```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
path= '/content/drive/MyDrive/Retail stores in US.csv'
```

```
import pandas as pd
df=pd.read_csv(path)
print(df)
```

```
           Ship Mode      Segment         State         Category    Sales  \
0        First Class     Consumer    California        Furniture  290.666
1        First Class    Corporate  Pennsylvania  Office Supplies   16.520
2        First Class    Corporate         Texas  Office Supplies    6.924
3        First Class     Consumer  Pennsylvania        Furniture  170.786
4        First Class    Corporate      New York  Office Supplies   18.900
...              ...          ...           ...              ...      ...
9989  Standard Class  Home Office       Florida       Technology  177.480
9990  Standard Class     Consumer  Pennsylvania       Technology  118.782
9991  Standard Class     Consumer  Pennsylvania  Office Supplies  769.184
9992  Standard Class    Corporate         Texas  Office Supplies   12.992
9993  Standard Class    Corporate         Texas  Office Supplies  149.352

      Quantity  Shipping Cost Order Priority
0            2          54.64           High
1            5           0.42           High
2            6           1.10         Medium
3            1          37.55           High
4            3           0.30           High
...        ...            ...            ...
9989         3          13.93         Medium
9990         3           6.08         Medium
9991         4          50.87         Medium
9992         1           1.60            Low
9993         3           9.11            Low

[9994 rows x 8 columns]
```

```
print(df.info(),'\n')
print(df.head(),'\n')
print(df.describe(),'\n')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Ship Mode       9994 non-null   object
 1   Segment         9994 non-null   object
 2   State           9994 non-null   object
```

```
 3   Category        9994 non-null   object
 4   Sales           9994 non-null   float64
 5   Quantity        9994 non-null   int64
 6   Shipping Cost   9994 non-null   float64
 7   Order Priority  9994 non-null   object
dtypes: float64(2), int64(1), object(5)
memory usage: 624.8+ KB
None
```

	Ship Mode	Segment	State	Category	Sales	Quantity
0	First Class	Consumer	California	Furniture	290.666	2
1	First Class	Corporate	Pennsylvania	Office Supplies	16.520	5
2	First Class	Corporate	Texas	Office Supplies	6.924	6
3	First Class	Consumer	Pennsylvania	Furniture	170.786	1
4	First Class	Corporate	New York	Office Supplies	18.900	3

	Shipping Cost	Order Priority
0	54.64	High
1	0.42	High
2	1.10	Medium
3	37.55	High
4	0.30	High

	Sales	Quantity	Shipping Cost
count	9994.000000	9994.000000	9994.000000
mean	229.858001	3.789574	23.831678
std	623.245101	2.225110	58.962848
min	0.444000	1.000000	0.010000
25%	17.280000	2.000000	1.490000
50%	54.490000	3.000000	5.100000
75%	209.940000	5.000000	19.985000
max	22638.480000	14.000000	933.570000

```
df.skew()
```

```
<ipython-input-69-9e0b1e29546f>:1: FutureWarning: The default value of
numeric_only in DataFrame.skew is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence
this warning.
  df.skew()
```

```
Sales            12.972752
Quantity          1.278545
Shipping Cost     6.294245
dtype: float64
```

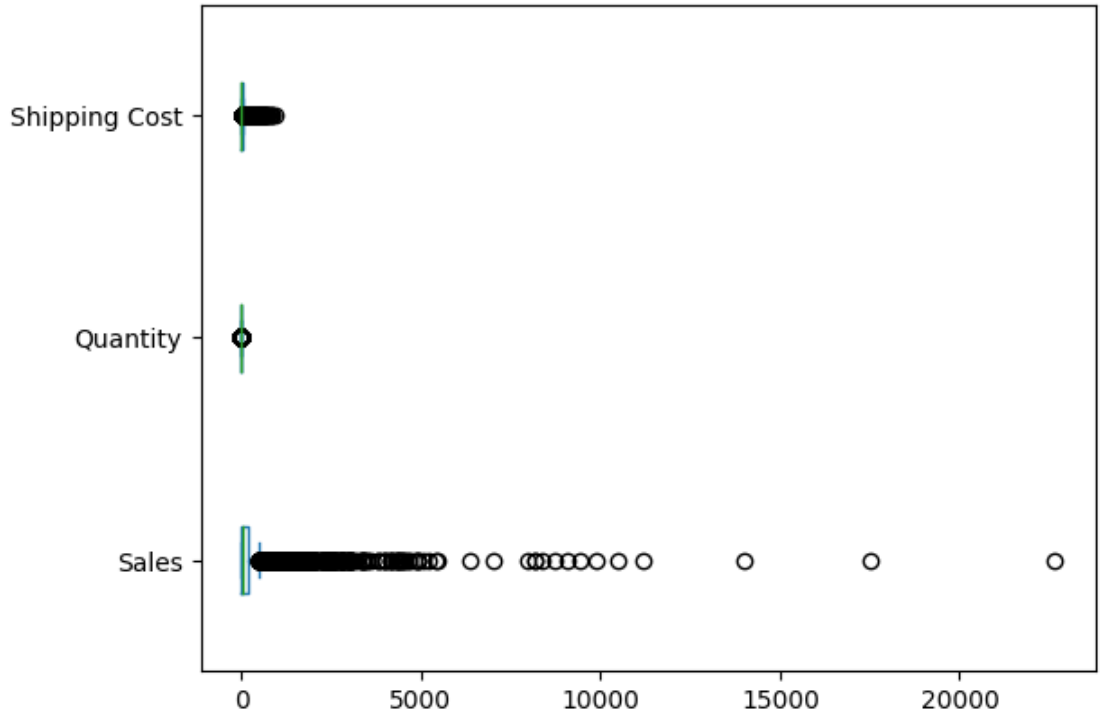
```
df.kurtosis()
```

```
<ipython-input-70-c7edf97eb14c>:1: FutureWarning: The default value of
numeric_only in DataFrame.kurt is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence
this warning.
  df.kurtosis()
```

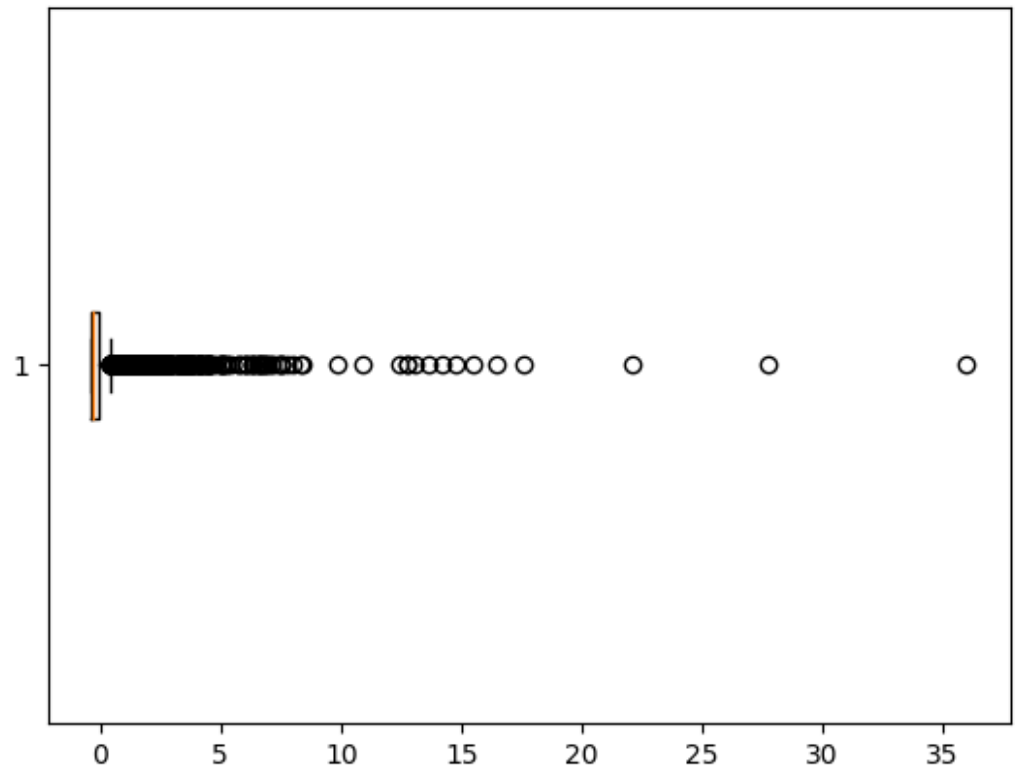
```
Sales            305.311753
Quantity           1.991889
Shipping Cost     55.374474
dtype: float64
```

```
# plotbox before removing outliers
df.plot.box(vert=False)
```

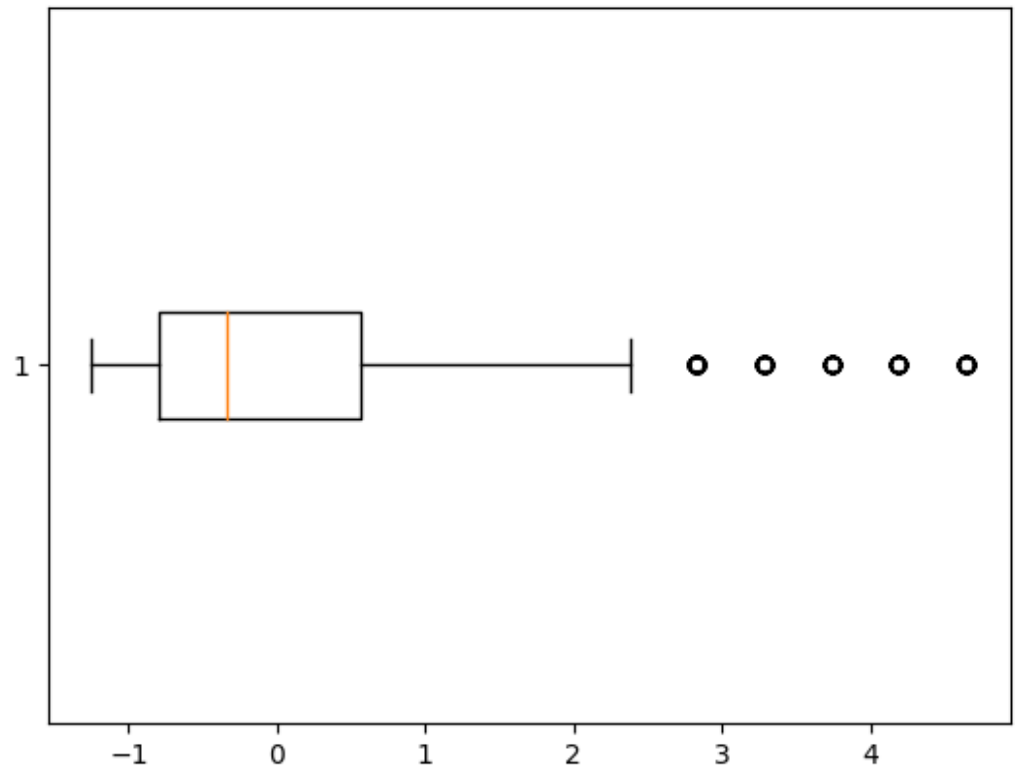
```
<Axes: >
```



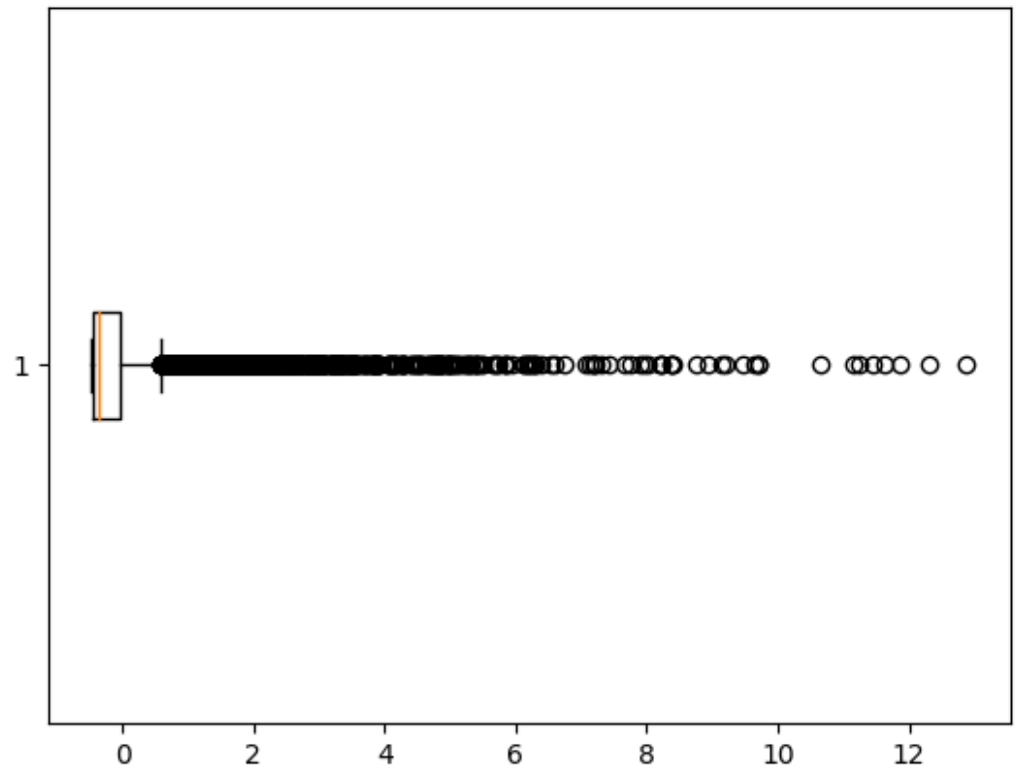

```
from scipy import stats
import matplotlib.pyplot as plt

# Calculate Z-score for Sales column
z_scores = stats.zscore(df['Sales'])
plt.boxplot(z_scores,vert=False)

# Reatin rows with Z-score greater than -3 or less than +3
df = df[(z_scores >= -3) & (z_scores <= 3)]
```




```
# Calculate Z-score for Quantity column
z_scores = stats.zscore(df['Quantity'])
plt.boxplot(z_scores,vert=False)

# Reatin rows with Z-score greater than -3 or less than +3
df = df[(z_scores >= -3) & (z_scores <= 3)]
```

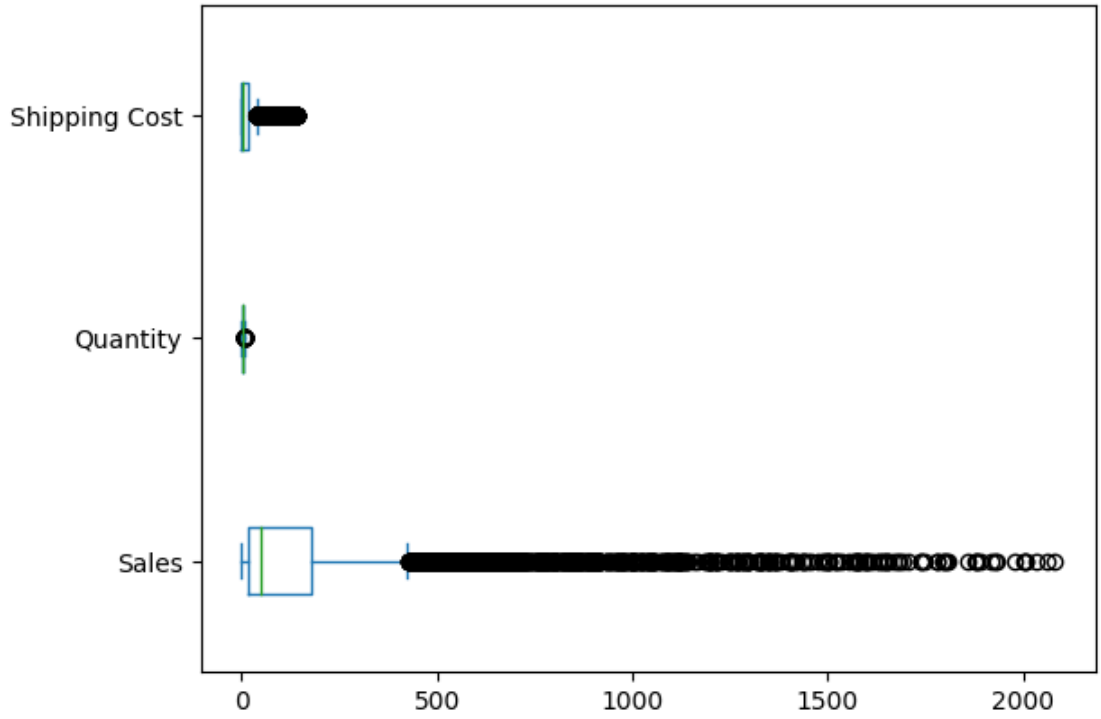



```
# Calculate Z-score for Shipping Cost column
z_scores = stats.zscore(df['Shipping Cost'])
plt.boxplot(z_scores,vert=False)

# Reatin rows with Z-score greater than -3 or less than +3
df = df[(z_scores >= -3) & (z_scores <= 3)]
```



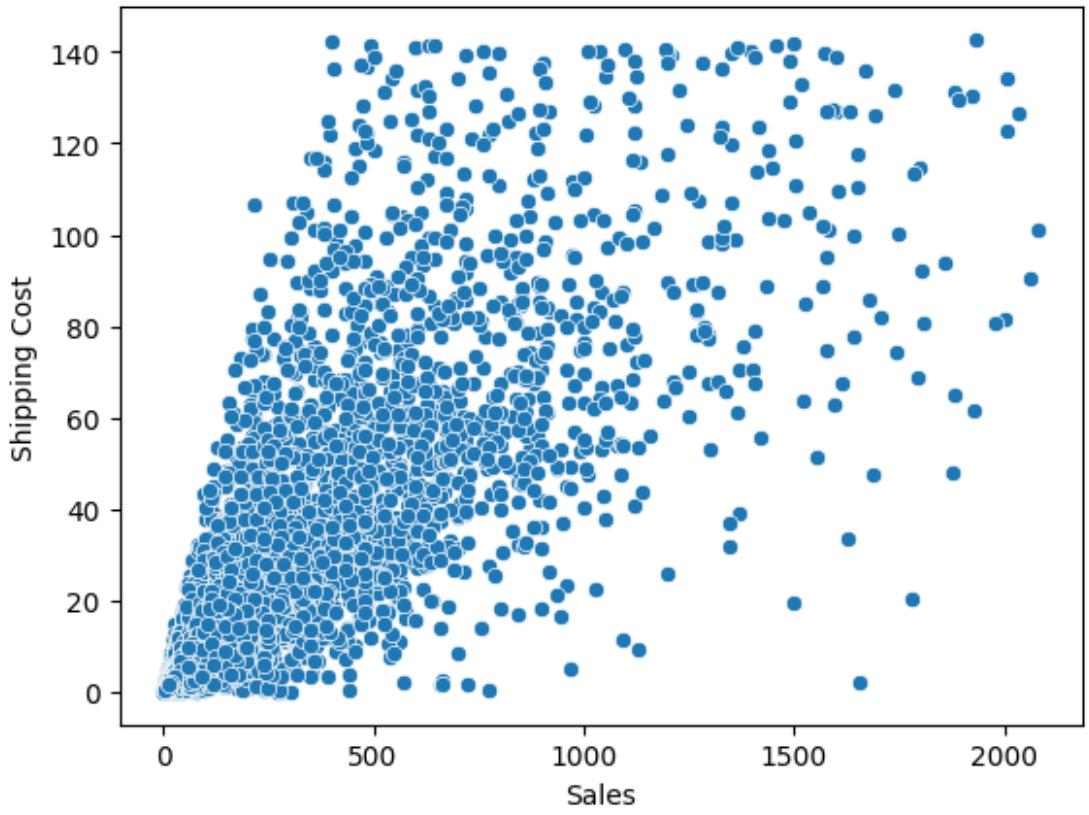

```
# plotbox after removing outliers
df.plot.box(vert=False)
```

```
<Axes: >
```



```
# compute the correlation between sales and shipping cost
import seaborn as sns
from scipy.stats import pearsonr

sns.scatterplot(x='Sales', y='Shipping Cost', data=df)
```

```
<Axes: xlabel='Sales', ylabel='Shipping Cost'>
```

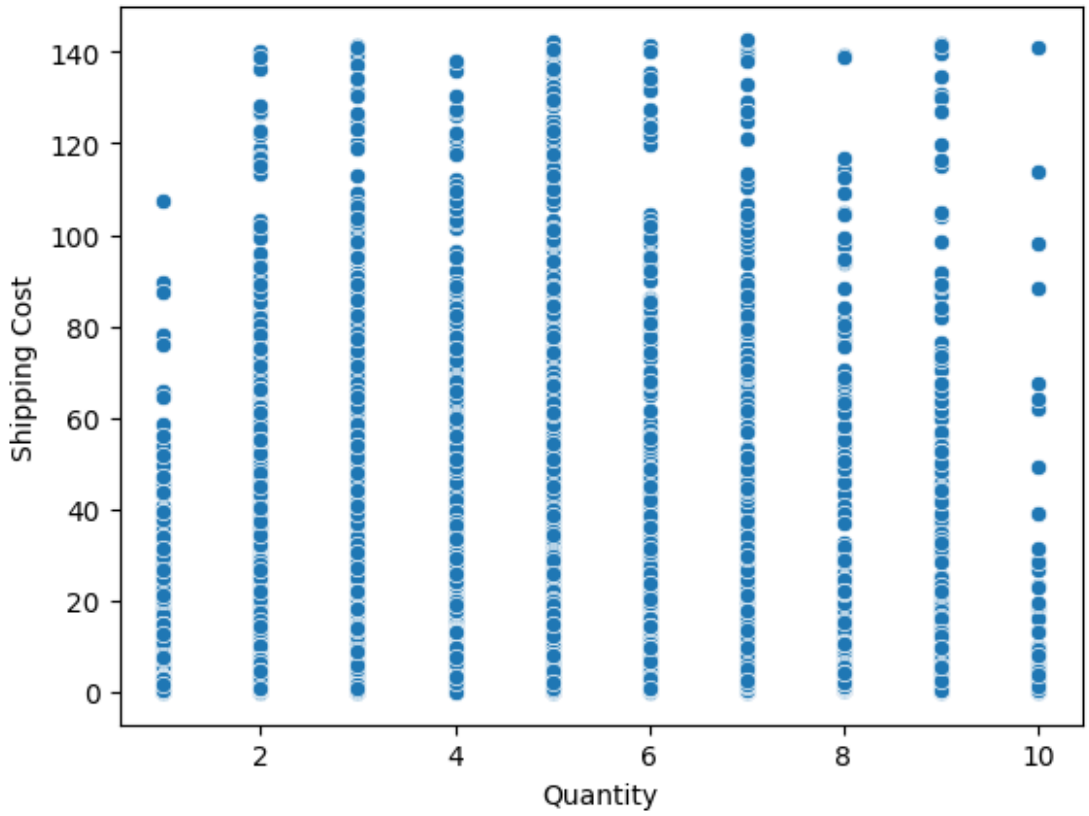



```
pearson_corr, pearson_p = pearsonr(df['Sales'], df['Shipping Cost'])
print('Pearson correlation coefficient:', pearson_corr)
print('Pearson correlation p-value:', pearson_p)
```

```
Pearson correlation coefficient: 0.8269130779793633
Pearson correlation p-value: 0.0
```

```
# compute the correlation between quantity and shipping cost
sns.scatterplot(x='Quantity', y='Shipping Cost', data=df)
```

```
<Axes: xlabel='Quantity', ylabel='Shipping Cost'>
```




```
pearson_corr, pearson_p = pearsonr(df['Quantity'], df['Shipping Cost'])
print('Pearson correlation coefficient:', pearson_corr)
print('Pearson correlation p-value:', pearson_p)
```

```
Pearson correlation coefficient: 0.2027351107508617
Pearson correlation p-value: 4.663816213712061e-89
```

```
# Descriptive statistics of the continous variable
df.describe()
```

	Sales	Quantity	Shipping Cost
count	9541.000000	9541.000000	9541.000000
mean	155.653721	3.629913	14.925716
std	256.224502	1.999534	24.089688
min	0.444000	1.000000	0.010000
25%	16.176000	2.000000	1.400000
50%	48.360000	3.000000	4.590000
75%	179.940000	5.000000	16.560000
max	2079.400000	10.000000	142.760000

```
# import the packeages for linear regression
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

```
import seaborn as sns
from math import sqrt

df=pd.read_csv(path)

df.head()
```

```
     Ship Mode    Segment         State         Category    Sales  Quantity
\
0  First Class   Consumer    California        Furniture  290.666         2
1  First Class  Corporate  Pennsylvania  Office Supplies   16.520         5
2  First Class  Corporate         Texas  Office Supplies    6.924         6
3  First Class   Consumer  Pennsylvania        Furniture  170.786         1
4  First Class  Corporate      New York  Office Supplies   18.900         3

   Shipping Cost Order Priority
0          54.64           High
1           0.42           High
2           1.10         Medium
3          37.55           High
4           0.30           High
```

```
# drop state categorical variable
df.drop(['State',], axis=1, inplace=True)

# the data frame after droping the state column
df
```

```
           Ship Mode      Segment         Category    Sales  Quantity  \
0        First Class     Consumer        Furniture  290.666         2
1        First Class    Corporate  Office Supplies   16.520         5
2        First Class    Corporate  Office Supplies    6.924         6
3        First Class     Consumer        Furniture  170.786         1
4        First Class    Corporate  Office Supplies   18.900         3
...              ...          ...              ...      ...       ...
9989  Standard Class  Home Office       Technology  177.480         3
9990  Standard Class     Consumer       Technology  118.782         3
9991  Standard Class     Consumer  Office Supplies  769.184         4
9992  Standard Class    Corporate  Office Supplies   12.992         1
9993  Standard Class    Corporate  Office Supplies  149.352         3

      Shipping Cost Order Priority
0             54.64           High
1              0.42           High
2              1.10         Medium
3             37.55           High
4              0.30           High
...             ...            ...
9989          13.93         Medium
9990           6.08         Medium
9991          50.87         Medium
```

```
9992           1.60            Low
9993           9.11            Low

[9994 rows x 7 columns]
```

```
# Get dummies for the categorical variables
df = pd.get_dummies(df, columns=['Ship Mode', 'Segment', 'Order Priority',
'Category'])

# Define the features and target variable
X = df.drop(['Shipping Cost'], axis=1)
y = df['Shipping Cost']

# split data into test and train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# fit the data
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

print("Intercept: \n",regressor.intercept_)
print("Coefficients: \n",regressor.coef_)
```

```
Intercept:
 13.862098596154663
Coefficients:
 [  0.05793245   2.79283351   2.30904053   8.35377678  -3.19276748
  -7.47004983  -0.21360316   0.64946006  -0.43585691  15.36811198
  -0.30917896  -2.69159251 -12.36734052   3.91710991  -8.75483178
   4.83772186]
```

```
y_pred
```

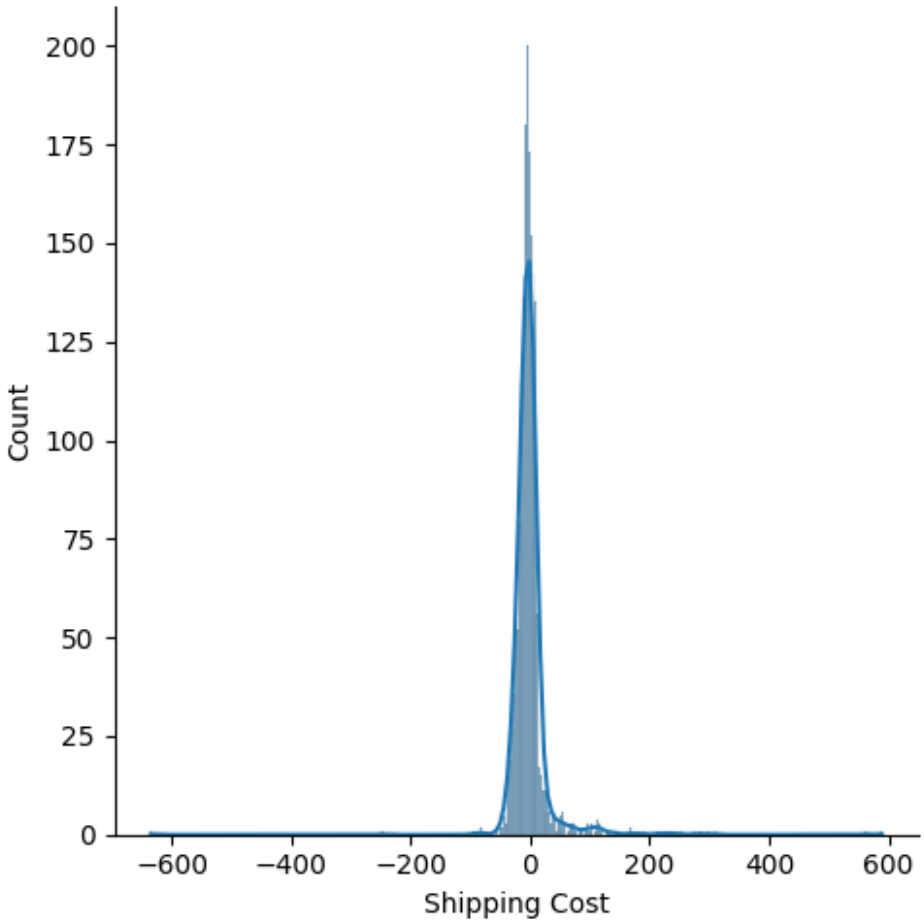
```
array([ 4.99501368, 27.76344866,  6.50397487, ..., 32.89130683,
       25.40895241, 10.20095251])
```

```
# Obtian the perfomance metrics
print('R-squared: {:.3f}'.format(r2_score(y_test, y_pred)))
print('MSE: {:.3f}'.format(mean_squared_error(y_test, y_pred)))
print('RMSE: {:.3f}'.format(sqrt(mean_squared_error(y_test, y_pred))))
print('MAE: {:.3f}'.format(mean_absolute_error(y_test, y_pred)))
```

```
R-squared: 0.559
MSE: 1401.167
RMSE: 37.432
MAE: 15.669
```

```
# Show the error distribution
error = y_test - y_pred
sns.displot(error, kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f6f3d0116f0>
```



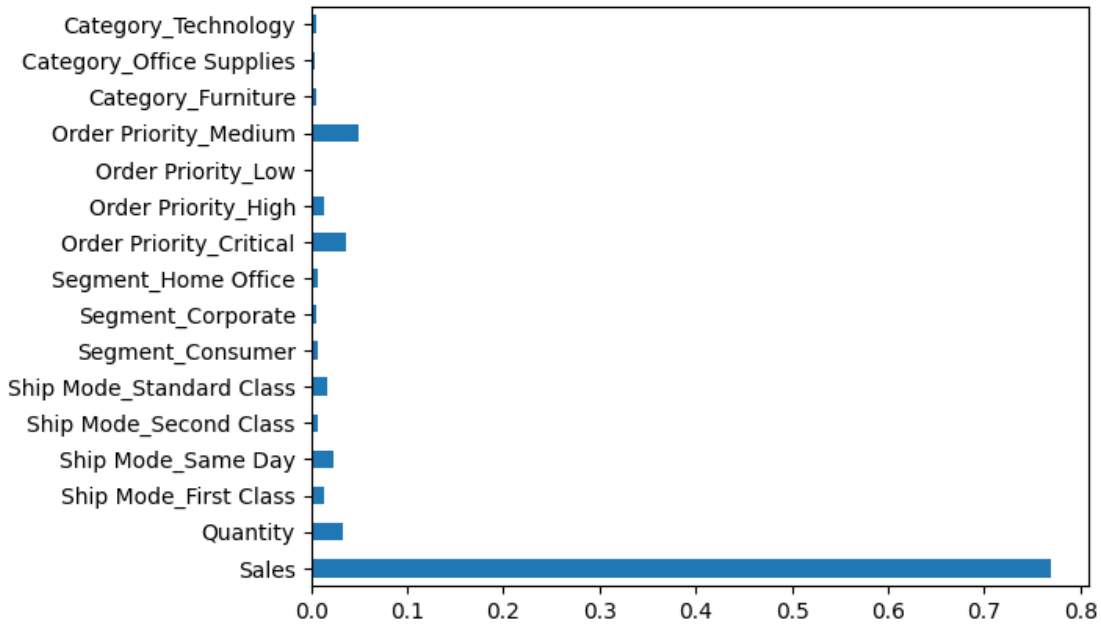

```
# import the packages for random forest regression
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Define the features and target variable
x= df.drop(['Shipping Cost'], axis=1)
y= df['Shipping Cost']

# split the data into test and train data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)

RandomForestRegressor(random_state=42)
```

```
# fit the model and obtain the performance metrics
y_pred = rf_model.predict(X_test)
print('Test R2 score:', r2_score(y_test, y_pred))
print('Test MSE score:', mean_squared_error(y_test, y_pred))
```

```
Test R2 score: 0.7574354174360364
Test MSE score: 770.2949575654815
```

```
# show the importanct features
importances = pd.Series(rf_model.feature_importances_, index=x.columns)
print(importances)
importances.plot(kind='barh')
plt.show()
```

```
Sales                          0.769286
Quantity                       0.032473
Ship Mode_First Class          0.014448
Ship Mode_Same Day             0.023455
Ship Mode_Second Class         0.006949
Ship Mode_Standard Class       0.017006
Segment_Consumer               0.006939
Segment_Corporate              0.005140
Segment_Home Office            0.007802
Order Priority_Critical        0.036812
Order Priority_High            0.013942
Order Priority_Low             0.000456
Order Priority_Medium          0.050031
Category_Furniture             0.005031
Category_Office Supplies       0.004532
Category_Technology            0.005699
dtype: float64
```

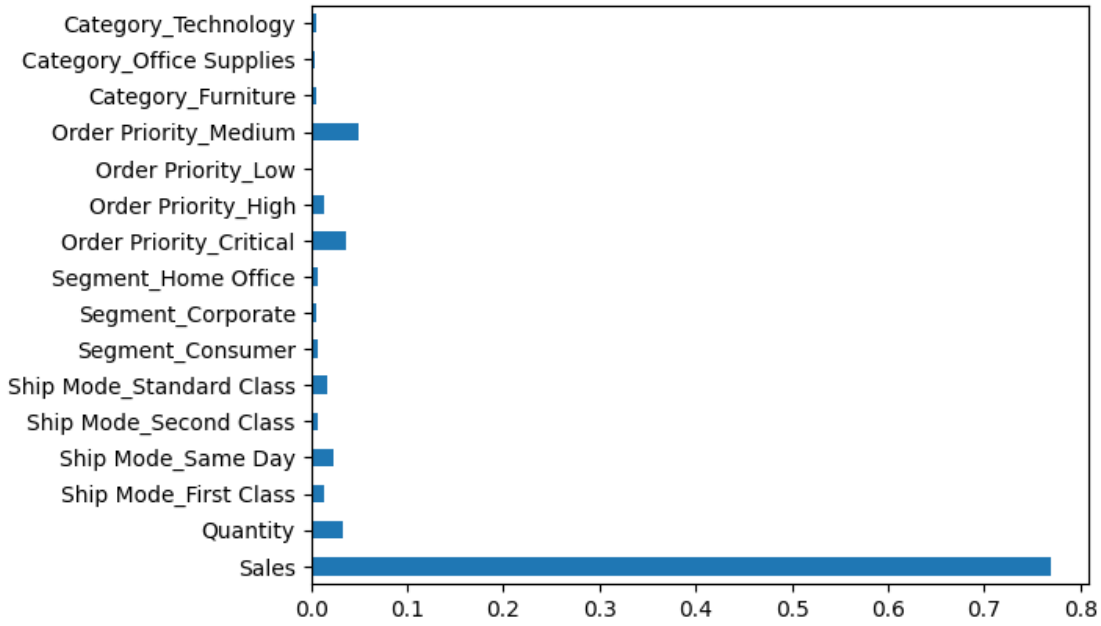


```
# import the packages for Out-of-bag regression model and obtain the
performace metrics
rf_modelWithoob = RandomForestRegressor(random_state=42, oob_score=True)
rf_modelWithoob.fit(X_train, y_train)
print('OOB R2 score:', rf_modelWithoob.oob_score_)
print('OOB MSE score:', mean_squared_error(y_train,
rf_modelWithoob.oob_prediction_))
```

```
OOB R2 score: 0.7866121489128859
OOB MSE score: 757.7892977955773
```

```
# show the importanct features
importances = pd.Series(rf_modelWithoob.feature_importances_,
index=x.columns)
print(importances)
importances.plot(kind='barh')
plt.show()
```

```
Sales                       0.769286
Quantity                    0.032473
Ship Mode_First Class       0.014448
Ship Mode_Same Day          0.023455
Ship Mode_Second Class      0.006949
Ship Mode_Standard Class    0.017006
Segment_Consumer            0.006939
Segment_Corporate           0.005140
Segment_Home Office         0.007802
Order Priority_Critical     0.036812
Order Priority_High         0.013942
Order Priority_Low          0.000456
Order Priority_Medium       0.050031
Category_Furniture          0.005031
Category_Office Supplies    0.004532
Category_Technology         0.005699
dtype: float64
```




```
# performance the hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300, 400]
    }

scoring = {'R2': 'r2', 'MSE': 'neg_mean_squared_error'}
print(scoring)
```

```
{'R2': 'r2', 'MSE': 'neg_mean_squared_error'}
```

```
# perform the hyperparameter tuning using grid search
grid_search = GridSearchCV(rf_model, param_grid=param_grid, cv=5,
scoring=scoring, refit='R2')
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             param_grid={'n_estimators': [100, 200, 300, 400]}, refit='R2',
             scoring={'MSE': 'neg_mean_squared_error', 'R2': 'r2'})
```

```
y_pred = grid_search.predict(X_test)
print('Test R2 score:', r2_score(y_test, y_pred))
print('Test MSE score:', mean_squared_error(y_test, y_pred))
```

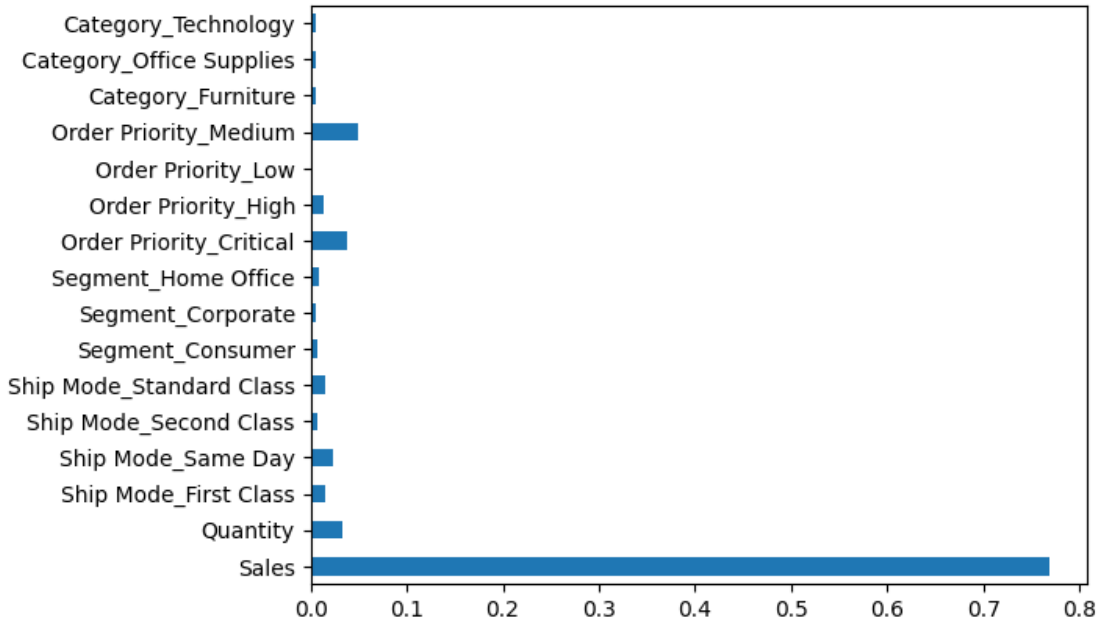
```
Test R2 score: 0.7572889050845752
Test MSE score: 770.7602263378581
```

```
print('Best hyperparameters:', grid_search.best_params_)
print('Best R2 score:', grid_search.best_score_)
print('Best MSE score:',
abs(grid_search.cv_results_['mean_test_MSE'][grid_search.best_index_]))
```

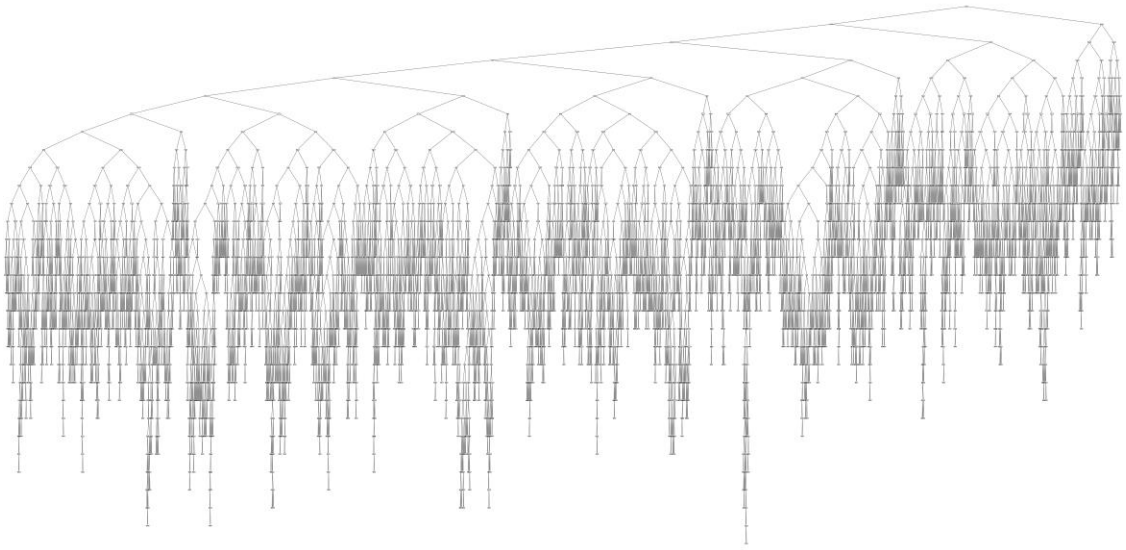
```
Best hyperparameters: {'n_estimators': 300}
Best R2 score: 0.7867475709746885
Best MSE score: 755.7615298497743
```

```
# obtain the importance features after hyperparameter tuning
importances = pd.Series(grid_search.best_estimator_.feature_importances_,
index=x.columns)
print(importances)
importances.plot(kind='barh')
plt.show()
```

```
Sales                         0.768497
Quantity                      0.032380
Ship Mode_First Class         0.015287
Ship Mode_Same Day            0.022995
Ship Mode_Second Class        0.007089
Ship Mode_Standard Class      0.016092
Segment_Consumer              0.006810
Segment_Corporate             0.005255
Segment_Home Office           0.007993
Order Priority_Critical       0.038142
Order Priority_High           0.013688
Order Priority_Low            0.000659
Order Priority_Medium         0.049780
Category_Furniture            0.005250
Category_Office Supplies      0.004799
Category_Technology           0.005285
dtype: float64
```



```
# show the number of trees
from sklearn.tree import plot_tree
fig, ax = plt.subplots(figsize=(100, 50))
plot_tree(grid_search.best_estimator_[0], ax=ax,
feature_names=X_train.columns)
plt.show()
```



```
# perform the hyperparameter tuning using random search
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

random_search = RandomizedSearchCV(rf_model, param_distributions=param_grid,
n_iter=20, cv=5, scoring=scoring, refit='R2')

random_search.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/model_selection/_search.py:305: UserWarning: The total space
of parameters 4 is smaller than n_iter=20. Running 4 iterations. For
exhaustive searches, use GridSearchCV.
  warnings.warn(

RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
                   n_iter=20,
                   param_distributions={'n_estimators': [100, 200, 300,
400]},
                   refit='R2',
                   scoring={'MSE': 'neg_mean_squared_error', 'R2': 'r2'})
```

```
# Obtain the best number of trees and performance metrics
best_model = random_search.best_estimator_
best_model.fit(X_train, y_train)

RandomForestRegressor(n_estimators=300, random_state=42)
```

```
y_pred = best_model.predict(X_test)
print("Test R2 score:", r2_score(y_test, y_pred))
print("Test MSE score:", mean_squared_error(y_test, y_pred))
print('\n')
```

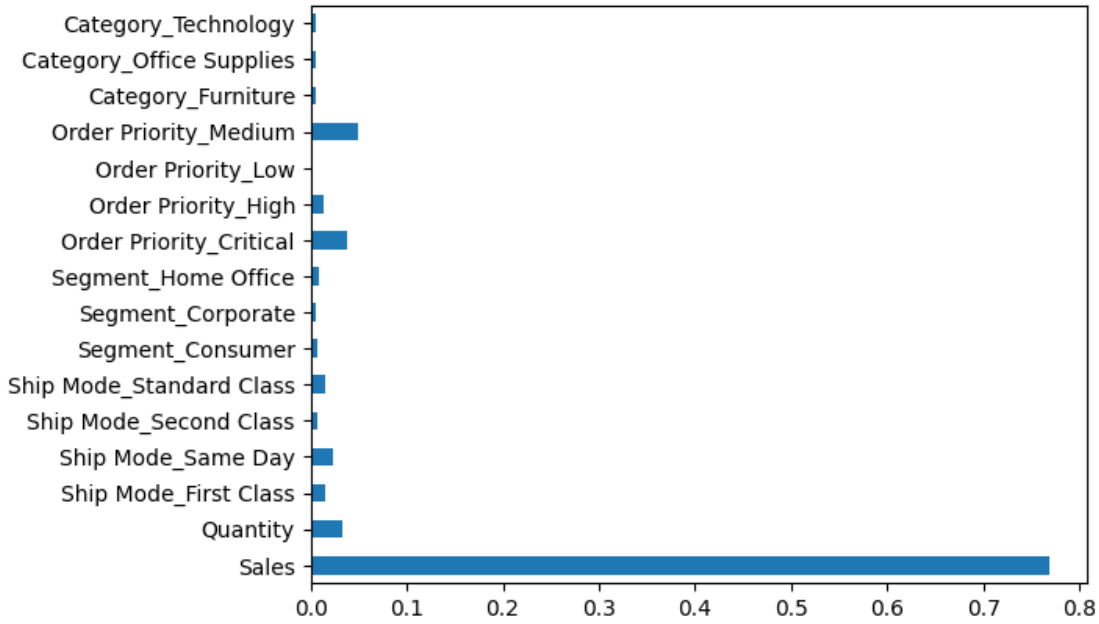
```
Test R2 score: 0.7572889050845752
Test MSE score: 770.7602263378581
```

```
print('Best hyperparameters:', random_search.best_params_)
print('Best R2 score:', random_search.best_score_)
print('Best MSE score:',
abs(random_search.cv_results_['mean_test_MSE'][random_search.best_index_]))
print('\n')
```

```
Best hyperparameters: {'n_estimators': 300}
Best R2 score: 0.7867475709746885
Best MSE score: 755.7615298497743
```

```
# obtain the importance metric using random search
importances = pd.Series(random_search.best_estimator_.feature_importances_,
index=x.columns)
print('Feature importances:')
print(importances)
importances.plot(kind='barh')
plt.show()
```

```
Feature importances:
Sales                        0.768497
Quantity                     0.032380
Ship Mode_First Class        0.015287
Ship Mode_Same Day           0.022995
Ship Mode_Second Class       0.007089
Ship Mode_Standard Class     0.016092
Segment_Consumer             0.006810
Segment_Corporate            0.005255
Segment_Home Office          0.007993
Order Priority_Critical      0.038142
Order Priority_High          0.013688
Order Priority_Low           0.000659
Order Priority_Medium        0.049780
Category_Furniture           0.005250
Category_Office Supplies     0.004799
Category_Technology          0.005285
dtype: float64
```




```
from sklearn.tree import plot_tree
fig, ax = plt.subplots(figsize=(50, 40))
plot_tree(random_search.best_estimator_[0], ax=ax,
feature_names=X_train.columns)
plt.show()
```