Anomaly Detection of Retail Store Sales

This hands-on mini-project will enable you to reinforce your learnings pertaining to anomaly detection in this unit. By now, you must already be aware of the key objective of anomaly detection. Just to refresh your memory, anomaly detection is the identification of outliers or rare event items in a dataset which potentially exhibit abnormal behavior or properties as compared to the rest of the datapoints.

There are a wide variety of anomaly detection methods including supervised, unsupervised and semi-supervised. Typically you can perform anomaly detection on univariate data, multivariate data as well as data which is temporal in nature. In this mini-project you will leverage state-of-the-art anomaly detection models from frameworks like **`scikit-learn`** and **`PyOD`** .

By the end of this mini-project, you will have successfully applied these techniques to find out potential outliers pertaining to sales transactional data in a retail store dataset and also learnt how to visualize outliers similar to the following plot.

We will be performing anomaly detection on both univariate and multivariate data and leverage the following anomaly detection techniques.

- Simple Statistical Models (mean & standard deviation: the three-sigma rule)
- Isolation Forest
- Clustering-Based Local Outlier Factor
- Auto-encoders

1. Getting and Loading the Dataset

The first step towards solving any data science or machine learning problem is to obtain the necessary data. In this scenario, we will be dealing with a popular retail dataset known as the SuperStore Sales Dataset which consists of transactional data pertaining to a retail store.

Please download the required dataset from here if necessary, although it will also be provided to you along with this notebook for this mini-project

Once we have the necessary data, we will load up the dataset and perform some initial exploratory data analysis

2. Exploratory Data Analysis

It's time to do some basic exploratory analysis on the retail store transactional data. We start by loading up the dataset into a pandas dataframe.

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline


df = pd.read_excel("./Superstore.xls")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   Row ID         9994 non-null   int64         
 1   Order ID       9994 non-null   object        
 2   Order Date     9994 non-null   datetime64[ns]
 3   Ship Date      9994 non-null   datetime64[ns]
 4   Ship Mode      9994 non-null   object        
 5   Customer ID    9994 non-null   object        
 6   Customer Name  9994 non-null   object        
 7   Segment        9994 non-null   object        
 8   Country        9994 non-null   object        
 9   City           9994 non-null   object        
 10  State          9994 non-null   object        
 11  Postal Code    9994 non-null   int64         
 12  Region         9994 non-null   object        
 13  Product ID     9994 non-null   object        
 14  Category       9994 non-null   object        
 15  Sub-Category   9994 non-null   object        
 16  Product Name   9994 non-null   object        
 17  Sales          9994 non-null   float64       
 18  Quantity       9994 non-null   int64         
 19  Discount       9994 non-null   float64       
 20  Profit         9994 non-null   float64       
dtypes: datetime64[ns](2), float64(3), int64(3), object(13)
memory usage: 1.6+ MB
```

We don't have any major missing values in our dataset and we can now look at a sample subset of the data

In [3]:
```
df.head()
```

Out[3]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	..
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	..
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	..

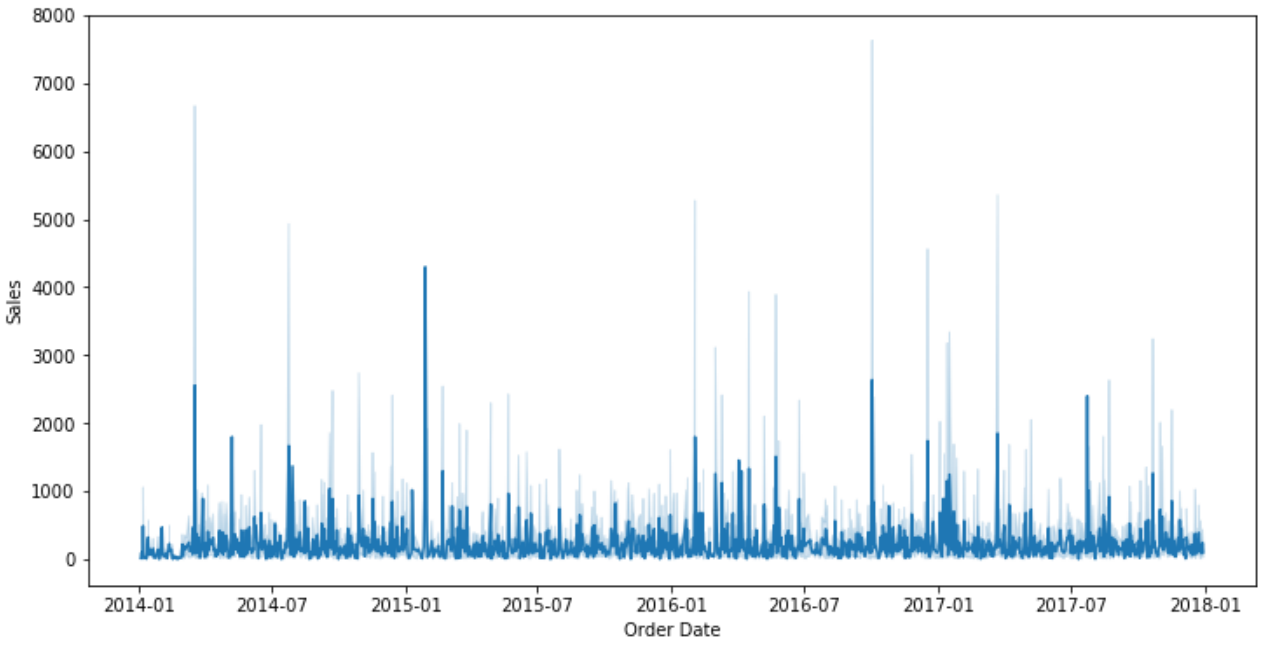
	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	..
2	3	CA-2016-138688	2016-06-12	2016-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	..
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	..
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	..

5 rows × 21 columns

Visualize Sales vs. Order Date

Let's look more closely at the **Sales** attribute of the dataset in the next few cells. We'll start by looking at typical sales over time

In [4]:
```
fig, ax = plt.subplots(1, 1, figsize=(12, 6))
sns.lineplot(x=df['Order Date'], y=df['Sales']);
```

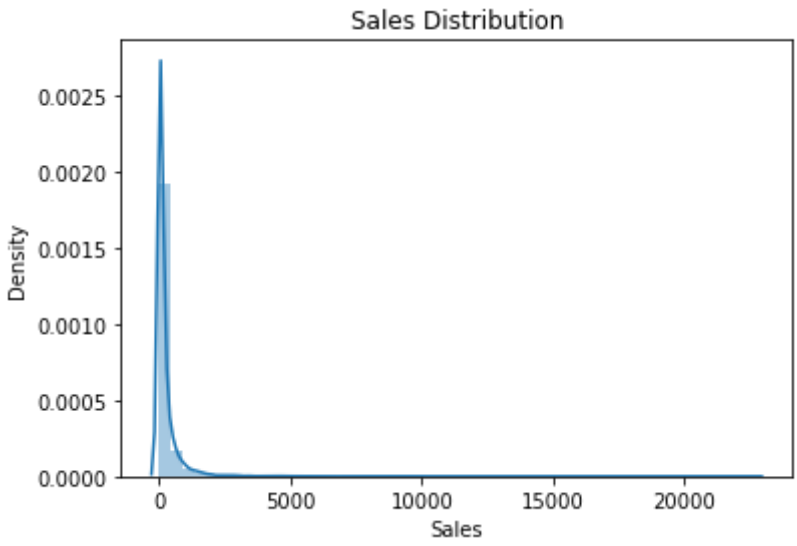


Visualize Sales Distribution

Let's now look at the data distribution for **Sales**

In [5]:
```
sns.distplot(df['Sales'])
```

```
plt.title("Sales Distribution");
```



In [6]:
```
df['Sales'].describe()
```

Out[6]:
```
count     9994.000000
mean       229.858001
std        623.245101
min          0.444000
25%         17.280000
50%         54.490000
75%        209.940000
max      22638.480000
Name: Sales, dtype: float64
```

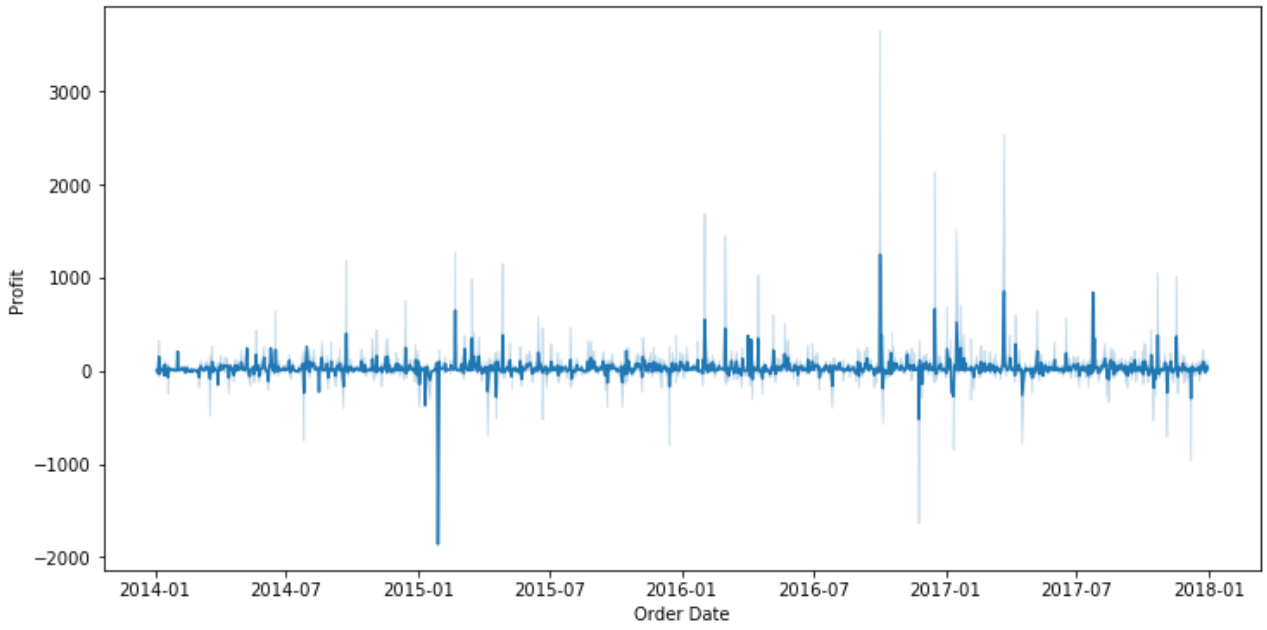
We can definitely see the presence of potential outliers in terms of the min or max values as compared to the meat of the distribution in the interquartile range as observed in the distribution statistics

Q 2.1: Visualize Profit vs. Order Date

Let's now look closely at the **`Profit`** attribute of the dataset in the next few cells. We'll start by looking at typical profits over time.

Your turn: Plot `Order Date` vs. `Profit` using a line plot

In [7]:
```
fig, ax = plt.subplots(1, 1, figsize=(12, 6))
sns.lineplot(x=df['Order Date'], y=df['Profit']);
```

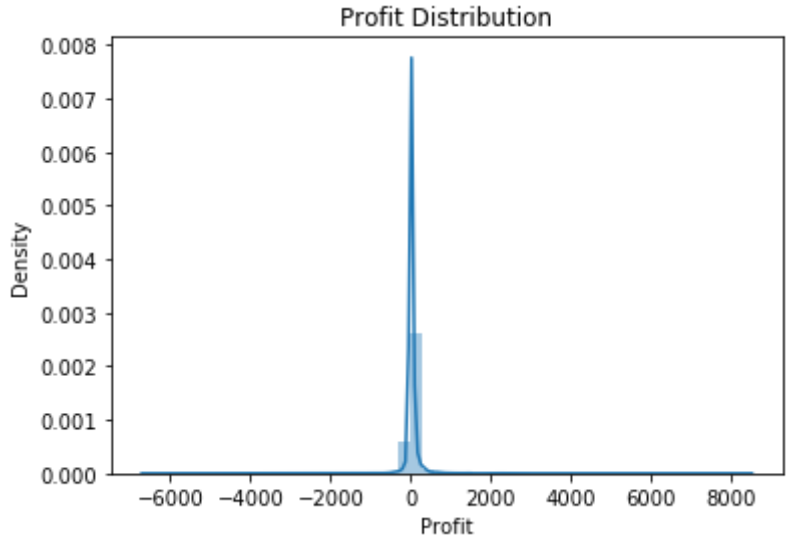


Q 2.2: Visualize Profit Distribution

Let's now look at the data distribution for **`Profit`**

Your turn: Plot the distribution for `Profit`

In [8]:
```
sns.distplot(df['Profit'])
plt.title("Profit Distribution");
```



Your turn: Get the essential descriptive statistics for `Profit` using an appropriate function

In [9]:
```
df['Profit'].describe()
```

Out[9]:
```
count    9994.000000
mean       28.656896
std       234.260108
min     -6599.978000
```

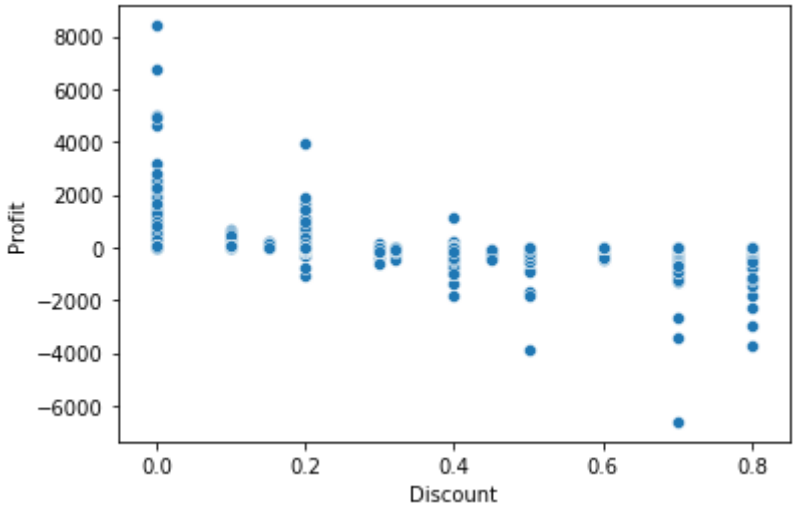
```
25%          1.728750
50%          8.666500
75%         29.364000
max       8399.976000
Name: Profit, dtype: float64
```

Your turn: Do you notice anything interesting about the distribution?

We have both positive and negative values in profits since it indicates either a profit or a loss based on the sales and original price of the items.

Visualize Discount vs. Profit

In [10]:
```
sns.scatterplot(x="Discount", y="Profit", data=df);
```



In the above visual, we look at a scatter plot showing the distribution of profits w.r.t discounts given

3. Univariate Anomaly Detection

Univariate is basically analysis done on a single attribute or feature. In this section, we will perform anomaly detection on a single attribute using the following methods.

- Statistical Process Control Methods (mean + 3sigma thresholding)
- Isolation Forest

We will start off by demonstrating both these techniques on the **`Sales`** attribute and later on, you will implement similar techniques on the **`Profit`** attribute.

3.1: Univariate Anomaly Detection on Sales using Statistical Modeling

Here we start off by implementing anomaly detecting using statistical modeling on the **`Sales`** attribute

Obtain Upper Limit Threshold for Sales

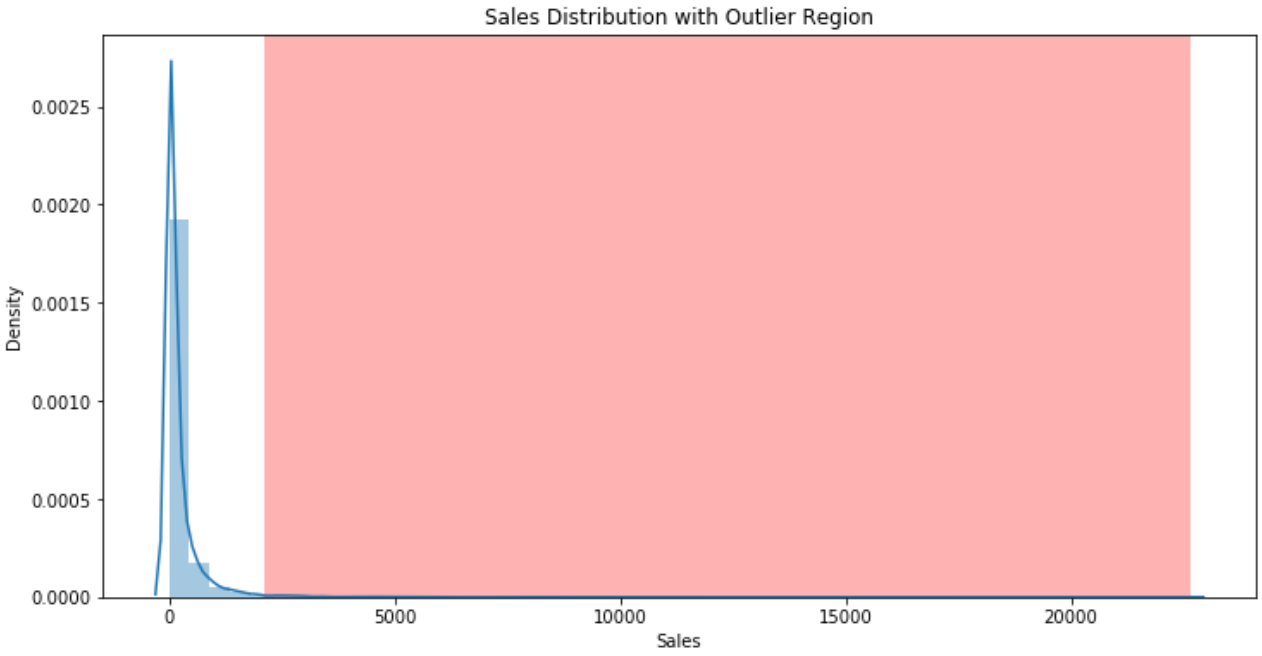
Here we are concerned about transactions with high sales values so we compute the upper limit using the $\mu + 3\sigma$ rule where μ is the mean of the distribution and σ is the standard deviation of the distribution.

In [11]:
```
mean_sales = df['Sales'].mean()
sigma_sales = df['Sales'].std()
three_sigma_sales = 3*sigma_sales

threshold_sales_value = mean_sales + three_sigma_sales
print('Threshold Sales:', threshold_sales_value)
```

```
Threshold Sales: 2099.593302356541
```

Visualize Outlier Region

In [12]:
```
fig, ax = plt.subplots(1, 1, figsize=(12, 6))

sns.distplot(df['Sales'])
plt.axvspan(threshold_sales_value, df['Sales'].max(), facecolor='r', alpha=0.3)
plt.title("Sales Distribution with Outlier Region");
```



Filter and Sort Outliers

Here we filter out the outlier observations and sort by descending order and view the top 5 outlier values

In [13]:
```
sales_outliers_df = df['Sales'][df['Sales'] > threshold_sales_value]
print('Total Sales Outliers:', len(sales_outliers_df))
sales_outliers_sorted = sales_outliers_df.sort_values(ascending=False)
sales_outliers_sorted.head(5)
```

```
Total Sales Outliers: 127
```

Out[13]:
```
2697    22638.480
6826    17499.950
8153    13999.960
2623    11199.968
4190    10499.970
Name: Sales, dtype: float64
```

View Top 10 Outlier Transactions

In [14]:
```
(df.loc[sales_outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Category
                                               'Sales', 'Quantity', 'Discount', '
```

Out[14]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	P
2697	Jacksonville	Technology	Machines	Cisco TelePresence System EX90 Videoconferenci...	22638.480	6	0.5	-1811.
6826	Lafayette	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	17499.950	5	0.0	8399.
8153	Seattle	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	13999.960	4	0.0	6719.
2623	New York City	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	11199.968	4	0.2	3919.
4190	Newark	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	10499.970	3	0.0	5039.
9039	Detroit	Office Supplies	Binders	GBC Ibimaster 500 Manual ProClick Binding System	9892.740	13	0.0	4946.
4098	Minneapolis	Office Supplies	Binders	Ibico EPK-21 Electric Binding System	9449.950	5	0.0	4630.
4277	Lakewood	Technology	Machines	3D Systems Cube Printer, 2nd Generation, Magenta	9099.930	7	0.0	2365.
8488	Arlington	Technology	Machines	HP Designjet T520 Inkjet Large Format Printer ...	8749.950	5	0.0	2799.
6425	Philadelphia	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	8399.976	4	0.4	1119.

View Bottom 10 Outlier Transactions

In [15]:
```
(df.loc[sales_outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Category
                                               'Sales', 'Quantity', 'Discount', '
```

Out[15]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	Pro
5300	Springfield	Furniture	Tables	Hon Rectangular Conference Tables	2275.500	10	0.0	386.83
6101	New York City	Furniture	Chairs	Global Troy Executive Leather Low-Back Tilter	2254.410	5	0.1	375.73
4881	Henderson	Technology	Accessories	Logitech diNovo Edge Keyboard	2249.910	9	0.0	517.47
7487	Dover	Technology	Accessories	Logitech diNovo Edge Keyboard	2249.910	9	0.0	517.47
1155	Harrisonburg	Furniture	Tables	Chromcraft 48" x 96" Racetrack Double Pedestal...	2244.480	7	0.0	493.78
8699	Nashville	Technology	Phones	Samsung Galaxy S III - 16GB - pebble blue (T-M...	2239.936	8	0.2	223.99
9774	San Antonio	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	2177.584	8	0.8	-3701.89
8424	Lakewood	Furniture	Bookcases	Global Adaptabilites Bookcase, Cherry/Storm Gr...	2154.900	5	0.0	129.29
6534	Buffalo	Office Supplies	Binders	GBC DocuBind TL300 Electric Binding System	2152.776	3	0.2	726.56
8680	Richmond	Office Supplies	Appliances	Honeywell Enviracaire Portable HEPA Air Cleane...	2104.550	7	0.0	694.50

Q 3.2: Univariate Anomaly Detection on Profit using Statistical Modeling

In this section you will use the learning from Section 3.1 and implement anomaly detecting using statistical modeling on the **`Profit`** attribute. Since we have both +ve (profits) and -ve (losses) values in the distribution, we will try to find anomalies for each.

Obtain Upper Limit Threshold for Profit

Your turn: Compute the upper and lower limits using the μ + 3 σ rule where μ is the mean of the distribution and σ is the standard deviation of the distribution.

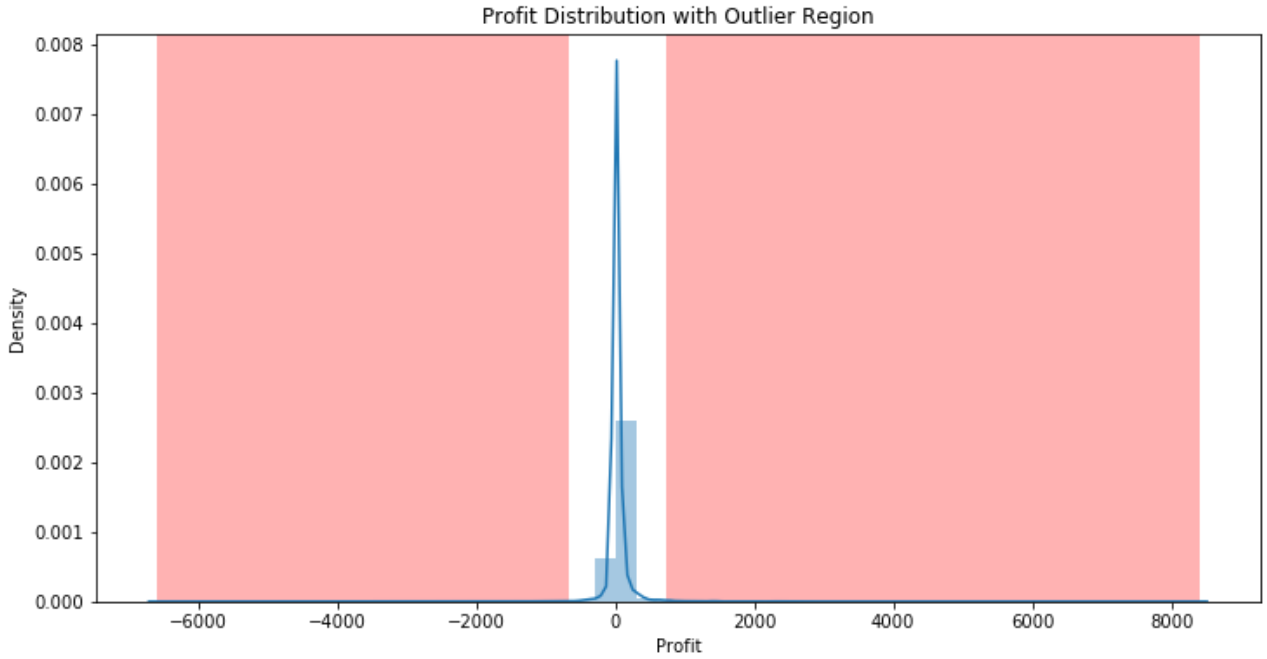
In [16]:
```
mean_profit = df['Profit'].mean()
sigma_profit = df['Profit'].std()
three_sigma_profit = 3*sigma_profit

threshold_profit_upper_limit = mean_profit + three_sigma_profit
threshold_profit_lower_limit = mean_profit - three_sigma_profit

print('Thresholds Profit:', threshold_profit_lower_limit, threshold_profit_upper
```

```
Thresholds Profit: -674.1234267650873 731.4372193806566
```

Visualize Outlier Regions

Your turn: Visualize the upper and lower outlier regions in the distribution similar to what you did in 3.1

In [17]:
```
fig, ax = plt.subplots(1, 1, figsize=(12, 6))

sns.distplot(df['Profit'])
plt.axvspan(threshold_profit_upper_limit, df['Profit'].max(), facecolor='r', alp
plt.axvspan(threshold_profit_lower_limit, df['Profit'].min(), facecolor='r', alp
plt.title("Profit Distribution with Outlier Region");
```



Filter and Sort Outliers

Your turn: Filter out the outlier observations and sort by descending order and view the top 5 outlier values

In [18]:
```
profit_outliers_df = df['Profit'][(df['Profit'] > threshold_profit_upper_limit)
print('Total Profit Outliers:', len(profit_outliers_df))
profit_outliers_sorted = profit_outliers_df.sort_values(ascending=False)
profit_outliers_sorted.head(5)
```

```
Total Profit Outliers: 107
```

Out[18]:
```
6826    8399.9760
8153    6719.9808
4190    5039.9856
9039    4946.3700
4098    4630.4755
Name: Profit, dtype: float64
```

View Top 10 Outlier Transactions

Your turn: View the top ten transactions based on highest profits

In [19]:
```
(df.loc[profit_outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Categor
                                                'Sales', 'Quantity', 'Discount', '
```

Out[19]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	Profit
6826	Lafayette	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	17499.950	5	0.0	8399.9760

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	Profit
8153	Seattle	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	13999.960	4	0.0	6719.9808
4190	Newark	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	10499.970	3	0.0	5039.9856
9039	Detroit	Office Supplies	Binders	GBC Ibimaster 500 Manual ProClick Binding System	9892.740	13	0.0	4946.3700
4098	Minneapolis	Office Supplies	Binders	Ibico EPK-21 Electric Binding System	9449.950	5	0.0	4630.4755
2623	New York City	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	11199.968	4	0.2	3919.9888
509	Atlanta	Office Supplies	Binders	Fellowes PB500 Electric Punch Plastic Comb Bin...	6354.950	5	0.0	3177.4750
8488	Arlington	Technology	Machines	HP Designjet T520 Inkjet Large Format Printer ...	8749.950	5	0.0	2799.9840
7666	Providence	Technology	Copiers	Hewlett Packard LaserJet 3310 Copier	5399.910	9	0.0	2591.9568
6520	Jackson	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	5443.960	4	0.0	2504.2216

Q: Do you notice any interesting insights based on these transactions?

A: Most of these are purchases for Copiers and Binders , looks like Canon products yielded some good profits`

View Bottom 10 Outlier Transactions

Your turn: View the bottom ten transactions based on lowest profits (highest losses)

In [20]:
```
(df.loc[profit_outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Categor
                                                'Sales', 'Quantity', 'Discount', '
```

Out[20]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	
2697	Jacksonville	Technology	Machines	Cisco TelePresence System EX90 Videoconferenci...	22638.480	6	0.5	-1811
1199	Houston	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	1088.792	4	0.8	-1850
9639	Concord	Furniture	Tables	Chromcraft Bull-Nose Wood Oval Conference Tabl...	4297.644	13	0.4	-1862
5310	Houston	Office Supplies	Binders	Fellowes PB500 Electric Punch Plastic Comb Bin...	1525.188	6	0.8	-2287
3151	Newark	Technology	Machines	Cubify CubeX 3D Printer Double Head Print	1799.994	2	0.7	-2639
4991	Chicago	Office Supplies	Binders	Ibico EPK-21 Electric Binding System	1889.990	5	0.8	-2929
3011	Louisville	Technology	Machines	Lexmark MX611dhe Monochrome Laser Printer	2549.985	5	0.7	-3399
9774	San Antonio	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	2177.584	8	0.8	-3701
683	Burlington	Technology	Machines	Cubify CubeX 3D Printer Triple Head Print	7999.980	4	0.5	-3839
7772	Lancaster	Technology	Machines	Cubify CubeX 3D Printer Double Head Print	4499.985	5	0.7	-6599

Q: Do you notice any interesting insights based on these transactions?

A: Most of these are purchases for Machines and Binders , looks like Cibify 3D Printers yielded high losses

3.3: Univariate Anomaly Detection on Sales using

Isolation Forest

You might have already learnt about this model from the curriculum. Just to briefly recap, the Isolation Forest model, 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

Recursive partitioning can be represented by a tree structure. Hence, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node. This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.

Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.

More details are available in this User Guide

Initialize and Train Model

Here we initialize the isolation forest model with some hyperparameters assuming the proportion of outliers to be 1% of the total data (using the `contamination` setting)

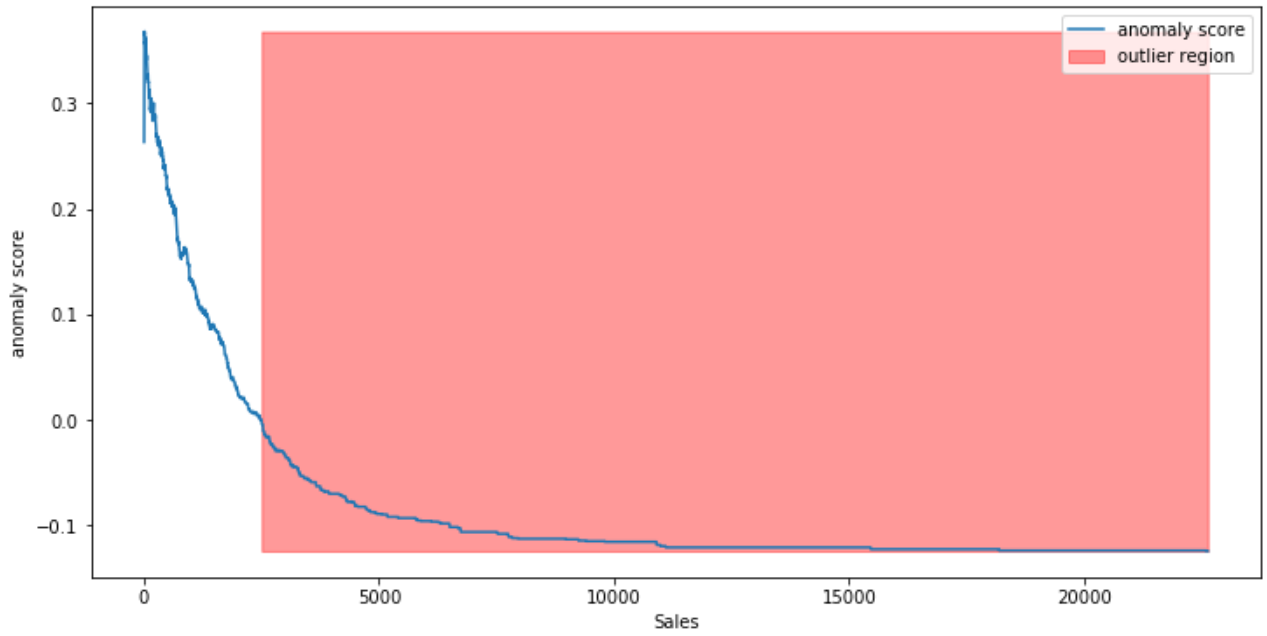
In [21]:
```
from sklearn.ensemble import IsolationForest

sales_ifmodel = IsolationForest(n_estimators=100,
                                contamination=0.01)
sales_ifmodel.fit(df[['Sales']])
```

Out[21]:
```
IsolationForest(contamination=0.01)
```

Visualize Outlier Region

Here we visualize the outlier region in the data distribution

In [22]:
```
xx = np.linspace(df['Sales'].min(), df['Sales'].max(), len(df)).reshape(-1,1)
anomaly_score = sales_ifmodel.decision_function(xx)
outlier = sales_ifmodel.predict(xx)
plt.figure(figsize=(12, 6))
plt.plot(xx, anomaly_score, label='anomaly score')
plt.fill_between(xx.T[0], np.min(anomaly_score), np.max(anomaly_score),
                 where=outlier==-1, color='r',
                 alpha=.4, label='outlier region')
plt.legend()
plt.ylabel('anomaly score')
plt.xlabel('Sales');
```



Filter and Sort Outliers

Here we predict outliers in our dataset using our trained model and filter out the outlier observations and sort by descending order and view the top 5 outlier values

In [23]:
```
outlier_predictions = sales_ifmodel.predict(df[['Sales']])

sales_outliers_df = df[['Sales']]
sales_outliers_df['Outlier'] = outlier_predictions
sales_outliers_df = sales_outliers_df[sales_outliers_df['Outlier'] == -1]['Sales

print('Total Sales Outliers:', len(sales_outliers_df))
sales_outliers_sorted = sales_outliers_df.sort_values(ascending=False)
sales_outliers_sorted.head(5)
```

```
Total Sales Outliers: 100
```

Out[23]:
```
2697    22638.480
6826    17499.950
8153    13999.960
2623    11199.968
4190    10499.970
Name: Sales, dtype: float64
```

View Top 10 Outlier Transactions

In [24]:
```
(df.loc[sales_outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Category
                                               'Sales', 'Quantity', 'Discount', '
```

Out[24]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	F
2697	Jacksonville	Technology	Machines	Cisco TelePresence System EX90 Videoconferenci...	22638.480	6	0.5	-1811.

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	P
6826	Lafayette	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	17499.950	5	0.0	8399.
8153	Seattle	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	13999.960	4	0.0	6719.
2623	New York City	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	11199.968	4	0.2	3919.
4190	Newark	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	10499.970	3	0.0	5039.
9039	Detroit	Office Supplies	Binders	GBC Ibimaster 500 Manual ProClick Binding System	9892.740	13	0.0	4946.
4098	Minneapolis	Office Supplies	Binders	Ibico EPK-21 Electric Binding System	9449.950	5	0.0	4630.
4277	Lakewood	Technology	Machines	3D Systems Cube Printer, 2nd Generation, Magenta	9099.930	7	0.0	2365.
8488	Arlington	Technology	Machines	HP Designjet T520 Inkjet Large Format Printer ...	8749.950	5	0.0	2799.
6425	Philadelphia	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	8399.976	4	0.4	1119.

View Bottom 10 Outlier Transactions

In [25]:
```
(df.loc[sales_outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Category
                                               'Sales', 'Quantity', 'Discount', '
```

Out[25]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	P
1805	Los Angeles	Technology	Phones	Samsung Galaxy Note 2	2575.944	7	0.2	257.
7474	Henderson	Furniture	Chairs	Global Deluxe High-Back Manager's Chair	2573.820	9	0.0	746.
6884	Minneapolis	Furniture	Chairs	Hon Pagoda Stacking Chairs	2567.840	8	0.0	770.

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	P
8271	New York City	Furniture	Chairs	Hon 4700 Series Mobuis Mid-Back Task Chairs wi...	2563.056	8	0.1	313.2
3011	Louisville	Technology	Machines	Lexmark MX611dhe Monochrome Laser Printer	2549.985	5	0.7	-3399.9
6010	Los Angeles	Technology	Machines	Zebra GX420t Direct Thermal/Thermal Transfer P...	2548.560	6	0.2	286.
7280	Columbia	Office Supplies	Binders	Fellowes PB500 Electric Punch Plastic Comb Bin...	2541.980	2	0.0	1270.9
263	Houston	Technology	Machines	Xerox WorkCentre 6505DN Laser Multifunction Pr...	2519.958	7	0.4	-251.9
7937	Brentwood	Office Supplies	Appliances	Sanyo 2.5 Cubic Foot Mid-Size Office Refrigera...	2518.290	9	0.0	654.7
3443	New York City	Office Supplies	Appliances	Hoover Shoulder Vac Commercial Portable Vacuum	2504.740	7	0.0	626.

Q 3.4: Univariate Anomaly Detection on Profit using Isolation Forest

In this section you will use the learning from Section 3.3 and implement anomaly detecting using isolation on the **`Profit`** attribute. Since we have both +ve (profits) and -ve (losses) values in the distribution, we will try to find anomalies for each.

Initialize and Train Model

Your Turn: Initialize the isolation forest model with similar hyperparameters as Section 3.3 and also assuming the proportion of outliers to be 1% of the total data (using the contamination setting)

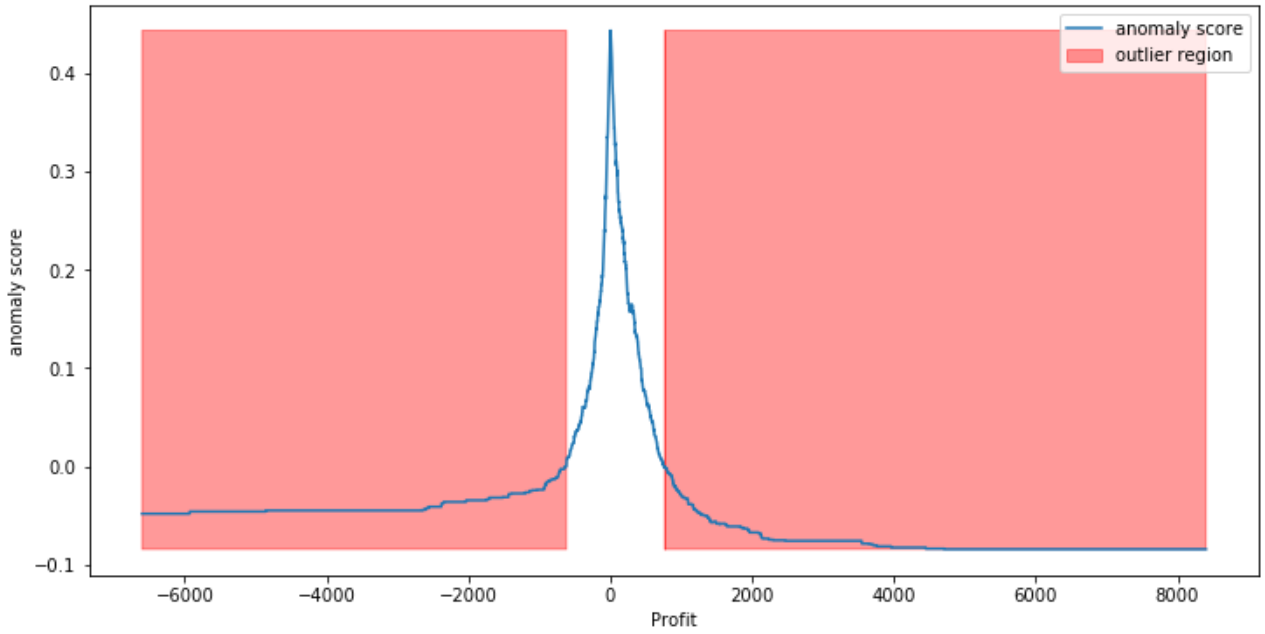
In [26]:
```
profit_ifmodel = IsolationForest(n_estimators=100,
                                 contamination=0.01)
profit_ifmodel.fit(df[['Profit']])
```

Out[26]:
```
IsolationForest(contamination=0.01)
```

Visualize Outlier Regions

Your turn: Visualize the upper and lower outlier regions in the distribution similar to what you did in 3.3

In [27]:
```
xx = np.linspace(df['Profit'].min(), df['Profit'].max(), len(df)).reshape(-1,1)
anomaly_score = profit_ifmodel.decision_function(xx)
outlier = profit_ifmodel.predict(xx)
plt.figure(figsize=(12, 6))
plt.plot(xx, anomaly_score, label='anomaly score')
plt.fill_between(xx.T[0], np.min(anomaly_score), np.max(anomaly_score),
                 where=outlier==-1, color='r',
                 alpha=.4, label='outlier region')
plt.legend()
plt.ylabel('anomaly score')
plt.xlabel('Profit');
```



Filter and Sort Outliers

Your Turn: Predict outliers in our dataset using our trained model and filter out the outlier observations and sort by descending order and view the top 5 outlier values similar to 3.3

In [28]:
```
outlier_predictions = profit_ifmodel.predict(df[['Profit']])

profit_outliers_df = df[['Profit']]
profit_outliers_df['Outlier'] = outlier_predictions
profit_outliers_df = profit_outliers_df[profit_outliers_df['Outlier'] == -1]['Pr

print('Total Profit Outliers:', len(profit_outliers_df))
profit_outliers_sorted = profit_outliers_df.sort_values(ascending=False)
profit_outliers_sorted.head(5)
```

```
Total Profit Outliers: 98
```

Out[28]:
```
6826    8399.9760
8153    6719.9808
4190    5039.9856
```

```
9039    4946.3700
4098    4630.4755
Name: Profit, dtype: float64
```

View Top 10 Outlier Transactions

Your turn: View the top ten transactions based on highest profits

In [29]:
```
(df.loc[profit_outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Categor
                                                'Sales', 'Quantity', 'Discount', '
```

Out[29]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	Profit
6826	Lafayette	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	17499.950	5	0.0	8399.9760
8153	Seattle	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	13999.960	4	0.0	6719.9808
4190	Newark	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	10499.970	3	0.0	5039.9856
9039	Detroit	Office Supplies	Binders	GBC Ibimaster 500 Manual ProClick Binding System	9892.740	13	0.0	4946.3700
4098	Minneapolis	Office Supplies	Binders	Ibico EPK-21 Electric Binding System	9449.950	5	0.0	4630.4755
2623	New York City	Technology	Copiers	Canon imageCLASS 2200 Advanced Copier	11199.968	4	0.2	3919.9888
509	Atlanta	Office Supplies	Binders	Fellowes PB500 Electric Punch Plastic Comb Bin...	6354.950	5	0.0	3177.4750
8488	Arlington	Technology	Machines	HP Designjet T520 Inkjet Large Format Printer ...	8749.950	5	0.0	2799.9840

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	Profit
7666	Providence	Technology	Copiers	Hewlett Packard LaserJet 3310 Copier	5399.910	9	0.0	2591.9568
6520	Jackson	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	5443.960	4	0.0	2504.2216

View Bottom 10 Outlier Transactions

Your turn: View the bottom ten transactions based on lowest profits (highest losses)

In [30]:
```
(df.loc[profit_outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Categor
                                                'Sales', 'Quantity', 'Discount', '
```

Out[30]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	
2697	Jacksonville	Technology	Machines	Cisco TelePresence System EX90 Videoconferenci...	22638.480	6	0.5	-1811
1199	Houston	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	1088.792	4	0.8	-1850
9639	Concord	Furniture	Tables	Chromcraft Bull-Nose Wood Oval Conference Tabl...	4297.644	13	0.4	-1862
5310	Houston	Office Supplies	Binders	Fellowes PB500 Electric Punch Plastic Comb Bin...	1525.188	6	0.8	-2287
3151	Newark	Technology	Machines	Cubify CubeX 3D Printer Double Head Print	1799.994	2	0.7	-2639
4991	Chicago	Office Supplies	Binders	Ibico EPK-21 Electric Binding System	1889.990	5	0.8	-2929
3011	Louisville	Technology	Machines	Lexmark MX611dhe Monochrome Laser Printer	2549.985	5	0.7	-3399
9774	San Antonio	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	2177.584	8	0.8	-3701

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	
683	Burlington	Technology	Machines	Cubify CubeX 3D Printer Triple Head Print	7999.980	4	0.5	-3839
7772	Lancaster	Technology	Machines	Cubify CubeX 3D Printer Double Head Print	4499.985	5	0.7	-6599

Q: Do you observe any similarity in the results with the previous method?

A: Yes

Another interesting approach to check out would be the Generalized ESD Test for Outliers

4. Multivariate Anomaly Detection

Multivariate is basically analysis done on more than one attribute or feature at a time. In this section, we will perform anomaly detection on two attributes (**`Discount`** & **`Profit`**) using the following methods.

- Clustering Based Local Outlier Factor (CBLOF)
- Isolation Forest
- Auto-Encoders

You will learn how to train these models to detect outliers and also visualize these outliers. For this section we will be using the **`pyod`** package so make sure you have it installed.

In [31]:
```
!pip install pyod
```

```
Collecting pyod
  Downloading pyod-0.8.9.tar.gz (104 kB)
     |████████████████████████████████| 104 kB 3.8 MB/s eta 0:00:01
Requirement already satisfied: joblib in /Users/austinf1/miniconda3/lib/python3.
8/site-packages (from pyod) (1.0.1)
Requirement already satisfied: matplotlib in /Users/austinf1/miniconda3/lib/pyth
on3.8/site-packages (from pyod) (3.1.1)
Requirement already satisfied: numpy>=1.13 in /Users/austinf1/miniconda3/lib/pyt
hon3.8/site-packages (from pyod) (1.19.2)
Collecting numba>=0.35
  Downloading numba-0.53.1-cp38-cp38-macosx_10_14_x86_64.whl (2.2 MB)
     |████████████████████████████████| 2.2 MB 38.6 MB/s eta 0:00:01
Requirement already satisfied: pandas>=0.25 in /Users/austinf1/miniconda3/lib/py
thon3.8/site-packages (from pyod) (1.2.3)
Requirement already satisfied: scipy>=0.19.1 in /Users/austinf1/miniconda3/lib/p
ython3.8/site-packages (from pyod) (1.6.3)
Requirement already satisfied: scikit_learn>=0.19.1 in /Users/austinf1/miniconda
3/lib/python3.8/site-packages (from pyod) (0.24.2)
Requirement already satisfied: six in /Users/austinf1/miniconda3/lib/python3.8/s
ite-packages (from pyod) (1.15.0)
Requirement already satisfied: statsmodels in /Users/austinf1/miniconda3/lib/pyt
hon3.8/site-packages (from pyod) (0.12.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/austinf1/miniconda3/l
```

```
ib/python3.8/site-packages (from matplotlib->pyod) (1.3.1)
Requirement already satisfied: python-dateutil>=2.1 in /Users/austinf1/miniconda
3/lib/python3.8/site-packages (from matplotlib->pyod) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /User
s/austinf1/miniconda3/lib/python3.8/site-packages (from matplotlib->pyod) (2.4.
7)
Requirement already satisfied: cycler>=0.10 in /Users/austinf1/miniconda3/lib/py
thon3.8/site-packages (from matplotlib->pyod) (0.10.0)
Requirement already satisfied: setuptools in /Users/austinf1/miniconda3/lib/pyth
on3.8/site-packages (from numba>=0.35->pyod) (50.3.1.post20201107)
Collecting llvmlite<0.37,>=0.36.0rc1
  Downloading llvmlite-0.36.0-cp38-cp38-macosx_10_9_x86_64.whl (18.5 MB)
     |████████████████████████████████| 18.5 MB 13.7 MB/s eta 0:00:01
Requirement already satisfied: pytz>=2017.3 in /Users/austinf1/miniconda3/lib/py
thon3.8/site-packages (from pandas>=0.25->pyod) (2021.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/austinf1/miniconda
3/lib/python3.8/site-packages (from scikit_learn>=0.19.1->pyod) (2.1.0)
Requirement already satisfied: patsy>=0.5 in /Users/austinf1/miniconda3/lib/pyth
on3.8/site-packages (from statsmodels->pyod) (0.5.1)
Building wheels for collected packages: pyod
  Building wheel for pyod (setup.py) ... done
  Created wheel for pyod: filename=pyod-0.8.9-py3-none-any.whl size=121411 sha25
6=4c261f2e545e54f0e86a560ac28a5626f8405c0dda30a29870ccafa8aaa7810f
  Stored in directory: /Users/austinf1/Library/Caches/pip/wheels/8f/d9/6d/df101a
4fa21ac257176d4c6ff4e24edd9c1fd992e53a0a7535
Successfully built pyod
Installing collected packages: llvmlite, numba, pyod
Successfully installed llvmlite-0.36.0 numba-0.53.1 pyod-0.8.9
```

Extract Subset Data for Outlier Detection

In [32]:

```
cols = ['Discount', 'Profit']
subset_df = df[cols]
subset_df.head()
```

Out[32]:

	Discount	Profit
0	0.00	41.9136
1	0.00	219.5820
2	0.00	6.8714
3	0.45	-383.0310
4	0.20	2.5164

Feature Scaling

In [33]:

```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler(feature_range=(0, 1))
subset_df[cols] = mms.fit_transform(subset_df)
subset_df.head()
```

Out[33]:

	Discount	Profit
0	0.0000	0.442794

	Discount	Profit
1	0.0000	0.454639
2	0.0000	0.440458
3	0.5625	0.414464
4	0.2500	0.440168

4.1: Multivariate Anomaly Detection with Clustering Based Local Outlier Factor (CBLOF)

The CBLOF model takes as an input the dataset and the cluster model that was generated by a clustering algorithm. It classifies the clusters into small clusters and large clusters using the parameters alpha and beta. The anomaly score is then calculated based on the size of the cluster the point belongs to as well as the distance to the nearest large cluster.

By default, kMeans is used for clustering algorithm. You can read more in the official documentation

Initialize and Train Model

Here we initialize the CBLOF model with some hyperparameters assuming the proportion of outliers to be 1% of the total data (using the `contamination` setting)

In [34]:
```
from pyod.models import cblof

cblof_model = cblof.CBLOF(contamination=0.01, random_state=42)
cblof_model.fit(subset_df)
```

Out[34]:
```
CBLOF(alpha=0.9, beta=5, check_estimator=False, clustering_estimator=None,
   contamination=0.01, n_clusters=8, n_jobs=1, random_state=42,
   use_weights=False)
```

Filter and Sort Outliers

Here we predict outliers in our dataset using our trained model and filter out the outlier observations and sort by descending order and view the top 5 outlier values

In [35]:
```
outlier_predictions = cblof_model.predict(subset_df)

outliers_df = subset_df.copy(deep=True)
outliers_df['Outlier'] = outlier_predictions
outliers_df = outliers_df[outliers_df['Outlier'] == 1]

print('Total Outliers:', len(outliers_df))
outliers_sorted = outliers_df.sort_values(by=['Profit', 'Discount'], ascending=F
outliers_sorted.head(5)
```

```
Total Outliers: 100
```

Out[35]:

	Discount	Profit	Outlier

	Discount	Profit	Outlier
6826	0.0	1.000000	1
8153	0.0	0.888000	1
4190	0.0	0.776000	1
9039	0.0	0.769759	1
4098	0.0	0.748699	1

View Bottom 10 Outlier Transactions

In [36]:
```
(df.loc[outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Category', 'Pr
                                         'Sales', 'Quantity', 'Discount', '
```

Out[36]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	
463	Phoenix	Furniture	Tables	Bush Advantage Collection Racetrack Conference...	1272.630	6	0.5	-81
4128	Charlotte	Technology	Machines	HP Designjet T520 Inkjet Large Format Printer ...	2624.985	3	0.5	-94
8993	Columbia	Furniture	Tables	Balt Solid Wood Round Tables	1875.258	7	0.4	-96
5320	Knoxville	Furniture	Tables	Chromcraft Bull-Nose Wood Oval Conference Tabl...	2314.116	7	0.4	-100
165	San Antonio	Technology	Machines	Lexmark MX611dhe Monochrome Laser Printer	8159.952	8	0.4	-135
27	Philadelphia	Furniture	Bookcases	Riverside Palais Royal Lawyers Bookcase, Royal...	3083.430	7	0.5	-166
2697	Jacksonville	Technology	Machines	Cisco TelePresence System EX90 Videoconferenci...	22638.480	6	0.5	-181
9639	Concord	Furniture	Tables	Chromcraft Bull-Nose Wood Oval Conference Tabl...	4297.644	13	0.4	-186
683	Burlington	Technology	Machines	Cubify CubeX 3D Printer Triple Head Print	7999.980	4	0.5	-383
7772	Lancaster	Technology	Machines	Cubify CubeX 3D Printer Double Head Print	4499.985	5	0.7	-659

We can definitely see some huge losses incurred based on giving higher discounts even if the

sales amount was high which is interesting as well as concerning.

Q 4.2: Multivariate Anomaly Detection with Isolation Forest

Here you will detect anomalies using the Isolation Forest model and use the learnings from 4.1. Here you will use the `pyod` version of Isolation Forest which is basically a wrapper over the `scikit-learn` version but with more functionalities.

Initialize and Train Model

Your Turn: Initialize the isolation forest model with similar hyperparameters as before and also assuming the proportion of outliers to be 1% of the total data (using the contamination setting)

In [37]:
```
from pyod.models import iforest

if_model = iforest.IForest(n_estimators=100, contamination=0.01)
if_model.fit(subset_df)
```

Out[37]:
```
IForest(behaviour='old', bootstrap=False, contamination=0.01,
    max_features=1.0, max_samples='auto', n_estimators=100, n_jobs=1,
    random_state=None, verbose=0)
```

Filter and Sort Outliers

Your Turn: Predict outliers in our dataset using our trained model and filter out the outlier observations and sort by descending order and view the top 5 outlier values similar to 4.1

In [38]:
```
outlier_predictions = if_model.predict(subset_df)

outliers_df = subset_df.copy(deep=True)
outliers_df['Outlier'] = outlier_predictions
outliers_df = outliers_df[outliers_df['Outlier'] == 1]

print('Total Outliers:', len(outliers_df))
outliers_sorted = outliers_df.sort_values(by=['Profit', 'Discount'], ascending=F
outliers_sorted.head(5)
```

```
Total Outliers: 99
```

Out[38]:

	Discount	Profit	Outlier
6826	0.0	1.000000	1
8153	0.0	0.888000	1
4190	0.0	0.776000	1
9039	0.0	0.769759	1
4098	0.0	0.748699	1

View Bottom 10 Outlier Transactions

Your turn: View the bottom ten transactions

In [39]:
```
(df.loc[outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Category', 'Pr
                                         'Sales', 'Quantity', 'Discount', '
```

Out[39]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	
2697	Jacksonville	Technology	Machines	Cisco TelePresence System EX90 Videoconferenci...	22638.480	6	0.5	-1811
1199	Houston	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	1088.792	4	0.8	-1850
9639	Concord	Furniture	Tables	Chromcraft Bull-Nose Wood Oval Conference Tabl...	4297.644	13	0.4	-1862
5310	Houston	Office Supplies	Binders	Fellowes PB500 Electric Punch Plastic Comb Bin...	1525.188	6	0.8	-2287
3151	Newark	Technology	Machines	Cubify CubeX 3D Printer Double Head Print	1799.994	2	0.7	-2639
4991	Chicago	Office Supplies	Binders	Ibico EPK-21 Electric Binding System	1889.990	5	0.8	-2929
3011	Louisville	Technology	Machines	Lexmark MX611dhe Monochrome Laser Printer	2549.985	5	0.7	-3399
9774	San Antonio	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	2177.584	8	0.8	-3701
683	Burlington	Technology	Machines	Cubify CubeX 3D Printer Triple Head Print	7999.980	4	0.5	-3839
7772	Lancaster	Technology	Machines	Cubify CubeX 3D Printer Double Head Print	4499.985	5	0.7	-6599

Q: Do you notice any differences in the results with the previous model?

We do notice some transactions with 80% discount and high losses

Q 4.3: Multivariate Anomaly Detection with Auto-encoders

Here you will detect anomalies using the Auto-encoder model and use the learnings from 4.1. Here you will use the Auto-encoder model from `pyod` which is a deep learning model often used for learning useful data representations in an unsupervised fashion without any labeled data.

Similar to PCA, AE could be used to detect outlier objects in the data by calculating the reconstruction errors

Initialize Model

Here we initiaze an auto-encoder network with a few hidden layers so that we could train it for a 100 epochs

In [40]:
```
from pyod.models import auto_encoder

ae_model = auto_encoder.AutoEncoder(hidden_neurons=[2, 32, 32, 2],
                                    hidden_activation='relu',
                                    output_activation='sigmoid',
                                    epochs=100,
                                    batch_size=32,
                                    contamination=0.01)
```

Train Model

Your turn: Train the model by calling the `fit()` function on the right data

In [41]:
```
ae_model.fit(subset_df)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2)                 6
_________________________________________________________________
dropout (Dropout)            (None, 2)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 6
_________________________________________________________________
dropout_1 (Dropout)          (None, 2)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 6
_________________________________________________________________
dropout_2 (Dropout)          (None, 2)                 0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                96
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                1056
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 66
_________________________________________________________________
```

```
dropout_5 (Dropout)          (None, 2)                 0
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 6
=================================================================
Total params: 1,242
Trainable params: 1,242
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/100
282/282 [==============================] - 1s 2ms/step - loss: 1.3511 - val_los
s: 0.8831
Epoch 2/100
282/282 [==============================] - 0s 936us/step - loss: 1.2661 - val_lo
ss: 0.8190
Epoch 3/100
282/282 [==============================] - 0s 940us/step - loss: 1.2107 - val_lo
ss: 0.7732
Epoch 4/100
282/282 [==============================] - 0s 961us/step - loss: 1.1707 - val_lo
ss: 0.7398
Epoch 5/100
282/282 [==============================] - 0s 1ms/step - loss: 1.1412 - val_los
s: 0.7152
Epoch 6/100
282/282 [==============================] - 0s 1ms/step - loss: 1.1194 - val_los
s: 0.6969
Epoch 7/100
282/282 [==============================] - 0s 1ms/step - loss: 1.1029 - val_los
s: 0.6829
Epoch 8/100
282/282 [==============================] - 0s 955us/step - loss: 1.0905 - val_lo
ss: 0.6723
Epoch 9/100
282/282 [==============================] - 0s 939us/step - loss: 1.0810 - val_lo
ss: 0.6641
Epoch 10/100
282/282 [==============================] - 0s 932us/step - loss: 1.0737 - val_lo
ss: 0.6576
Epoch 11/100
282/282 [==============================] - 0s 960us/step - loss: 1.0680 - val_lo
ss: 0.6525
Epoch 12/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0635 - val_los
s: 0.6483
Epoch 13/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0599 - val_los
s: 0.6450
Epoch 14/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0570 - val_los
s: 0.6421
Epoch 15/100
282/282 [==============================] - 0s 1000us/step - loss: 1.0546 - val_l
oss: 0.6397
Epoch 16/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0527 - val_los
s: 0.6377
Epoch 17/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0512 - val_los
s: 0.6360
Epoch 18/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0499 - val_los
s: 0.6346
Epoch 19/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0489 - val_los
```

```
s: 0.6334
Epoch 20/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0480 - val_los
s: 0.6323
Epoch 21/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0472 - val_los
s: 0.6313
Epoch 22/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0466 - val_los
s: 0.6305
Epoch 23/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0460 - val_los
s: 0.6298
Epoch 24/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0456 - val_los
s: 0.6291
Epoch 25/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0452 - val_los
s: 0.6286
Epoch 26/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0448 - val_los
s: 0.6281
Epoch 27/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0445 - val_los
s: 0.6276
Epoch 28/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0442 - val_los
s: 0.6273
Epoch 29/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0440 - val_los
s: 0.6269
Epoch 30/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0438 - val_los
s: 0.6266
Epoch 31/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0437 - val_los
s: 0.6263
Epoch 32/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0435 - val_los
s: 0.6261
Epoch 33/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0433 - val_los
s: 0.6259
Epoch 34/100
282/282 [==============================] - 0s 995us/step - loss: 1.0432 - val_lo
ss: 0.6257
Epoch 35/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0431 - val_los
s: 0.6255
Epoch 36/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0430 - val_los
s: 0.6253
Epoch 37/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0429 - val_los
s: 0.6251
Epoch 38/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0429 - val_los
s: 0.6250
Epoch 39/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0428 - val_los
s: 0.6249
Epoch 40/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0427 - val_los
s: 0.6248
Epoch 41/100
```

```
282/282 [==============================] - 0s 1000us/step - loss: 1.0427 - val_l
oss: 0.6246
Epoch 42/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0426 - val_los
s: 0.6245
Epoch 43/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0426 - val_los
s: 0.6244
Epoch 44/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0425 - val_los
s: 0.6243
Epoch 45/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0425 - val_los
s: 0.6243
Epoch 46/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0425 - val_los
s: 0.6242
Epoch 47/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0424 - val_los
s: 0.6241
Epoch 48/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0424 - val_los
s: 0.6241
Epoch 49/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0424 - val_los
s: 0.6240
Epoch 50/100
282/282 [==============================] - 0s 940us/step - loss: 1.0423 - val_lo
ss: 0.6239
Epoch 51/100
282/282 [==============================] - 0s 946us/step - loss: 1.0423 - val_lo
ss: 0.6239
Epoch 52/100
282/282 [==============================] - 0s 948us/step - loss: 1.0423 - val_lo
ss: 0.6238
Epoch 53/100
282/282 [==============================] - 0s 966us/step - loss: 1.0423 - val_lo
ss: 0.6238
Epoch 54/100
282/282 [==============================] - 0s 981us/step - loss: 1.0423 - val_lo
ss: 0.6237
Epoch 55/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0422 - val_los
s: 0.6237
Epoch 56/100
282/282 [==============================] - 0s 966us/step - loss: 1.0422 - val_lo
ss: 0.6237
Epoch 57/100
282/282 [==============================] - 0s 955us/step - loss: 1.0422 - val_lo
ss: 0.6236
Epoch 58/100
282/282 [==============================] - 0s 945us/step - loss: 1.0422 - val_lo
ss: 0.6236
Epoch 59/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0422 - val_los
s: 0.6236
Epoch 60/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0422 - val_los
s: 0.6235
Epoch 61/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0422 - val_los
s: 0.6235
Epoch 62/100
282/282 [==============================] - 0s 992us/step - loss: 1.0422 - val_lo
ss: 0.6235
```

```
Epoch 63/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0422 - val_los
s: 0.6234
Epoch 64/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6234
Epoch 65/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6234
Epoch 66/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6234
Epoch 67/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6233
Epoch 68/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6233
Epoch 69/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6233
Epoch 70/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6233
Epoch 71/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6233
Epoch 72/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6232
Epoch 73/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6232
Epoch 74/100
282/282 [==============================] - 0s 983us/step - loss: 1.0421 - val_lo
ss: 0.6232
Epoch 75/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6232
Epoch 76/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6232
Epoch 77/100
282/282 [==============================] - 0s 968us/step - loss: 1.0421 - val_lo
ss: 0.6232
Epoch 78/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6231
Epoch 79/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6231
Epoch 80/100
282/282 [==============================] - 0s 976us/step - loss: 1.0421 - val_lo
ss: 0.6231
Epoch 81/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0421 - val_los
s: 0.6231
Epoch 82/100
282/282 [==============================] - 0s 954us/step - loss: 1.0421 - val_lo
ss: 0.6231
Epoch 83/100
282/282 [==============================] - 0s 980us/step - loss: 1.0421 - val_lo
ss: 0.6231
Epoch 84/100
282/282 [==============================] - 0s 979us/step - loss: 1.0420 - val_lo
```

```
ss: 0.6231
Epoch 85/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6231
Epoch 86/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6231
Epoch 87/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 88/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 89/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 90/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 91/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 92/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 93/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 94/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 95/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 96/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 97/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 98/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 99/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6230
Epoch 100/100
282/282 [==============================] - 0s 1ms/step - loss: 1.0420 - val_los
s: 0.6229
```

Out[41]:
```
AutoEncoder(batch_size=32, contamination=0.01, dropout_rate=0.2, epochs=100,
      hidden_activation='relu', hidden_neurons=[2, 32, 32, 2],
      l2_regularizer=0.1,
      loss=<function mean_squared_error at 0x7f86ed6c5dc0>,
      optimizer='adam', output_activation='sigmoid', preprocessing=True,
      random_state=None, validation_size=0.1, verbose=1)
```

Filter and Sort Outliers

Your Turn: Predict outliers in our dataset using our trained model and filter out the outlier observations and sort by descending order and view the top 5 outlier values similar to 4.1

In [42]:
```
outlier_predictions = ae_model.predict(subset_df)
```

```
outliers_df = subset_df.copy(deep=True)
outliers_df['Outlier'] = outlier_predictions
outliers_df = outliers_df[outliers_df['Outlier'] == 1]

print('Total Outliers:', len(outliers_df))
outliers_sorted = outliers_df.sort_values(by=['Profit', 'Discount'], ascending=F
outliers_sorted.head(5)
```

```
Total Outliers: 100
```

Out[42]:

	Discount	Profit	Outlier
6826	0.0	1.000000	1
8153	0.0	0.888000	1
4190	0.0	0.776000	1
9039	0.0	0.769759	1
4098	0.0	0.748699	1

View Bottom 10 Outlier Transactions

Your turn: View the bottom ten transactions

In [43]:

```
(df.loc[outliers_sorted.index.tolist()][['City', 'Category', 'Sub-Category', 'Pr
                                         'Sales', 'Quantity', 'Discount', '
```

Out[43]:

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	
2697	Jacksonville	Technology	Machines	Cisco TelePresence System EX90 Videoconferenci...	22638.480	6	0.5	-1811
1199	Houston	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	1088.792	4	0.8	-1850
9639	Concord	Furniture	Tables	Chromcraft Bull-Nose Wood Oval Conference Tabl...	4297.644	13	0.4	-1862
5310	Houston	Office Supplies	Binders	Fellowes PB500 Electric Punch Plastic Comb Bin...	1525.188	6	0.8	-2287
3151	Newark	Technology	Machines	Cubify CubeX 3D Printer Double Head Print	1799.994	2	0.7	-2639
4991	Chicago	Office Supplies	Binders	Ibico EPK-21 Electric Binding System	1889.990	5	0.8	-2929
3011	Louisville	Technology	Machines	Lexmark MX611dhe Monochrome Laser Printer	2549.985	5	0.7	-3399

	City	Category	Sub-Category	Product Name	Sales	Quantity	Discount	
9774	San Antonio	Office Supplies	Binders	GBC DocuBind P400 Electric Binding System	2177.584	8	0.8	-3701
683	Burlington	Technology	Machines	Cubify CubeX 3D Printer Triple Head Print	7999.980	4	0.5	-3839
7772	Lancaster	Technology	Machines	Cubify CubeX 3D Printer Double Head Print	4499.985	5	0.7	-6599

4.4: Visualize Anomalies and Compare Anomaly Detection Models

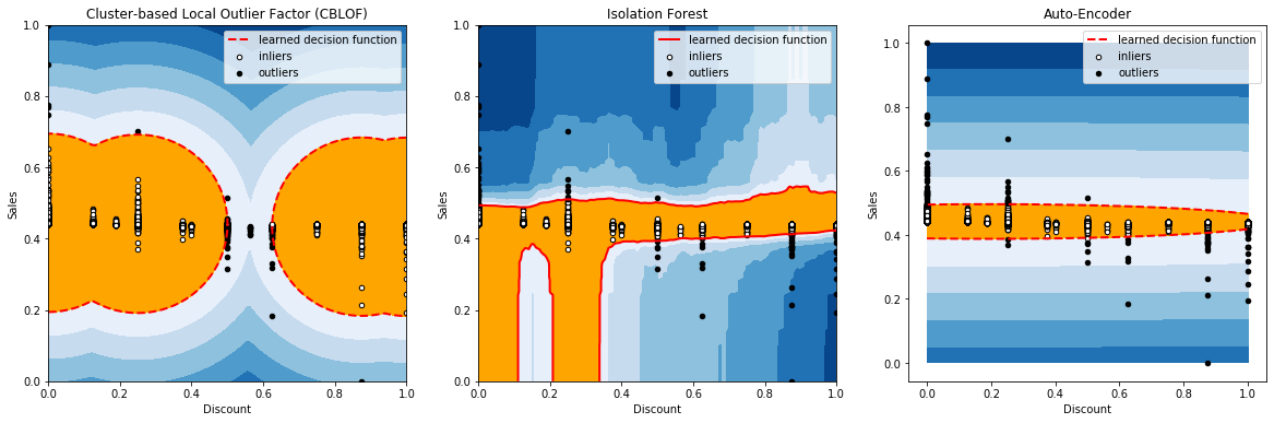
Here we will look at the visual plots of anomalies as detected by the above three models

In [44]:
```
def visualize_anomalies(model, xx, yy, data_df, ax_obj, subplot_title):

    # predict raw anomaly score
    scores_pred = model.decision_function(data_df) * -1
    # prediction of a datapoint category outlier or inlier
    y_pred = model.predict(data_df)
    n_inliers = len(y_pred) - np.count_nonzero(y_pred)
    n_outliers = np.count_nonzero(y_pred == 1)


    out_df = data_df.copy(deep=True)
    out_df['Outlier'] = y_pred.tolist()
    # discount - inlier feature 1,  profit - inlier feature 2
    inliers_discount = out_df[out_df['Outlier'] == 0]['Discount'].values
    inliers_profit = out_df[out_df['Outlier'] == 0]['Profit'].values
    # discount - outlier feature 1, profit - outlier feature 2
    outliers_discount = out_df[out_df['Outlier'] == 1]['Discount'].values
    outliers_profit = out_df[out_df['Outlier'] == 1]['Profit'].values

    # Use threshold value to consider a datapoint inlier or outlier
    # threshold = stats.scoreatpercentile(scores_pred,100 * outliers_fraction)
    threshold = np.percentile(scores_pred, 100 * outliers_fraction)
    # decision function calculates the raw anomaly score for every point
    Z = model.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
    Z = Z.reshape(xx.shape)
    # fill blue map colormap from minimum anomaly score to threshold value
    ax_obj.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),cmap=pl
    # draw red contour line where anomaly score is equal to thresold
    a = ax_obj.contour(xx, yy, Z, levels=[threshold],linewidths=2, colors='red')
    # fill orange contour lines where range of anomaly score is from threshold t
    ax_obj.contourf(xx, yy, Z, levels=[threshold, Z.max()],colors='orange')
    b = ax_obj.scatter(inliers_discount, inliers_profit, c='white',s=20, edgecol
    c = ax_obj.scatter(outliers_discount, outliers_profit, c='black',s=20, edgec


    ax_obj.legend([a.collections[0], b,c], ['learned decision function', 'inlier
                  prop=matplotlib.font_manager.FontProperties(size=10),loc='upper r
```

```
    ax_obj.set_xlim((0, 1))
    ax_obj.set_ylim((0, 1))
    ax_obj.set_xlabel('Discount')
    ax_obj.set_ylabel('Sales')
    ax_obj.set_title(subplot_title)
```

In [45]:

```
outliers_fraction = 0.01
xx , yy = np.meshgrid(np.linspace(0, 1, 100), np.linspace(0, 1, 100))
fig, ax = plt.subplots(1, 3, figsize=(20, 6))

ax_objs = [ax[0], ax[1], ax[2]]
models = [cblof_model, if_model, ae_model]
plot_titles = ['Cluster-based Local Outlier Factor (CBLOF)',
               'Isolation Forest',
               'Auto-Encoder']

for ax_obj, model, plot_title in zip(ax_objs, models, plot_titles):
    visualize_anomalies(model=model,
                        xx=xx, yy=yy,
                        data_df=subset_df,
                        ax_obj=ax_obj,
                        subplot_title=plot_title)
plt.axis('tight');
```



In [ ]: