In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
pd.set_option('display.max_columns', None)
```

In [2]:
```
df = pd.read_csv('supply_chain_data.csv')
df.head()
```

Out[2]:

	Product type	SKU	Price	Availability	Number of products sold	Revenue generated	Customer demographics	Stock levels	Lead times	Order quantities	Shipping times	Shi ca
0	haircare	SKU0	69.808006	55	802	8661.996792	Non-binary	58	7	96	4	Ca
1	skincare	SKU1	14.843523	95	736	7460.900065	Female	53	30	37	2	Ca
2	haircare	SKU2	11.319683	34	8	9577.749626	Unknown	1	10	88	2	Ca
3	skincare	SKU3	61.163343	68	83	7766.836426	Non-binary	23	13	59	6	Ca
4	skincare	SKU4	4.805496	26	871	2686.505152	Non-binary	5	3	56	8	Ca

Data Preprocessing Part 1

In [3]:
```
df.dtypes
```

Out[3]:
```
Product type                object
SKU                         object
Price                      float64
Availability                 int64
Number of products sold      int64
Revenue generated          float64
Customer demographics       object
Stock levels                 int64
Lead times                   int64
Order quantities             int64
Shipping times               int64
Shipping carriers           object
Shipping costs             float64
Supplier name               object
Location                    object
Lead time                    int64
Production volumes           int64
Manufacturing lead time      int64
Manufacturing costs        float64
Inspection results          object
Defect rates               float64
Transportation modes        object
Routes                      object
Costs                      float64
dtype: object
```

In [4]:
```
df.select_dtypes(include='object').nunique()
```

Out[4]:
```
Product type               3
SKU                      100
Customer demographics      4
Shipping carriers          3
Supplier name              5
Location                   5
Inspection results         3
Transportation modes       4
Routes                     3
dtype: int64
```

In [5]:
```
df.shape
```

Out[5]:
```
(100, 24)
```

In [6]:
```
#Drop SKU Column because this is just supply chain id
df.drop(columns=['SKU'], inplace=True)
df.shape
```

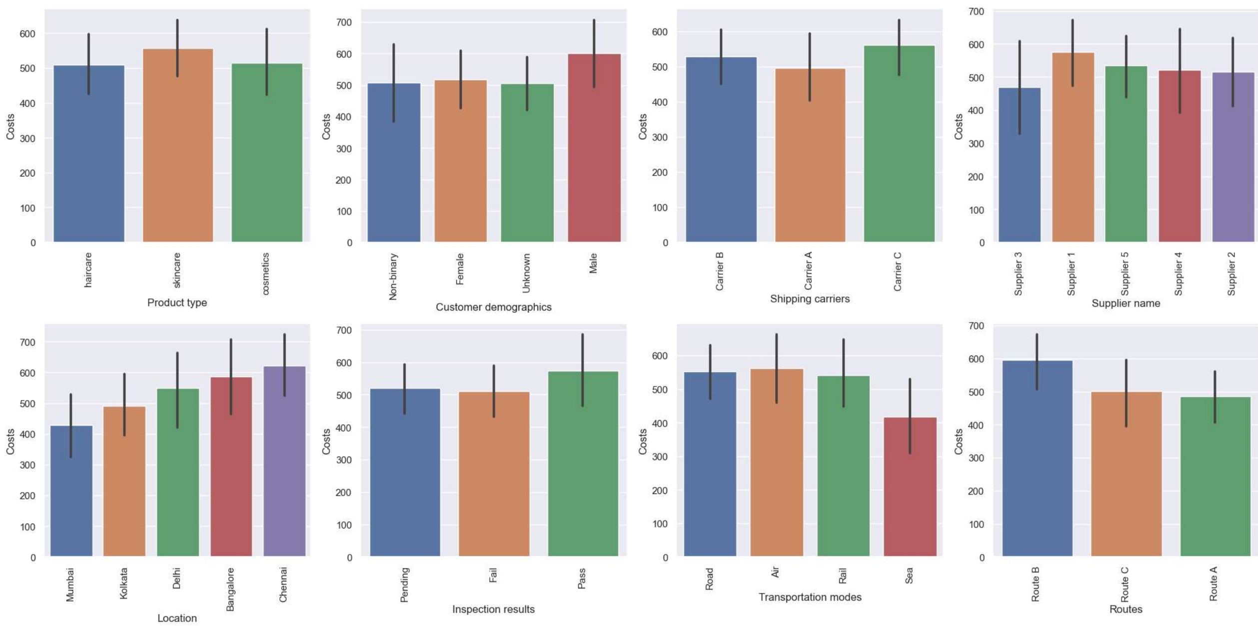
Out[6]:
```
(100, 23)
```

In [7]:
```
df.nunique()
```

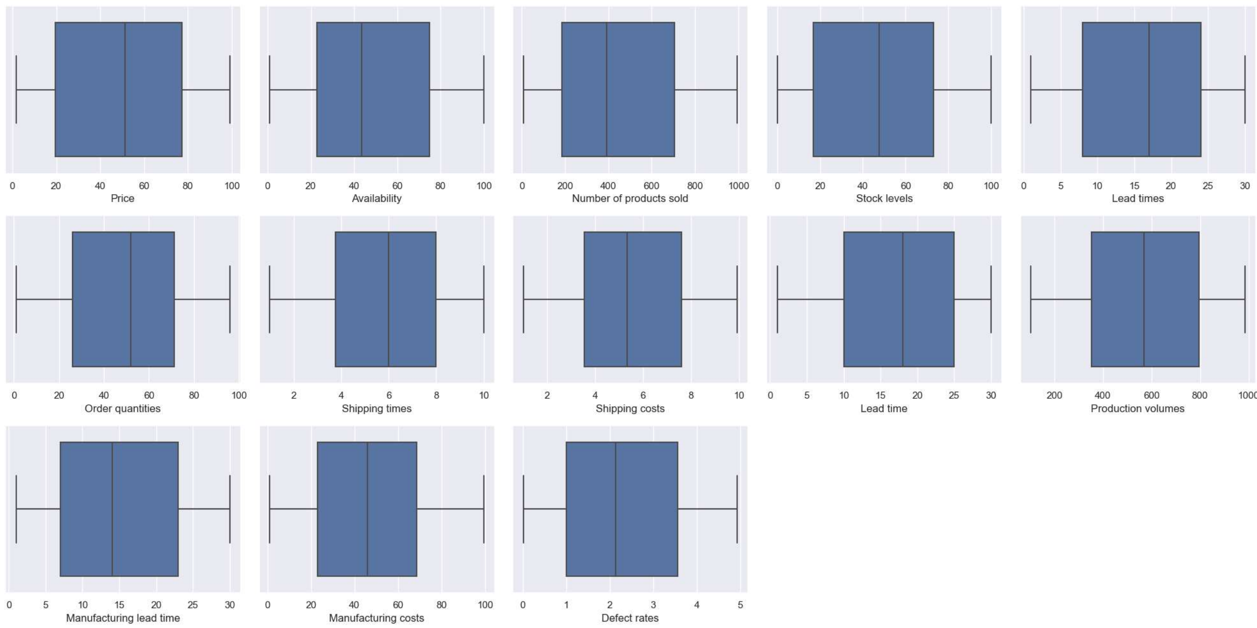
Out[7]:
```
Product type                 3
Price                      100
Availability                63
Number of products sold     96
Revenue generated          100
Customer demographics        4
Stock levels                65
Lead times                  29
Order quantities            61
Shipping times              10
Shipping carriers            3
Shipping costs             100
Supplier name                5
Location                     5
Lead time                   29
Production volumes          96
Manufacturing lead time     30
Manufacturing costs        100
Inspection results           3
Defect rates               100
Transportation modes         4
Routes                       3
Costs                      100
dtype: int64
```

Exploratory Data Analysis

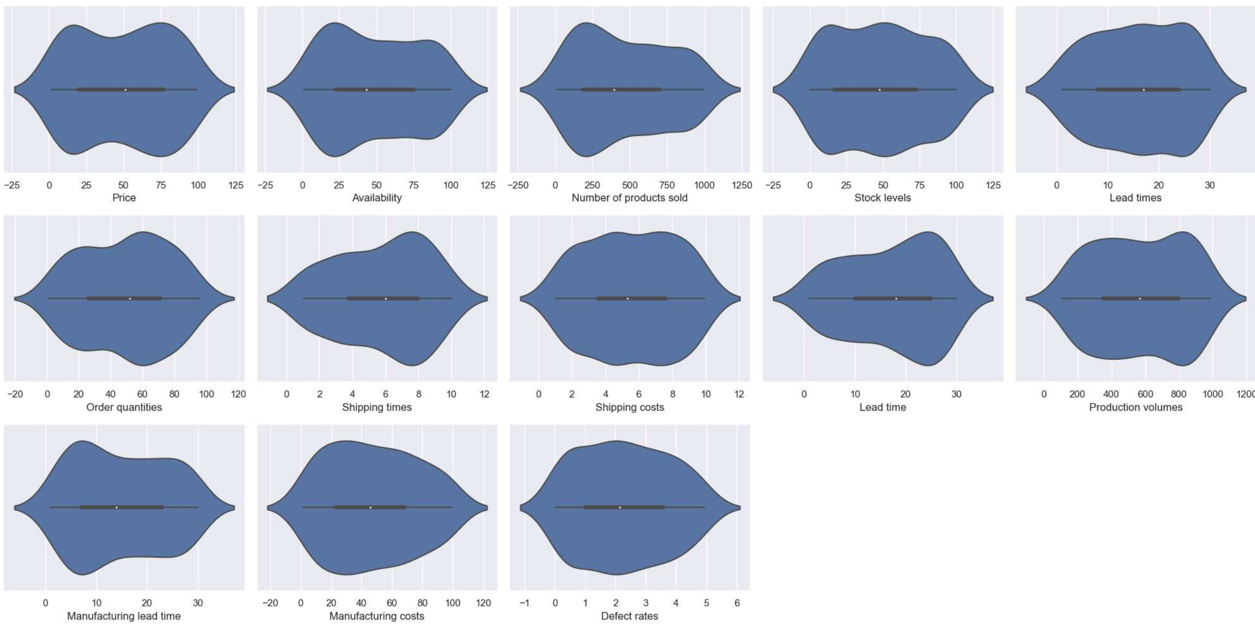
In [8]:
```
# list of categorical variables to plot
cat_vars = ['Product type', 'Customer demographics', 'Shipping carriers', 'Supplier name', 'Locat
            'Inspection results', 'Transportation modes', 'Routes']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.barplot(x=var, y='Costs', data=df, ax=axs[i], estimator=np.mean)
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



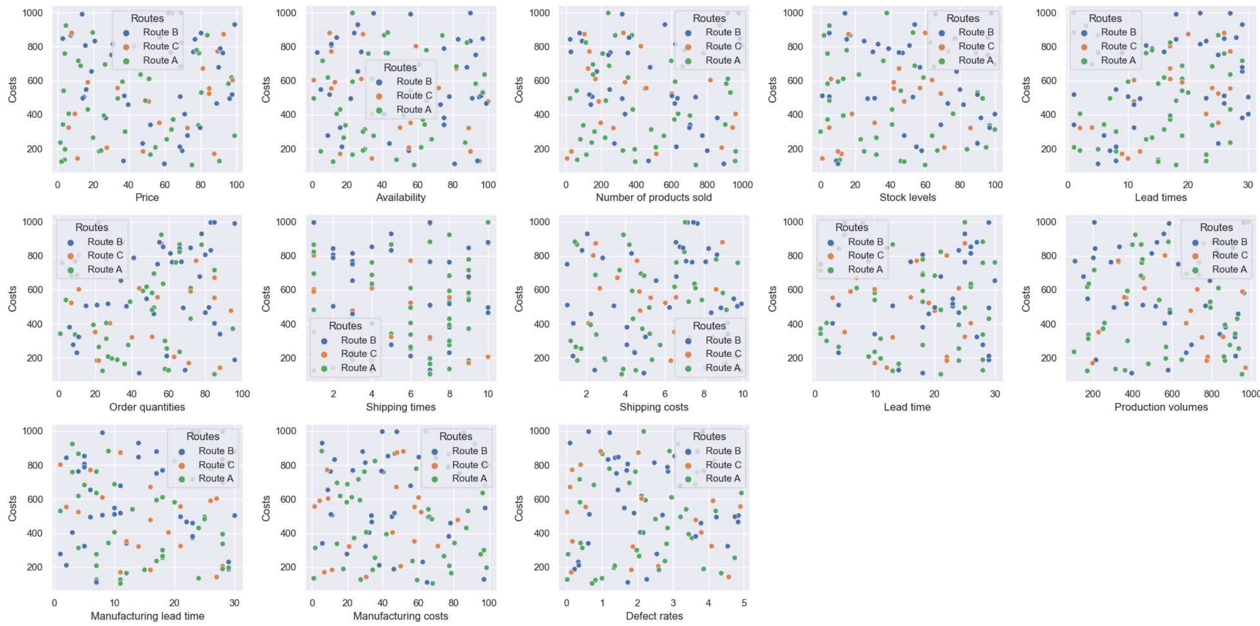
In [9]:
```
num_vars = ['Price', 'Availability', 'Number of products sold', 'Stock levels',
            'Lead times', 'Order quantities', 'Shipping times', 'Shipping costs',
            'Lead time', 'Production volumes', 'Manufacturing lead time', 'Manufacturing costs',
            'Defect rates']

fig, axs = plt.subplots(nrows=3, ncols=5, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

# remove the 14th subplot
fig.delaxes(axs[13])
# remove the 15th subplot
fig.delaxes(axs[14])

fig.tight_layout()
plt.show()
```



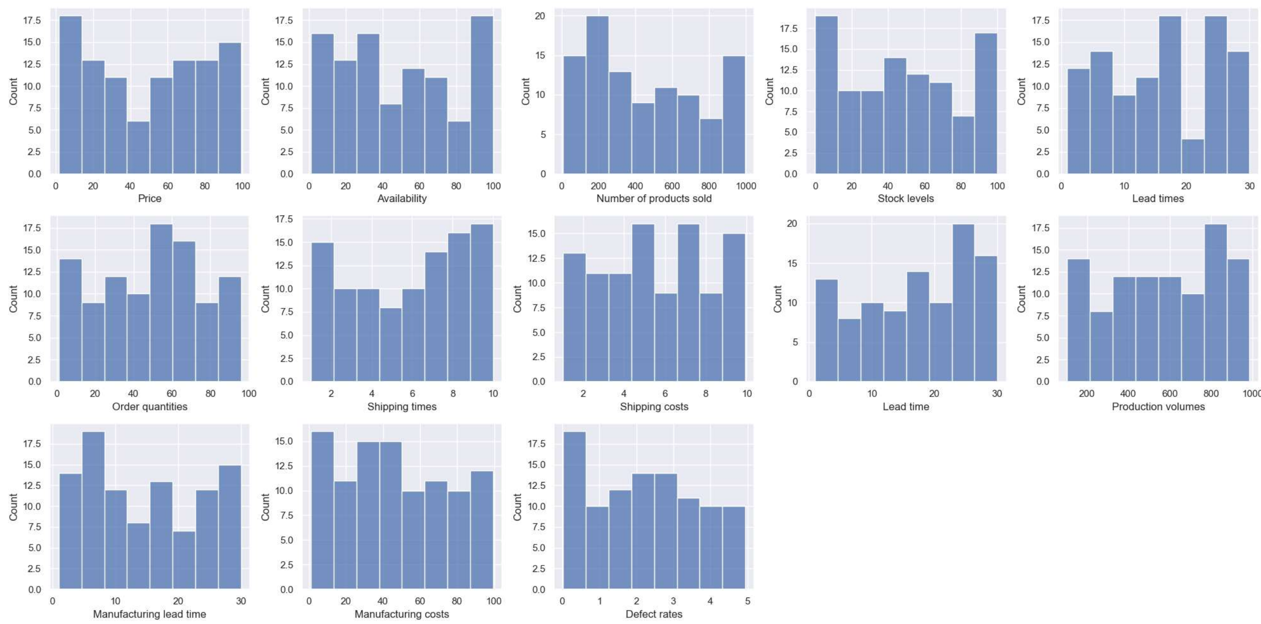
In [10]:
```
num_vars = ['Price', 'Availability', 'Number of products sold', 'Stock levels',
            'Lead times', 'Order quantities', 'Shipping times', 'Shipping costs',
            'Lead time', 'Production volumes', 'Manufacturing lead time', 'Manufacturing costs',
            'Defect rates']

fig, axs = plt.subplots(nrows=3, ncols=5, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

# remove the 14th subplot
fig.delaxes(axs[13])
# remove the 15th subplot
fig.delaxes(axs[14])

fig.tight_layout()
plt.show()
```



In [11]:
```
num_vars = ['Price', 'Availability', 'Number of products sold', 'Stock levels',
            'Lead times', 'Order quantities', 'Shipping times', 'Shipping costs',
            'Lead time', 'Production volumes', 'Manufacturing lead time', 'Manufacturing costs',
            'Defect rates']

fig, axs = plt.subplots(nrows=3, ncols=5, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.scatterplot(x=var, y='Costs', hue='Routes', data=df, ax=axs[i])

# remove the 14th subplot
fig.delaxes(axs[13])
# remove the 15th subplot
fig.delaxes(axs[14])

fig.tight_layout()

plt.show()
```



In [12]:
```
num_vars = ['Price', 'Availability', 'Number of products sold', 'Stock levels',
            'Lead times', 'Order quantities', 'Shipping times', 'Shipping costs',
            'Lead time', 'Production volumes', 'Manufacturing lead time', 'Manufacturing costs',
            'Defect rates']

fig, axs = plt.subplots(nrows=3, ncols=5, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.histplot(x=var, data=df, ax=axs[i])

# remove the 14th subplot
fig.delaxes(axs[13])
# remove the 15th subplot
fig.delaxes(axs[14])

fig.tight_layout()
plt.show()
```



Data Preprocessing Part 2

In [13]:
```
#Check the missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[13]:
```
Series([], dtype: float64)
```

Label Encoding for Object datatypes

In [14]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
Product type: ['haircare' 'skincare' 'cosmetics']
Customer demographics: ['Non-binary' 'Female' 'Unknown' 'Male']
Shipping carriers: ['Carrier B' 'Carrier A' 'Carrier C']
Supplier name: ['Supplier 3' 'Supplier 1' 'Supplier 5' 'Supplier 4' 'Supplier 2']
Location: ['Mumbai' 'Kolkata' 'Delhi' 'Bangalore' 'Chennai']
Inspection results: ['Pending' 'Fail' 'Pass']
Transportation modes: ['Road' 'Air' 'Rail' 'Sea']
Routes: ['Route B' 'Route C' 'Route A']
```

In [15]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Product type: [1 2 0]
Customer demographics: [2 0 3 1]
Shipping carriers: [1 0 2]
Supplier name: [2 0 4 3 1]
Location: [4 3 2 0 1]
Inspection results: [2 0 1]
Transportation modes: [2 0 1 3]
Routes: [1 2 0]
```

In [16]:
```
df.dtypes
```

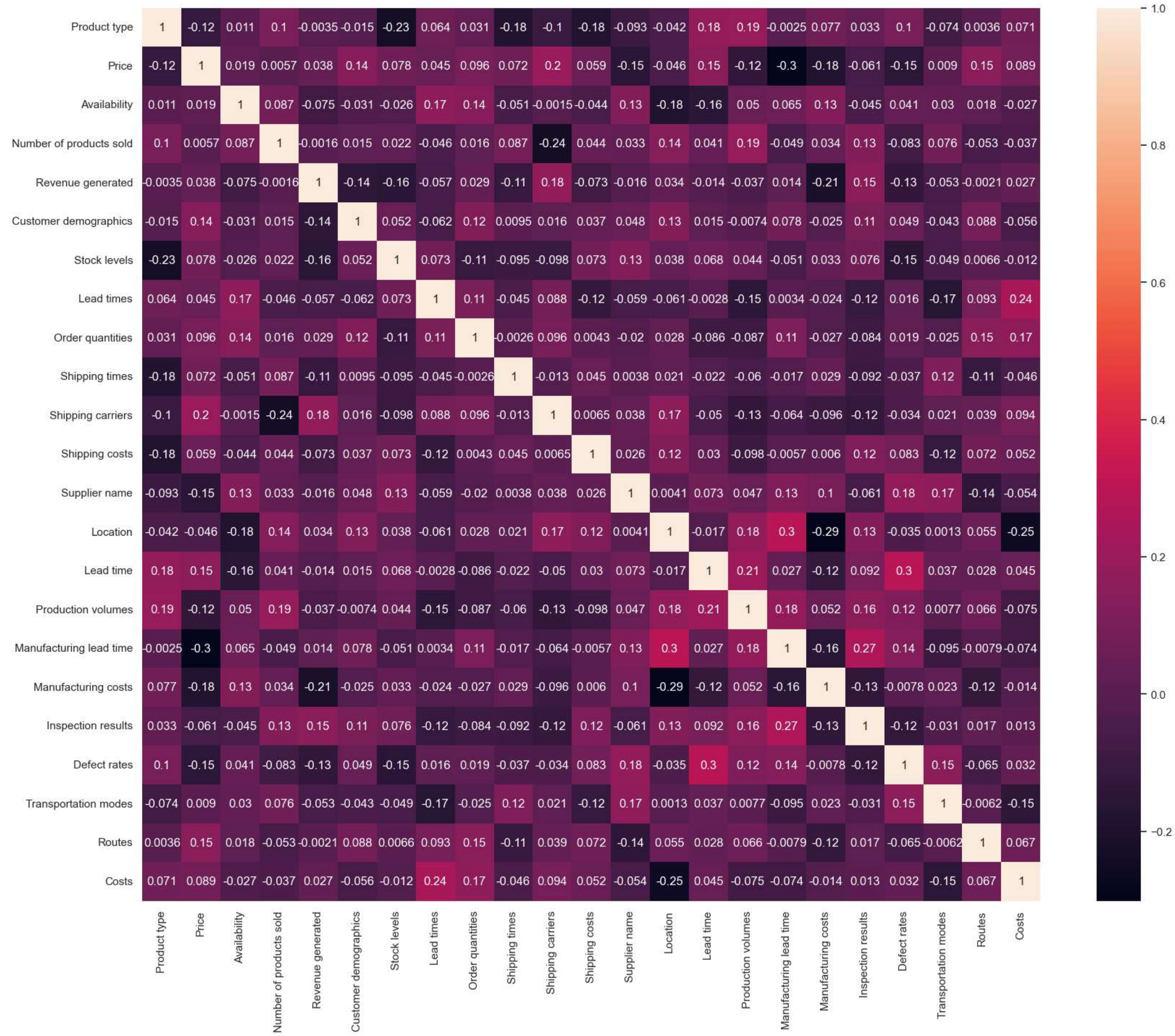
Out[16]:
```
Product type                int32
Price                     float64
Availability                int64
Number of products sold     int64
Revenue generated         float64
Customer demographics       int32
Stock levels                int64
Lead times                  int64
Order quantities            int64
Shipping times              int64
Shipping carriers           int32
Shipping costs            float64
Supplier name               int32
Location                    int32
Lead time                   int64
Production volumes          int64
Manufacturing lead time     int64
Manufacturing costs       float64
Inspection results          int32
Defect rates              float64
Transportation modes        int32
Routes                      int32
Costs                     float64
dtype: object
```

There's no outlier so we dont have to remove it

Correlation Heatmap

In [17]:
```
#Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[17]: <AxesSubplot:>



Train test Split

In [50]:
```
X = df.drop('Costs', axis=1)
y = df['Costs']
```

In [51]:
```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0)
```

Decision Tree Regressor

In [52]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_boston


# Create a DecisionTreeRegressor object
dtree = DecisionTreeRegressor()

# Define the hyperparameters to tune and their values
param_grid = {
    'max_depth': [2, 4, 6, 8],
    'min_samples_split': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 3, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'random_state': [0, 7, 42]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(dtree, param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 2, 'max_features': 'sqrt', 'min_samples_leaf': 3, 'min_samples_split': 2, 'random_
state': 0}
```

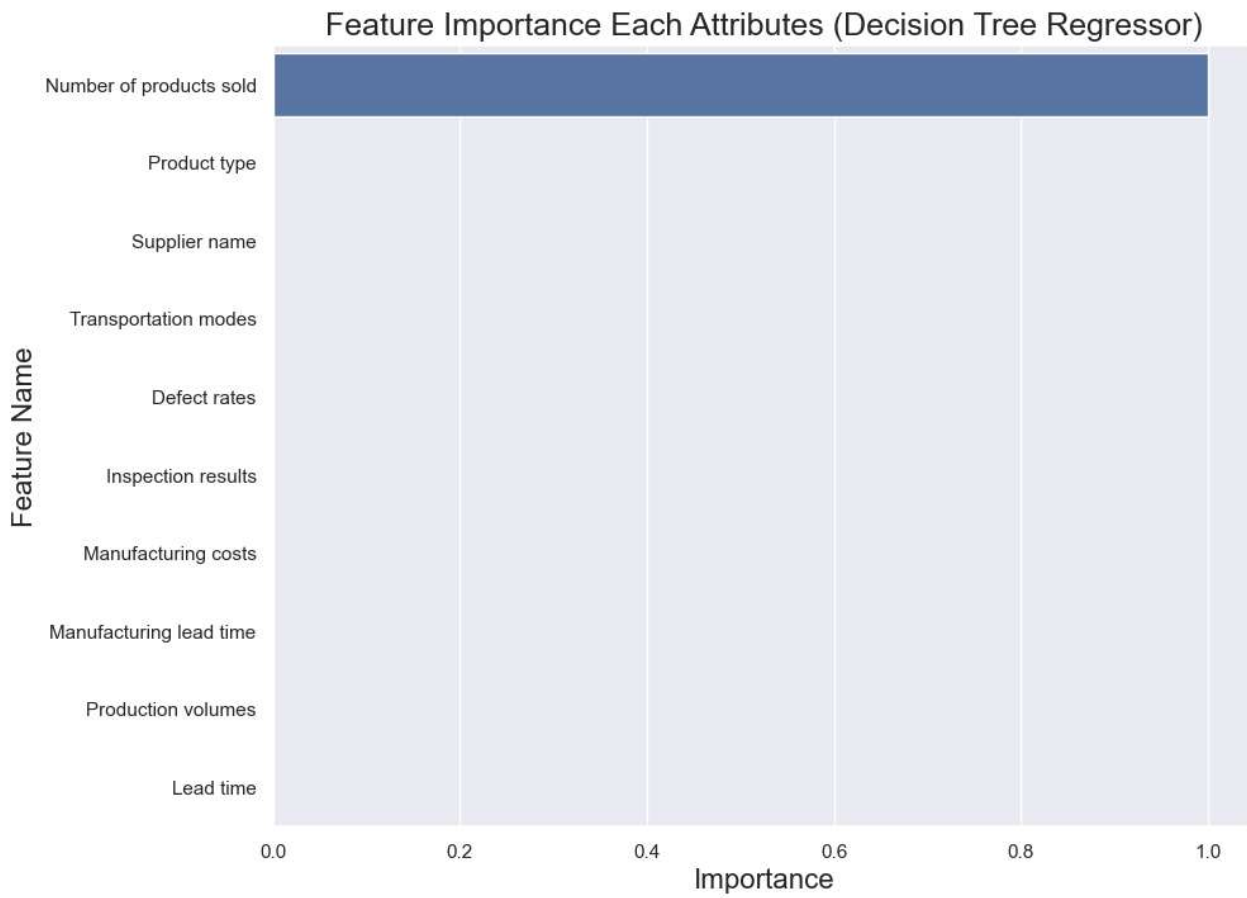
In [60]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=2, max_features='sqrt', min_samples_leaf=
dtree.fit(X_train, y_train)
```

Out[60]:
```
DecisionTreeRegressor(max_depth=2, max_features='sqrt', min_samples_leaf=3,
                      random_state=0)
```

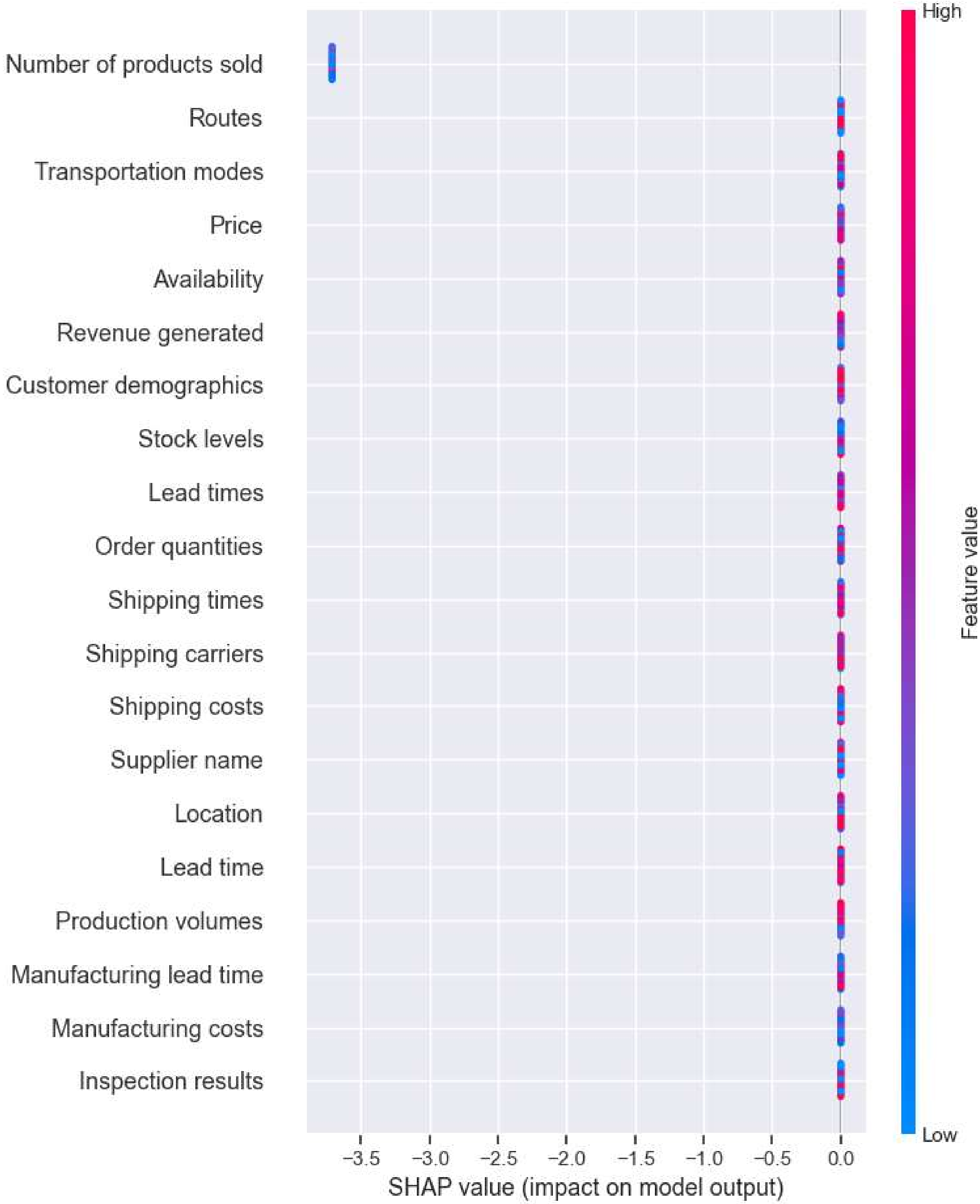
In [61]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 248.4413893861546
MAPE is 0.5893818876444419
MSE is 72806.47766651674
R2 score is -0.08647889188367719
RMSE score is 269.8267549123266
```

In [62]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Decision Tree Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [63]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



Random Forest Regressor

In [57]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Create a Random Forest Regressor object
rf = RandomForestRegressor()

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt'],
    'random_state': [0, 7, 42]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='r2')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'max_depth': 3, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samp
les_split': 5, 'random_state': 0}
```

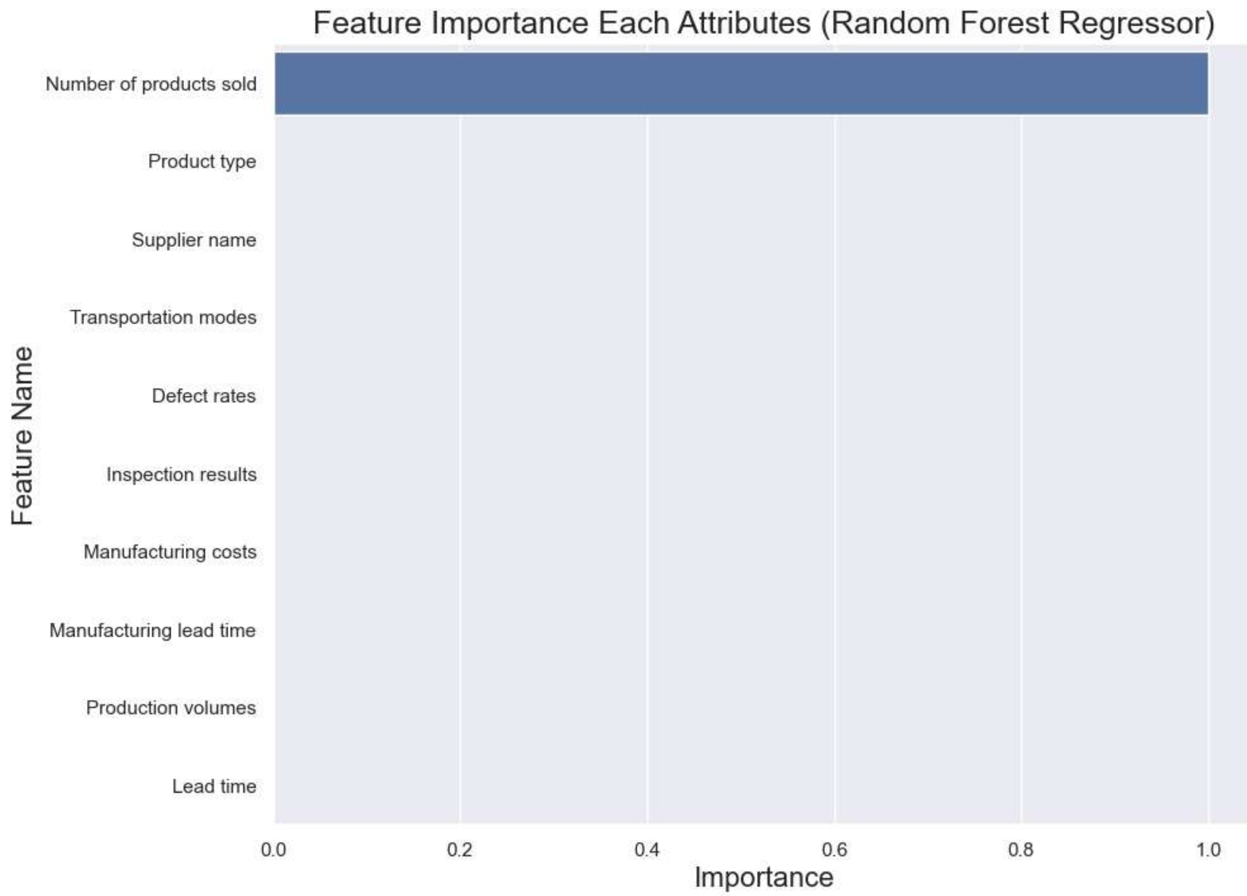
In [64]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=0, max_depth=3, min_samples_split=5, min_samples_leaf=2,
                           max_features='sqrt')
rf.fit(X_train, y_train)
```

Out[64]:
```
RandomForestRegressor(max_depth=3, max_features='sqrt', min_samples_leaf=2,
                      min_samples_split=5, random_state=0)
```

In [65]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = rf.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 247.33969719962744
MAPE is 0.6029768224226728
MSE is 71899.28833186119
R2 score is -0.07294105713825938
RMSE score is 268.14042651540103
```

In [66]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Random Forest Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [67]:
```
import shap
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

AdaBoost Regressor

In [68]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import GridSearchCV

# Create an AdaBoost Regressor object
ada = AdaBoostRegressor()

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 150, 200],
    'learning_rate': [0.01, 0.1, 1],
    'loss': ['linear', 'square', 'exponential'],
    'random_state': [0, 7, 42]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(ada, param_grid, cv=5, scoring='r2')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'learning_rate': 1, 'loss': 'linear', 'n_estimators': 50, 'random_stat
e': 7}
```

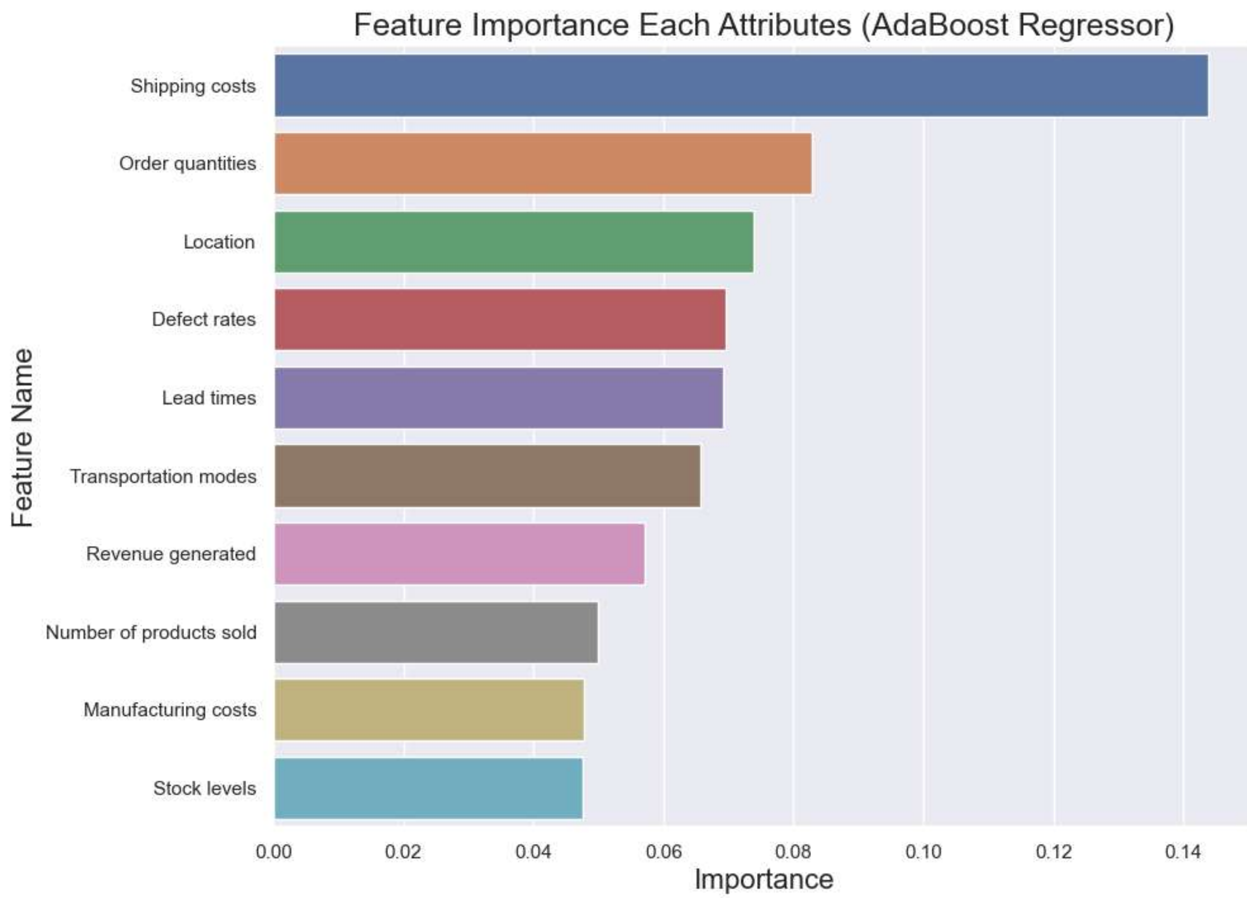
In [69]:
```
from sklearn.ensemble import AdaBoostRegressor
ada = AdaBoostRegressor(random_state=7, n_estimators=50, learning_rate=1, loss='linear')
ada.fit(X_train, y_train)
```

Out[69]:
```
AdaBoostRegressor(learning_rate=1, random_state=7)
```

In [70]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = ada.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 255.28612180541396
MAPE is 0.5859844238678936
MSE is 78800.74415400976
R2 score is -0.17593032834538103
RMSE score is 280.71470241868303
```

In [72]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": ada.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (AdaBoost Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



All of the Algorithms got bad R2 Score and MAPE Score even with hyperparameter tuning because we only have 100 data and the distribution is spread