feel
FONTSIZE_S = 10
FONTSIZE_M = 12
FONTSIZE_L = 14
plt.style.use('default')
plt.rcParams['figure.titlesize'] = FONTSIZE_L
plt.rcParams['figure.figsize'] = (9, 9 / (16 / 9))
plt.rcParams['figure.subplot.left'] = '0.1'
plt.rcParams['figure.subplot.bottom'] = '0.1'
plt.rcParams['figure.subplot.top'] = '0.9'
plt.rcParams['figure.subplot.wspace'] = '0.4'
plt.rcParams['lines.linewidth'] = '2'
plt.rcParams['axes.linewidth'] = '2'
plt.rcParams['axes.titlesize'] = '8'
#plt.rcParams['axes.titleweight'] = 'bold'
plt.rcParams['axes.labelsize'] = FONTSIZE_M
plt.rcParams['xtick.labelsize'] = FONTSIZE_S
plt.rcParams['ytick.labelsize'] = FONTSIZE_S
plt.rcParams['grid.linewidth'] = '1'
plt.rcParams['legend.fontsize'] = FONTSIZE_S
plt.rcParams['legend.title_fontsize'] = FONTSIZE_S
```

Load Data from Github Repository

https://github.com/VSbr22/ADS505B-Fall22-Group-1

In [5]:
```python
df= pd.read_csv('/Users/vannesasalazar/Documents/ADS505/Group Project/Global_Superstore2.csv')
df.head(2)
```

Out[5]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	City	State	...	Product ID	Category	Sub-Category	Pr
0	32298	CA-2012-124891	31-07-2012	31-07-2012	Same Day	RH-19495	Rick Hansen	Consumer	New York City	New York	...	TEC-AC-10003033	Technology	Accessories	Plan C Ove mc
1	26341	IN-2013-77878	05-02-2013	07-02-2013	Second Class	JR-16210	Justin Ritter	Corporate	Wollongong	New South Wales	...	FUR-CH-10003950	Furniture	Chairs	N Exe L Arr

2 rows × 24 columns

Format data

For easier data manipulation, the dataset is formatted so that column names are consistent and will not cause errors within a pandas dataframe. Features are coerced into the correct data types for analysis.

In [6]:
```
# Format column names
df.columns = [d.replace(' ', '_') for d in df.columns]
df.columns = [d.replace('-', '_') for d in df.columns]
```

In [7]:
```
# Format data and time
df['Order_Date'] = pd.to_datetime(df['Order_Date'], infer_datetime_format=True)
df['Ship_Date'] = pd.to_datetime(df['Ship_Date'], infer_datetime_format=True)
```

In [8]:
```
# Change data types as needed
df['Ship_Mode']= df['Ship_Mode'].astype('category')
df['Segment']= df['Segment'].astype('category')
df['Country']= df['Country'].astype('category')
df['Market']= df['Market'].astype('category')
df['Region']= df['Region'].astype('category')
df['Category']= df['Category'].astype('category')
df['Sub_Category']= df['Sub_Category'].astype('category')
df['Order_Priority']= df['Order_Priority'].astype('category')
```

Explore Data

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51290 entries, 0 to 51289
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Row_ID          51290 non-null  int64
 1   Order_ID        51290 non-null  object
 2   Order_Date      51290 non-null  datetime64[ns]
 3   Ship_Date       51290 non-null  datetime64[ns]
 4   Ship_Mode       51290 non-null  category
 5   Customer_ID     51290 non-null  object
 6   Customer_Name   51290 non-null  object
 7   Segment         51290 non-null  category
 8   City            51290 non-null  object
 9   State           51290 non-null  object
 10  Country         51290 non-null  category
 11  Postal_Code     9994 non-null   float64
 12  Market          51290 non-null  category
 13  Region          51290 non-null  category
 14  Product_ID      51290 non-null  object
 15  Category        51290 non-null  category
 16  Sub_Category    51290 non-null  category
 17  Product_Name    51290 non-null  object
 18  Sales           51290 non-null  float64
 19  Quantity        51290 non-null  int64
 20  Discount        51290 non-null  float64
 21  Profit          51290 non-null  float64
 22  Shipping_Cost   51290 non-null  float64
 23  Order_Priority  51290 non-null  category
dtypes: category(8), datetime64[ns](2), float64(5), int64(2), object(7)
memory usage: 6.7+ MB
```

In [10]:
```
df.describe()
```

Out[10]:

	Row_ID	Postal_Code	Sales	Quantity	Discount	Profit	Shipping_Cost
count	51290.00	9994.00	51290.00	51290.00	51290.00	51290.00	51290.00
mean	25645.50	55190.38	246.49	3.48	0.14	28.61	26.38
std	14806.29	32063.69	487.57	2.28	0.21	174.34	57.30
min	1.00	1040.00	0.44	1.00	0.00	-6599.98	0.00
25%	12823.25	23223.00	30.76	2.00	0.00	0.00	2.61
50%	25645.50	56430.50	85.05	3.00	0.00	9.24	7.79
75%	38467.75	90008.00	251.05	5.00	0.20	36.81	24.45
max	51290.00	99301.00	22638.48	14.00	0.85	8399.98	933.57

In [11]:

```
# Skewed predictors
print('Sales Skew:', skew(df['Sales'], axis = 0, bias = True ))
print('Quantity Skew:', skew(df['Quantity'], axis = 0, bias = True ))
print('Discount Skew:', skew(df['Discount'], axis = 0, bias = True ))
print('Profit Skew:', skew(df['Profit'], axis = 0, bias = True ))
print('Shipping Cost Skew:', skew(df['Shipping_Cost'], axis = 0, bias = True ))
```

```
Sales Skew: 8.137842017336732
Quantity Skew: 1.3603279457897046
Discount Skew: 1.3877339656893208
Profit Skew: 4.157066952869827
Shipping Cost Skew: 5.863054951522988
```

In [12]:

```
# Missing values
df.isnull().sum(axis = 0)
```

Out[12]:

```
Row_ID                0
Order_ID              0
Order_Date            0
Ship_Date             0
Ship_Mode             0
Customer_ID           0
Customer_Name         0
Segment               0
City                  0
State                 0
Country               0
Postal_Code       41296
Market                0
Region                0
Product_ID            0
Category              0
Sub_Category          0
Product_Name          0
Sales                 0
Quantity              0
Discount              0
Profit                0
Shipping_Cost         0
Order_Priority        0
dtype: int64
```

In [13]:

```
# Least profitable products
df.groupby(['Product_Name']).sum()[['Profit']].nsmallest(n=5, columns=['Profit'])
```

```
/var/folders/5_/tllj2d8s5md8mp1n3qryzzhh0000gn/T/ipykernel_11368/304467740.py:2: FutureWarning: The default valu
e of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False
. Either specify numeric_only or select only columns which should be valid for the function.
  df.groupby(['Product_Name']).sum()[['Profit']].nsmallest(n=5, columns=['Profit'])
```

Out[13]:

	Profit
Product_Name	
Cubify CubeX 3D Printer Double Head Print	-8879.97
Lexmark MX611dhe Monochrome Laser Printer	-4589.97
Motorola Smart Phone, Cordless	-4447.04
Cubify CubeX 3D Printer Triple Head Print	-3839.99
Bevis Round Table, Adjustable Height	-3649.89

In [14]:

```
# Most profitable products
df.groupby(['Product_Name']).sum()[['Profit']].nlargest(n=5, columns=['Profit'])
```

```
/var/folders/5_/tllj2d8s5md8mp1n3qryzzhh0000gn/T/ipykernel_11368/3107184226.py:2: FutureWarning: The default val
ue of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to Fals
e. Either specify numeric_only or select only columns which should be valid for the function.
  df.groupby(['Product_Name']).sum()[['Profit']].nlargest(n=5, columns=['Profit'])
```

Out[14]:

	Profit
Product_Name	
Canon imageCLASS 2200 Advanced Copier	25199.93
Cisco Smart Phone, Full Size	17238.52
Motorola Smart Phone, Full Size	17027.11
Hoover Stove, Red	11807.97
Sauder Classic Bookcase, Traditional	10672.07

In [72]:
```
# # Proportion of Profitable to Non-Profitable
# df.groupby(['Product_Name']).sum()[['Profit']]

# p = len(df.groupby(['Product_Name']).sum()[['Profit']].query('Profit > 0'))
# np = len(df.groupby(['Product_Name']).sum()[['Profit']].query('Profit <=0'))

# prop = round((p/(p+np)) *100, 2)

# print(prop, '% of products are profitable')
```

In [16]:
```
# Total loss from non profitable products
loss = df.groupby(['Product_Name']).sum()[['Profit']].query('Profit <=0').sum()
gain = df.groupby(['Product_Name']).sum()[['Profit']].query('Profit > 0').sum()
loss, gain

print('Average Profit of profitable products')
print(df.groupby(['Product_Name']).sum()[['Profit']].query('Profit > 0').mean())

print('Number of Unprofitable products:',
      df.groupby(['Product_Name']).sum()[['Profit']].query('Profit <= 0').count())
```

```
Average Profit of profitable products
Profit    551.90
dtype: float64
Number of Unprofitable products: Profit    680
dtype: int64
```

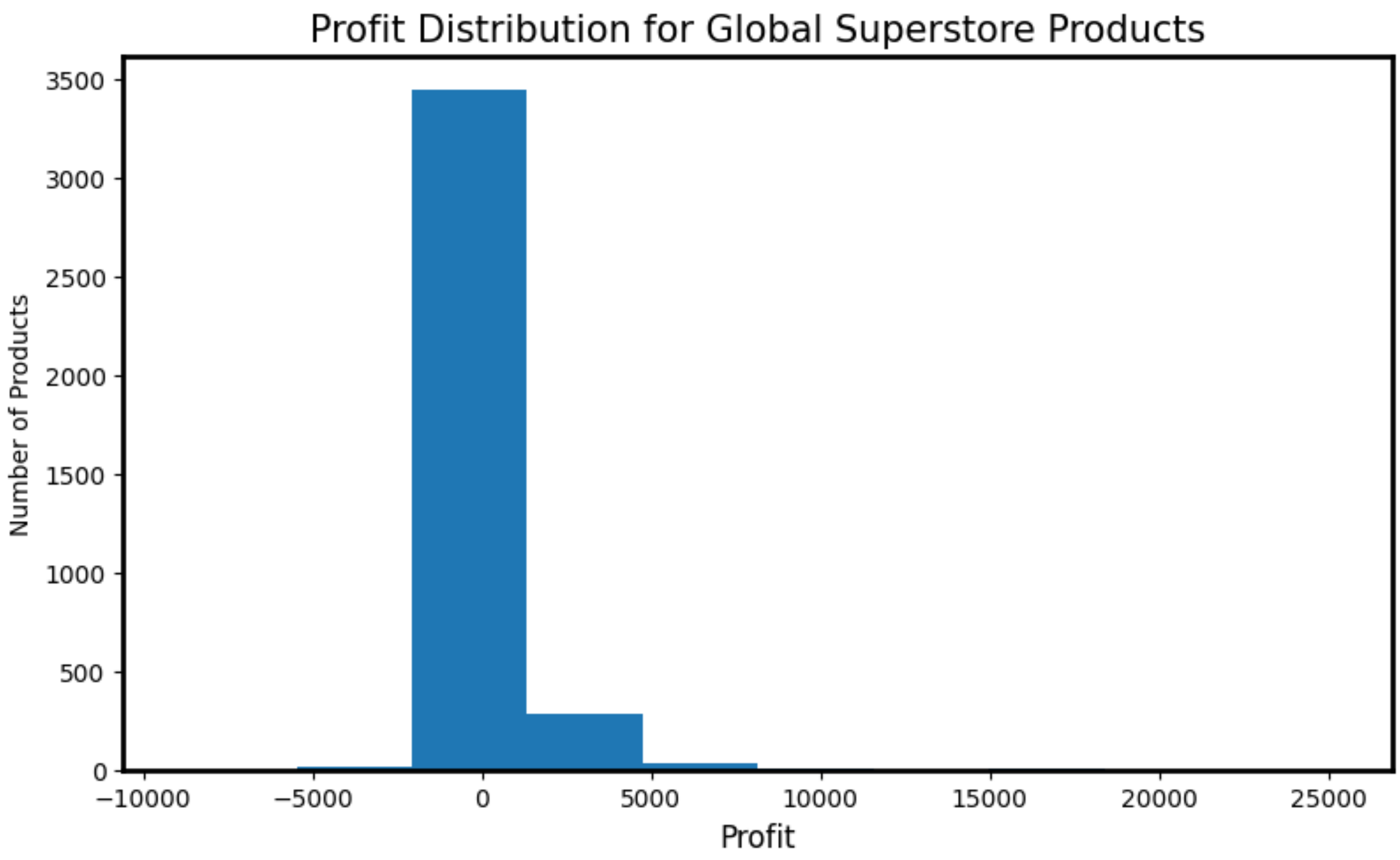
```
/var/folders/5_/tllj2d8s5md8mp1n3qryzzhh0000gn/T/ipykernel_11368/269765817.py:2: FutureWarning: The default valu
e of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False
. Either specify numeric_only or select only columns which should be valid for the function.
  loss = df.groupby(['Product_Name']).sum()[['Profit']].query('Profit <=0').sum()
/var/folders/5_/tllj2d8s5md8mp1n3qryzzhh0000gn/T/ipykernel_11368/269765817.py:3: FutureWarning: The default valu
e of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False
. Either specify numeric_only or select only columns which should be valid for the function.
  gain = df.groupby(['Product_Name']).sum()[['Profit']].query('Profit > 0').sum()
/var/folders/5_/tllj2d8s5md8mp1n3qryzzhh0000gn/T/ipykernel_11368/269765817.py:7: FutureWarning: The default valu
e of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False
. Either specify numeric_only or select only columns which should be valid for the function.
  print(df.groupby(['Product_Name']).sum()[['Profit']].query('Profit > 0').mean())
/var/folders/5_/tllj2d8s5md8mp1n3qryzzhh0000gn/T/ipykernel_11368/269765817.py:10: FutureWarning: The default val
ue of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to Fals
e. Either specify numeric_only or select only columns which should be valid for the function.
  df.groupby(['Product_Name']).sum()[['Profit']].query('Profit <= 0').count())
```

In [17]:
```
a = (df.groupby(['Product_Name']).sum()[['Profit']]).sort_values(by = 'Profit')

plt.hist(a)
plt.yticks()
plt.xlabel('Profit', fontsize=12)
plt.ylabel('Number of Products', fontsize=10)
plt.title('Profit Distribution for Global Superstore Products', fontsize=15)
```

```
/var/folders/5_/tllj2d8s5md8mp1n3qryzzhh0000gn/T/ipykernel_11368/239680151.py:1: FutureWarning: The default valu
e of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False
. Either specify numeric_only or select only columns which should be valid for the function.
  a = (df.groupby(['Product_Name']).sum()[['Profit']]).sort_values(by = 'Profit')
```

Out[17]:
```
Text(0.5, 1.0, 'Profit Distribution for Global Superstore Products')
```



The above cells show us that only 82% of products are profitable, meaning that about 682 products cause a loss in profit the majority of the time. By eliminating these products, and replacing them with products that profit (on average) the same as other products we can recover $374,680 in lost value.

In [18]:
```
# How many product ID's are there? How many Product Names?
print('There are', df['Product_ID'].nunique(), 'unique product IDs, and',
      df['Product_Name'].nunique(), ' unique product names')

group = pd.DataFrame(df.groupby('Product_Name')['Product_ID'].unique())
group['Product_ID'] = group['Product_ID'].astype('str')
group['#Product_IDs'] = group['Product_ID'].str.split(' ').apply(len)

products  = group.sort_values(by = ['#Product_IDs'], ascending = False)
products.head()
```

```
There are 10292 unique product IDs, and 3788  unique product names
```

Out[18]:

Product_Name	Product_ID	#Product_IDs
Staples	['OFF-EN-10004773' 'OFF-EN-10003286' 'OFF-PA-1...	46
Stockwell Paper Clips, Assorted Sizes	['OFF-FA-10002017' 'OFF-FA-10004265' 'OFF-FA-1...	16
Acco Index Tab, Clear	['OFF-BI-10002738' 'OFF-BI-10002386' 'OFF-BI-1...	12
HP Copy Machine, Color	['TEC-CO-10004563' 'TEC-CO-10003901' 'TEC-HP -...	12
Stockwell Push Pins, 12 Pack	['OFF-FA-10002577' 'OFF-FA-10002790' 'OFF-FA-1...	11

In [19]:
```
# How many customer ID's are there? How many customer Names?
print('There are', df['Customer_ID'].nunique(), 'unique Customer IDs, and',
      df['Customer_Name'].nunique(), ' unique Customer names')

group1 = pd.DataFrame(df.groupby('Customer_Name')['Customer_ID'].unique())
group1['Customer_ID'] = group1['Customer_ID'].astype('str')
group1['#Customer_IDs'] = group1['Customer_ID'].str.split(' ').apply(len)

customers  = group1.sort_values(by = ['#Customer_IDs'], ascending = False)
print(customers['#Customer_IDs'].mean())
# It seems that there are two Customer ID's for each Name - Redundant
```

```
There are 1590 unique Customer IDs, and 795  unique Customer names
2.0
```

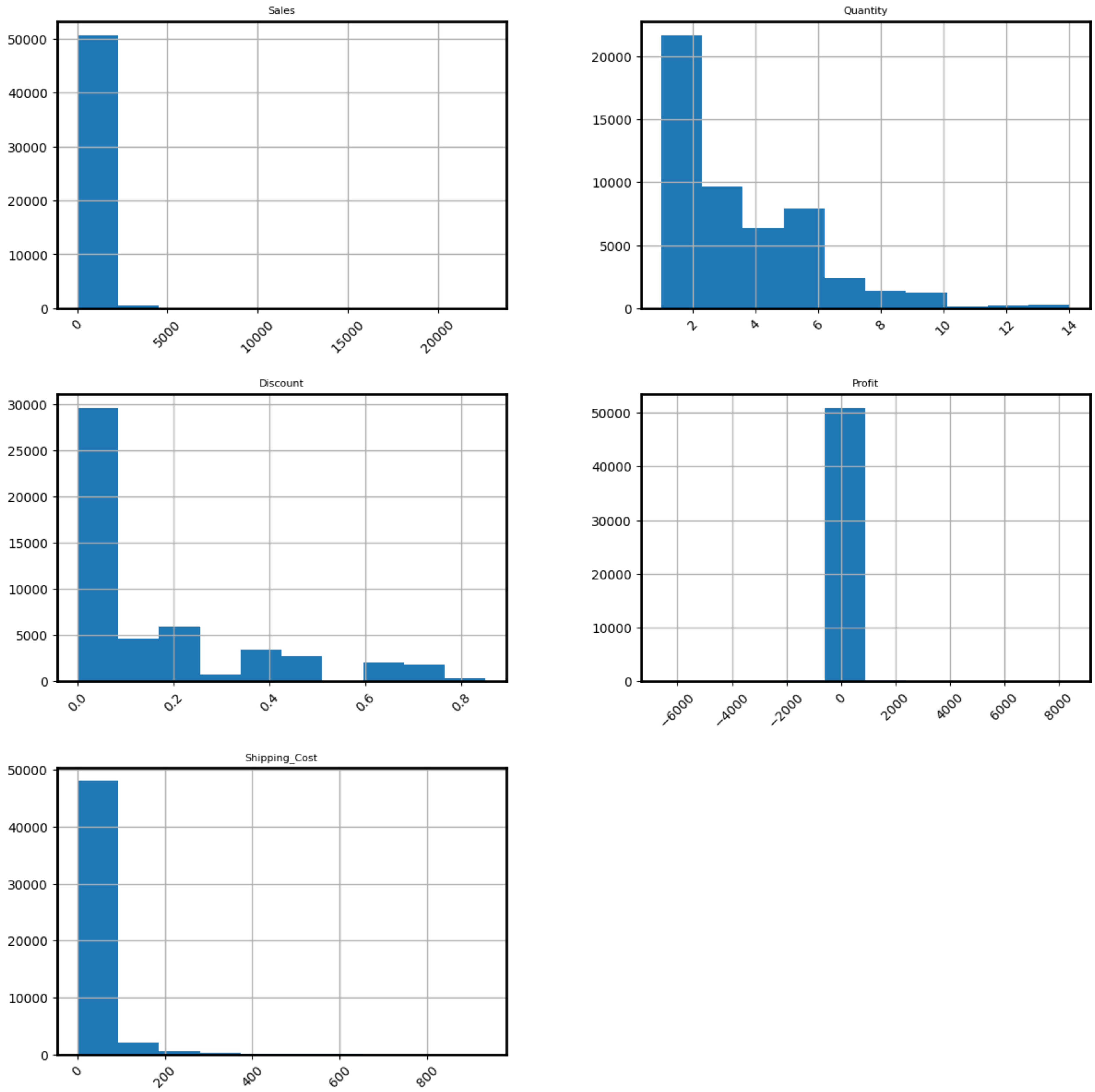
Data Visualization

In [20]:
```
# Feature Selection, comment out features to drop
numeric = df[[
    'Sales',
    'Quantity',
    'Discount',
    'Profit',
    'Shipping_Cost'
]]

ordinal = pd.DataFrame(df['Order_Priority'])

categorical = df[[
#     'Row_ID',
#     'Order_ID',
#     'Order_Date',
#     'Ship_Date',
#     'Ship_Mode',
#     'Customer_ID',
#     'Customer_Name',
    'Segment',
    'City',
    'State',
    'Country',
#     'Postal_Code',
    'Market',
    'Region',
#     'Product_ID',
    'Category',
    'Sub_Category',
    'Product_Name'
]]
```

In [21]:
```
# Numeric histograms
numeric.hist(figsize=(14,14), xrot=45)
plt.show()
```



In [22]:

```
# Categorical Count plots
for column in df.select_dtypes(include='object'):
    if df[column].nunique() < 10:
        sns.countplot(y=column, data=df)
        plt.show()
```

In [23]:

```
# Categorical Box plots
for column in df.select_dtypes(include='object'):
    if df[column].nunique() < 10:
        sns.boxplot(y=column, x='Profit', data=df)
        plt.show()
```

In [24]:

```
# Correlations among numeric predictors
sns.heatmap(numeric.corr(), annot = True, cmap= 'coolwarm')
```

Out[24]: <AxesSubplot: >

Data Preprocessing

Data is prepared for analysis in varying ways based on the type of data. The ordinal feature Order_priority was encoded, numeric values were normalized using MinMaxScaler(), and categorical features were dummy encoded. Once all respective preprocessing was complete the features were merged back into a complete dataframe using the pandas concat function. The code block above the visualizations allows users to select which features are included in modeling by commenting out whichever features should be dropped.

In [25]:
```
# Ordinal Encoding
dict = {'Critical': 4, 'Medium' : 3, 'High' : 2, 'Low': 1}
ordinal = ordinal.replace({'Order_Priority': dict})
ordinal.head(2)
```

Out[25]:

	Order_Priority
0	4
1	4

In [26]:
```
# create binary variable that = 1 if positive profit and = 0 if negative profit
profitable = pd.DataFrame(np.where(df['Profit'] > 0, 1, 0), columns = ['Profitable'])
profitable.head()
```

Out[26]:

	Profitable
0	1
1	0
2	1
3	0
4	1

In [27]:
```
# categorical variables - dummy encode
s1 = pd.get_dummies(categorical)

#numerical variables - normalize
s2 = pd.DataFrame(MinMaxScaler().fit_transform(numeric), columns = numeric.columns)

# # create binary variable that = 1 if positive profit and = 0 if negative profit
s3 = pd.DataFrame(np.where(df['Profit'] > 0, 1, 0), columns = ['Profitable'])
```

In [28]:
```
# Add together chosen and preprocessed features
df = (pd.concat([s1,s2, s3], axis = 1)).dropna()
df.head()
```

Out[28]:

	Segment_Consumer	Segment_Corporate	Segment_Home Office	City_Aachen	City_Aalen	City_Aalst	City_Aba	City_Abadan	City_Abakaliki
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0

5 rows × 8714 columns

Modeling

Our goal is to predict whether or not a product is worth selling, based on the expected profit or loss. To find if a product is worth selling, predictive models were built and tested based on predictive accuracy. Those models with initial accuracy worth exploring were tuned via hyper parameter tuning to improve predictive performance. Ultimately, our team decided to move forward with Logistic Regression, based on it's performance and interpretability. The use of predictive modeling in this application will help stakeholders decide if future product opportunites are likely to bring in a profit or loss in revenue.

Classification - Predicting if an occurrence is or is not profitable

In [29]:
```
# Split into X/ Y
X = df.drop(columns=['Profitable','Profit'])
yc = df['Profitable']    # regression target
yr = df['Profit']        # classification target

#Split into Train/Test - Regression and Classification
Xr_train, Xr_valid, yr_train, yr_valid= train_test_split(X, yr, test_size=0.2, random_state=47)
Xc_train, Xc_valid, yc_train, yc_valid= train_test_split(X, yc, test_size=0.2, random_state=47)
```

Logistic Regression

In [30]:
```
# Logistic Regression - Classification
logr = LogisticRegression(penalty = 'l2', C = 1e12, solver = 'liblinear').fit(Xc_train, yc_train)

# obtaining accuracy scores for logistic regression model
print(f'training model score:{logr.score(Xc_train, yc_train)}')
print(f'test model score:{logr.score(Xc_valid, yc_valid)}')
```

```
training model score:0.9708032754922987
test model score:0.9205498147787093
```

K-Nearest Neighbor

In [31]:
```
k = 4
knn = KNeighborsClassifier(n_neighbors = k).fit(Xc_train, yc_train)
pred_y = knn.predict(Xc_valid)
print("Accuracy of model at K=4 is",metrics.accuracy_score(yc_valid, pred_y))
```

```
Accuracy of model at K=4 is 0.9011503217001364
```

In [32]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(Xc_train, yc_train)
pred_y = knn.predict(Xc_valid)
print("Accuracy of model at K=5 is",metrics.accuracy_score(yc_valid, pred_y))
```

```
Accuracy of model at K=5 is 0.9103139013452914
```

In [33]:
```
scores = cross_val_score(knn, Xc_train, yc_train, cv=10, scoring='accuracy')
print(scores)
```

```
[0.9  0.91 0.91 0.91 0.91 0.92 0.91 0.92 0.91 0.91]
```

In [34]:
```
print(scores.mean())
```

```
0.911654579043813
```

Regression - Predicting Profit Value

Support Vector Machine

In [35]:
```
# Support Vector Machine -  Regression
svm = SVR(C = 2, kernel = 'linear', max_iter = 1000).fit(Xr_train, yr_train)

# obtaining accuracy scores for logistic regression model
print(f'training model score:{svm.score(Xr_train, yr_train)}')
print(f'test model score:{svm.score(Xr_valid, yr_valid)}')
```

```
/Users/vannesasalazar/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:301: ConvergenceWarning: So
lver terminated early (max_iter=1000).  Consider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
training model score:-9.778150347063889
test model score:-14.582610482981037
```

In [36]:
```
# filtering with query method
(pd.DataFrame(df.query('Profitable == 1')))
```

Out[36]:

	Segment_Consumer	Segment_Corporate	Segment_Home Office	City_Aachen	City_Aalen	City_Aalst	City_Aba	City_Abadan	City_Abal
0	1	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	
4	1	0	0	0	0	0	0	0	
5	0	1	0	0	0	0	0	0	
6	1	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	
51284	0	0	1	0	0	0	0	0	
51285	0	1	0	0	0	0	0	0	
51287	0	0	1	0	0	0	0	0	
51288	0	0	1	0	0	0	0	0	
51289	1	0	0	0	0	0	0	0	

38078 rows × 8714 columns

XGboost- Predicting Profit customers

In [37]:
```
#making a copy of the data
df9 = pd.read_csv('https://raw.githubusercontent.com/VSbr22/ADS505B-Fall22-Group-1/main/Global_Superstore2.csv')
```

In [38]:
```
#order data, shipping data dont add inforamation beside when they bought the iteam?
#orderID and RowID don't hould any further information that the customer name dosent give?
df9 = df9.drop(df9.columns[[0, 1, 2, 3]],axis = 1)
```

In [39]:
```
#filter the data fram for only point to point (profit >0 )
pf = df9[df9['Profit'] > 50]
```

In [40]:
```
pf.tail()
```

Out[40]:

	Ship Mode	Customer ID	Customer Name	Segment	City	State	Country	Postal Code	Market	Region	Product ID	Category	Su Catego
50771	Second Class	SJ-20215	Sarah Jordon	Consumer	Mexico City	Distrito Federal	Mexico	NaN	LATAM	North	OFF-SU-10001722	Office Supplies	Suppl
50778	Same Day	CR-12820	Cyra Reiten	Home Office	Vienna	Vienna	Austria	NaN	EU	Central	OFF-ST-10000624	Office Supplies	Stora
50789	Standard Class	AJ-10795	Anthony Johnson	Corporate	San Francisco	California	United States	94110.00	US	West	OFF-LA-10001474	Office Supplies	Lab
50974	Standard Class	KW-6570	Kelly Williams	Consumer	Cairo	Al Qahirah	Egypt	NaN	Africa	Africa	FUR-SAF-10002314	Furniture	Cha
51178	Standard Class	JM-5250	Janet Martin	Consumer	Bur Sudan	Red Sea	Sudan	NaN	Africa	Africa	TEC-MOT-10001088	Technology	Phor

In [41]:
```
#Assigning input features
f = ['Ship Mode', 'Segment', 'Market', 'Category', 'Sales','Product Name']
#assigning output features
y = pf["Profit"]
yq = pf["Quantity"]
```

In [42]:
```
x = pf[f]
x.head()
```

Out[42]:

	Ship Mode	Segment	Market	Category	Sales	Product Name
0	Same Day	Consumer	US	Technology	2309.65	Plantronics CS510 - Over-the-Head monaural Wir...
2	First Class	Consumer	APAC	Technology	5175.17	Nokia Smart Phone, with Caller ID
4	Same Day	Consumer	Africa	Technology	2832.96	Sharp Wireless Fax, High-Speed
5	Second Class	Corporate	APAC	Technology	2862.68	Samsung Smart Phone, with Caller ID
6	First Class	Consumer	APAC	Furniture	1822.08	Novimex Executive Leather Armchair, Adjustable

In [43]:
```
#transformation of cat data and allowing the interger to pass through
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder (sparse = False)
from sklearn.compose import make_column_transformer
from IPython.display import display_html
column_trans = make_column_transformer(
    (OneHotEncoder(), ['Ship Mode', 'Segment', 'Market', 'Category','Product Name']),
    remainder = 'passthrough')
```

In [44]:
```
#apply the transformations
xt = column_trans.fit_transform(x)
```

In [45]:
```
#making the model
#packages that are needed
from scipy.stats import uniform, randint
import xgboost as xgb
from sklearn.metrics import auc, accuracy_score, confusion_matrix, mean_squared_error
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold, RandomizedSearchCV
```

In [46]:
```
#making a def for Accuracy
def display_scores(scores):
    print("Scores: {0}\nMean: {1:.3f}\nStd: {2:.3f}".format(scores, np.mean(scores), np.std(scores)))
```

In [47]:
```
def report_best_scores(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
```

In [48]:
```
#splitting up te data
X_train, X_test, y_train, y_test = train_test_split(xt, y, test_size=0.7, random_state=1)
```

In [49]:
```
#runnign the model
xgb_model = xgb.XGBRegressor(objective="reg:linear", random_state=42)

xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)

mse=mean_squared_error(y_test, y_pred)

print(np.sqrt(mse))
```

```
[22:59:35] WARNING: /Users/runner/work/xgboost/xgboost/python-package/build/temp.macosx-10.9-x86_64-cpython-37/x
gboost/src/objective/regression_obj.cu:203: reg:linear is now deprecated in favor of reg:squarederror.
174.28680737800502
```

Hyperparameter tuning the XGBoost

In [50]:
```
#using the CV to get the parameter for the XGBoost
xgb_model = xgb.XGBRegressor()

params = {
    "colsample_bytree": uniform(0.7, 0.3),
    "gamma": uniform(0, 0.5),
    "learning_rate": uniform(0.03, 0.3), # default 0.1
    "max_depth": randint(2, 6), # default 3
    "n_estimators": randint(100, 150), # default 100
    "subsample": uniform(0.6, 0.4)
}

search = RandomizedSearchCV(xgb_model, param_distributions=params, random_state=42, n_iter=200, cv=5,
                            verbose=1, n_jobs=1, return_train_score=True)

search.fit(X_train, y_train)

report_best_scores(search.cv_results_, 1)
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
Model with rank: 1
Mean validation score: 0.718 (std: 0.103)
Parameters: {'colsample_bytree': 0.9575076414441159, 'gamma': 0.16297945260094238, 'learning_rate': 0.0960723142
6966449, 'max_depth': 3, 'n_estimators': 102, 'subsample': 0.9238004184558861}
```

In [51]:
```
#fitting the best model with the best parameter
xgb_model = xgb.XGBRegressor()

params = {
    'colsample_bytree': 0.9575076414441159,
    'gamma': 0.16297945260094238,
    'learning_rate': 0.09607231426966449,
    'max_depth': 3,
    'n_estimators': 102,
    'subsample': 0.9238004184558861
}
xgb_model.fit(X_train, y_train)
```

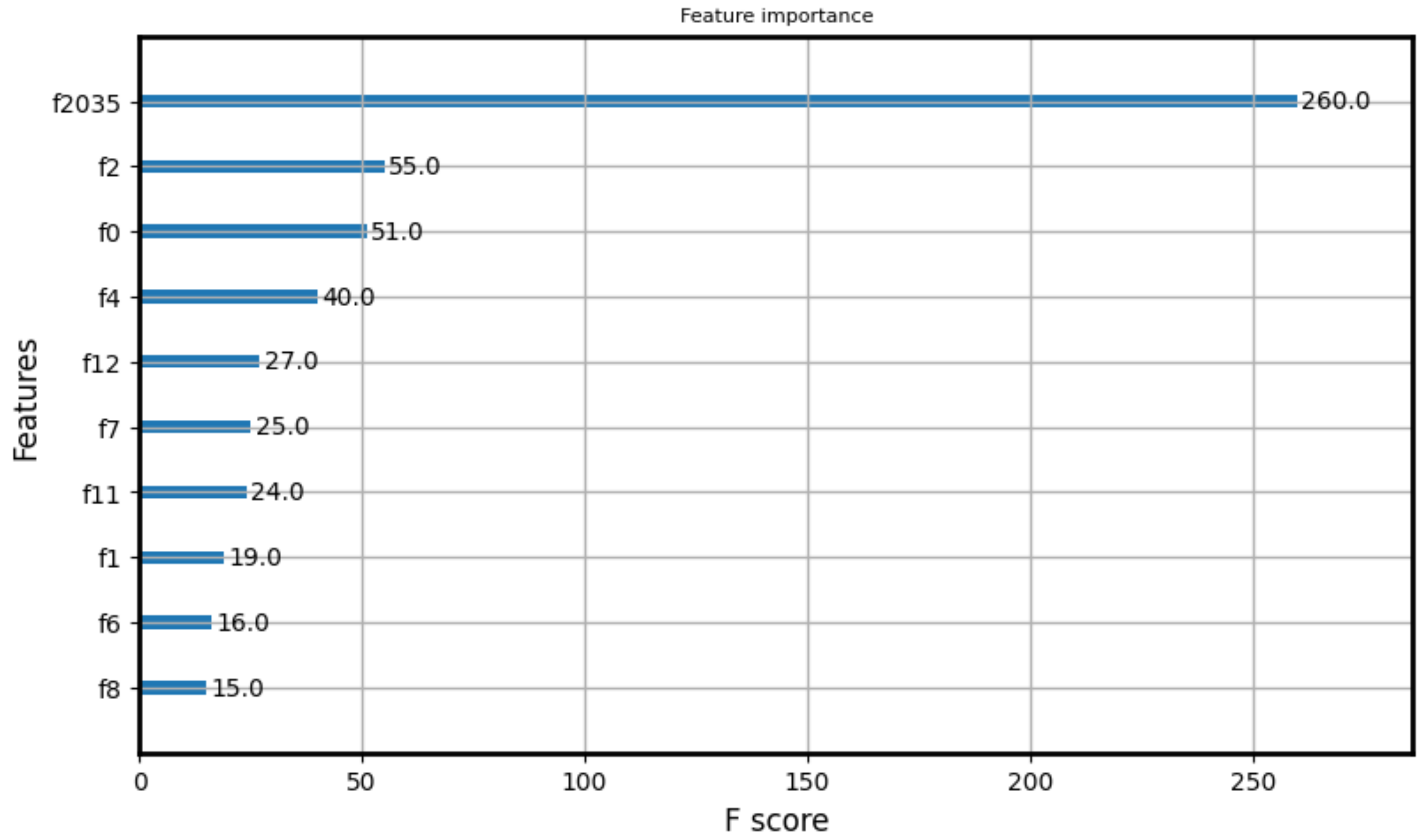
Out[51]:

▾ XGBRegressor

```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, ...)
```

In [52]:
```
xgb.plot_importance(xgb_model, max_num_features=10)
```

Out[52]:
```
<AxesSubplot: title={'center': 'Feature importance'}, xlabel='F score', ylabel='Features'>
```



XGBoost for predicting if a product is Profitable

In [53]:
```
# Combininng the data frams
xt2 = (pd.concat([df9,profitable,], axis = 1)).dropna()
```

In [54]:
```
xt2.head()
```

Out[54]:

	Ship Mode	Customer ID	Customer Name	Segment	City	State	Country	Postal Code	Market	Region	...	Category	Sub-Category	
0	Same Day	RH-19495	Rick Hansen	Consumer	New York City	New York	United States	10024.00	US	East	...	Technology	Accessories	Pl C
8	Standard Class	JW-15220	Jane Waco	Corporate	Sacramento	California	United States	95823.00	US	West	...	Office Supplies	Binders	Co
9	Second Class	JH-15985	Joseph Holt	Consumer	Concord	North Carolina	United States	28027.00	US	South	...	Furniture	Tables	Ch E W Co
10	Second Class	GM-14695	Greg Maxwell	Corporate	Alexandria	Virginia	United States	22304.00	US	South	...	Office Supplies	Supplies	M Le
16	Second Class	TB-21175	Thomas Boland	Corporate	Henderson	Kentucky	United States	42420.00	US	South	...	Technology	Accessories	I

5 rows × 21 columns

In [55]:
```
#Setting s index for input features
p=['Product Name',"Shipping Cost",'Sales']
#output feature
l= ['Profitable']
```

In [56]:
```
#assigning variable
xf= xt2[p]
y3 = xt2[l]
```

In [57]:
```
#transformation of cat data and allowing the interger to pass through
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder (sparse = False)
from sklearn.compose import make_column_transformer
column_trans = make_column_transformer(
    (OneHotEncoder(), ['Product Name']),
    remainder = 'passthrough')
```

In [58]:
```
#apply the transformations
dx = column_trans.fit_transform(xf)
```

In [59]:
```
#splitting up te data
X1_train, X1_test, y1_train, y1_test = train_test_split(dx, y3 , test_size=0.7, random_state=1)
```

making a xgboost model to i dentified the product that are profitable

In [60]:
```
#packages needed
import xgboost as xgb

xgb_cl = xgb.XGBClassifier()
```

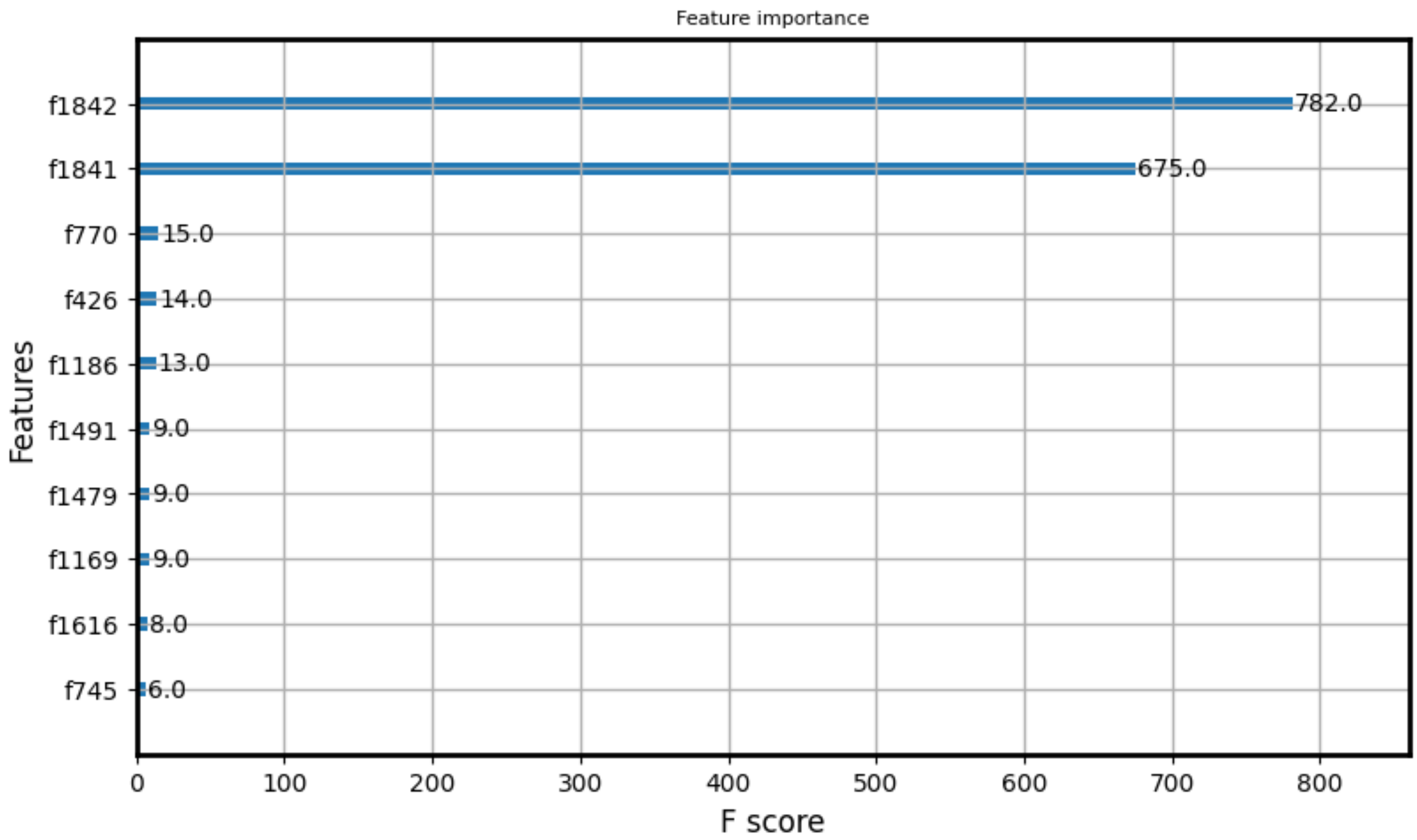
In [61]:
```
# Fit
xgb_cl.fit(X1_train, y1_train)

# Predict
preds = xgb_cl.predict(X1_test)

# Score
accuracy_score(y1_test, preds)
```

Out[61]:
```
0.7890222984562607
```

In [62]:
```
#the top ten
from xgboost import plot_importance
plot_importance(xgb_cl, max_num_features=10) # top 10 product names
plt.show()
```



Hyperparameter tuning the model

In [63]:
```
xgb_cl = xgb.XGBClassifier()

params = {
    "colsample_bytree": uniform(0.7, 0.3),
    "gamma": uniform(0, 0.5),
    "learning_rate": uniform(0.03, 0.3), # default 0.1
    "max_depth": randint(2, 6), # default 3
    "n_estimators": randint(100, 150), # default 100
    "subsample": uniform(0.6, 0.4)
}

search = RandomizedSearchCV(xgb_cl, param_distributions=params, random_state=42, n_iter=100, cv=3,
                            verbose=1, n_jobs=1, return_train_score=True)

search.fit(X1_train, y1_train)

report_best_scores(search.cv_results_, 1)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Model with rank: 1
Mean validation score: 0.815 (std: 0.003)
Parameters: {'colsample_bytree': 0.8405981925982379, 'gamma': 0.028151637840918675, 'learning_rate': 0.065645374
88042158, 'max_depth': 2, 'n_estimators': 140, 'subsample': 0.8976846627773192}
```

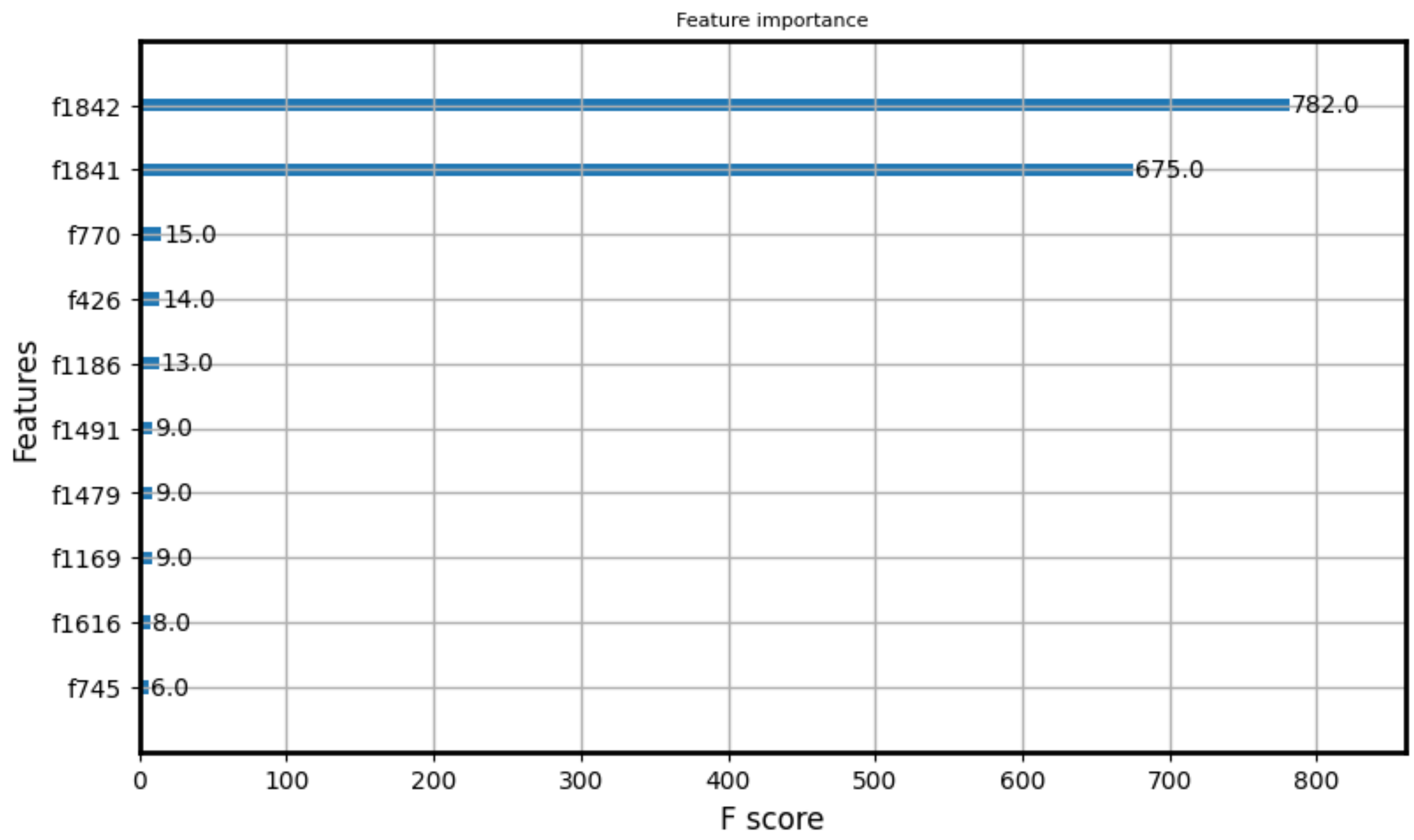
In [64]:
```
#model
xgb_cl = xgb.XGBClassifier()

params = {
    'colsample_bytree': 0.8405981925982379,
    'gamma': 0.028151637840918675,
    'learning_rate': 0.06564537488042158,
    'max_depth': 2,
    'n_estimators': 140,
    'subsample': 0.8976846627773192
}
xgb_cl.fit(X1_train, y1_train)
# Predict
preds = xgb_cl.predict(X1_test)

# Score
accuracy_score(y1_test, preds)
```

Out[64]:
```
0.7890222984562607
```

In [65]:
```
#the top ten
plot_importance(xgb_cl, max_num_features=10) # top 10 product names
plt.show()
```



Kbeast to suggest the top products

In [66]:
```
#packages needed to make suggestion

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.linear_model import Perceptron
```

In [67]:
```
#subseting the product name
g2 = pf['Product Name']
```

In [68]:
```
#transforming using tfidVectorizer
vectorizer = TfidfVectorizer()
spmat = vectorizer.fit_transform(g2)
```

In [69]:
```
#applying the vectorizer
feat_names = vectorizer.get_feature_names()
X3 = pd.DataFrame.sparse.from_spmatrix(spmat, columns=feat_names)
Xtrain, Xtest, ytrain, ytest = train_test_split(X3, yq, test_size=0.5, random_state=1)
```

```
/Users/vannesasalazar/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning:
Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Ple
ase use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

In [70]:
```
#model
kbest = SelectKBest(chi2, k= 10)
X_new = kbest.fit_transform(Xtrain, ytrain)
#Visualize the top ten product
print('Top 10 features %s' % Xtrain.columns[kbest.get_support()].tolist())
```

```
Top 10 features ['06', '1883', '485', 'connectors', 'f9h710', 'flexible', 'ruler', 'southworth', 'supervisor', '
trimflex']
```