CHAPTER 8

Customer Segmentation and Effective Cross Selling

Money makes the world go round and in the current ecosystem of data intensive business practices, it is safe to claim that data also makes the world go round. A very important skill set for data scientists is to match the technical aspects of analytics with its business value, i.e., its monetary value. This can be done in a variety of ways and is very much dependent on the type of business and the data available. In the earlier chapters, we covered problems that can be framed as business problems (leveraging the CRISP-DM model) and linked to revenue generation. In this chapter we will directly focus on two very important problems that can directly have a positive impact on the revenue streams of businesses and establishments particularly from the retail domain. This chapter is also unique in the way that we address a different paradigm of Machine Learning algorithm altogether, focusing more on tasks pertaining to pattern recognition and unsupervised learning.

In this chapter, first we digress from our usual technical focus and try to gather some business and domain knowledge. This knowledge is quite important, as often this is the stumbling block for many data scientists in scenarios where a perfectly developed Machine Learning solution is not productionalized due to a lack of focus in the actual value obtained from the solution based on business demands. A firm grasp on the underlying business (and monetary) motivation helps the data scientists with defining the value aspect of their solutions and hence ensuring that they are deployed and contribute to generation of realizable value for their employers.

For achieving this objective, we will start with a retail transactions based dataset sourced from UCI Machine Learning Repository (`http://archive.ics.uci.edu/ml/datasets/online+retail`) and we will use this dataset to target two fairly simple but important problems. The detailed code, notebooks, and datasets used in this chapter are available in the respective directory for Chapter 8 in the GitHub repository at `https://github.com/dipanjanS/practical-machine-learning-with-python`.

- **Customer segmentation**: Customer segmentation is the problem of uncovering information about a firm's customer base, based on their interactions with the business. In most cases this interaction is in terms of their purchase behavior and patterns. We explore some of the ways in which this can be used.
- **Market basket analysis**: Market basket analysis is a method to gain insights into granular behavior of customers. This is helpful in devising strategies which uncovers deeper understanding of purchase decisions taken by the customers. This is interesting as a lot of times even the customer will be unaware of such biases or trends in their purchasing behavior.

D. Sarkar et al., *Practical Machine Learning with Python*, https://doi.org/10.1007/978-1-4842-3207-1_8

Online Retail Transactions Dataset

The online retail transactions dataset is available from the UCI Machine Learning Repository. We already used some datasets from this repository in our earlier chapters and this should underline the importance of this repository to the users. The dataset we will be using for our analysis is quite simple. Based on its description on the UCI web site, it contains all the transactions occurring between `01/12/2010` and `09/12/2011` for a UK-based and registered non-store online retail. From the web site, we also learn that the company sells unique all-occasion gift items and a lot of customers of the organization are wholesalers.

The last piece of information is particularly important as gives us an opportunity to explore purchase behaviors of large-scale customers instead of normal retail customers only. The dataset does not have any information that will help us distinguish between a wholesale purchase and a retail purchase. Before we get started, make sure you load the following dependencies.

```
import pandas as pd
import datetime
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab

%matplotlib inline
```

■ **Note** We encourage you to check out the UCI Machine Learning Repository and the page for this particular dataset at `http://archive.ics.uci.edu/ml/datasets/online+retail`. On the web site, you can find some research papers that use the same dataset. We believe the papers along with the analysis performed in this chapter will make an interesting read for all our readers.

Exploratory Data Analysis

We have always maintained that irrespective of the actual use case or the algorithm, we intend to implement the standard analysis workflow, which should always start with exploratory data analysis (EDA). So following the tradition, we will start with EDA on our dataset.

The first thing you should notice about the dataset is its format. Unlike most of the datasets that we have handled in this book the dataset is not in a CSV format and instead comes as an Excel file. In some other languages (or even frameworks) it could have been a cause of problem but with python and particularly `pandas` we don't face any such problem and we can read the dataset using the function `read_excel` provided by the `pandas` library. We also take a look at some of the lines in the dataset.

```
In [3]: cs_df = pd.read_excel(io=r'Online Retail.xlsx')
```

The few lines from the dataset gives us information about the attributes of the datasets, as shown in Figure 8-1.

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

Figure 8-1. *Sample transactions from the Retail Transactions dataset*

The attributes of the dataset are easily identifiable from their names. We know right away what each of these fields might mean. For the sake of completeness, we include the description of each column here:

- **InvoiceNo**: A unique identifier for the invoice. An invoice number shared across rows means that those transactions were performed in a single invoice (multiple purchases).
- **StockCode**: Identifier for items contained in an invoice.
- **Description**: Textual description of each of the stock item.
- **Quantity**: The quantity of the item purchased.
- **InvoiceDate**: Date of purchase.
- **UnitPrice**: Value of each item.
- **CustomerID**: Identifier for customer making the purchase.
- **Country**: Country of customer.

Let's analyze this data and first determine which are the top countries the retailer is shipping its items to, and how are the volumes of sales for those countries.

```
In [5]: cs_df.Country.value_counts().reset_index().head(n=10)
Out[5]:
            index  Country
0  United Kingdom   495478
1         Germany     9495
2          France     8557
3            EIRE     8196
4           Spain     2533
5     Netherlands     2371
6         Belgium     2069
7     Switzerland     2002
8        Portugal     1519
9       Australia     1259
```

This shows us that the bulk of ordering is taking place in its home country only which is not surprising. We also notice the odd country name `EIRE`, which is a little concerning. But a quick web search indicates that it is just an old name for Ireland, so no harm done! Interestingly, Australia is also in the top-ten list of sales by country.

Next we might be interested in how many unique customers the retailer is having and how do they stack up in the number of orders they make. We are also interested in knowing that what percentage of orders is made by the top 10 customers of the retailer. This information is interesting as it would tell us whether the user base of the firm is distributed relatively uniformly.

```
In [7]: cs_df.CustomerID.unique().shape
Out[7]: (4373,)

In [8]: (cs_df.CustomerID.value_counts()/sum(cs_df.CustomerID.value_counts())*100).
head(n=13).cumsum()
Out[8]:
17841.0    1.962249
14911.0    3.413228
14096.0    4.673708
12748.0    5.814728
14606.0    6.498553
15311.0    7.110850
14646.0    7.623350
13089.0    8.079807
13263.0    8.492020
14298.0    8.895138
15039.0    9.265809
14156.0    9.614850
18118.0    9.930462
Name: CustomerID, dtype: float64
```

This tells us that we have 4,373 unique customers but almost 10% of total sales are contributed by only 13 customers (based on the cumulative percentage aggregation in the preceding output). This is expected given the fact that we have both wholesale and retail customers. The next thing we want to determine is how many unique items the firm is selling and check whether we have equal number of descriptions for them.

```
In [9]: cs_df.StockCode.unique().shape
Out[9]: (4070,)

In [10]: cs_df.Description.unique().shape
Out[10]: (4224,)
```

We have a mismatch in the number of `StockCode` and `Description`, as we can see that item descriptions are more than stock code values, which means that we have multiple descriptions for some of the stock codes. Although this is not going to interfere with our analysis, we would like to dig a little deeper in this to find out what may have caused this issue or what kind of duplicated descriptions are present in the data.

```
cat_des_df = cs_df.groupby(["StockCode","Description"]).count().reset_index()
cat_des_df.StockCode.value_counts()[cat_des_df.StockCode.value_counts()>1].reset_index().head()
```

```
   index  StockCode
0  20713          8
1  23084          7
2  85175          6
3  21830          6
4  21181          5
```

```
In [13]: cs_df[cs_df['StockCode']
    ...:       ==cat_des_df.StockCode.value_counts()[cat_des_df.StockCode.value_counts()>1].
    ...:       reset_index()['index'][6]]['Description'].unique()
Out[13]:
array(['MISTLETOE HEART WREATH CREAM', 'MISTLETOE HEART WREATH WHITE',
       'MISTLETOE HEART WREATH CREAM', '?', 'had been put aside', nan], dtype=object)
```

This gives the multiple descriptions for one of those items and we witness the simple ways in which data quality can be corrupted in any dataset. A simple spelling mistake can end up in reducing data quality and an erroneous analysis. In an enterprise-level scenario, dedicated people work toward restoring data quality manually over time. Since the intent of this section is to focus on customer segmentation, we will be skipping this tedious activity for now. Let's now verify the sanctity of the `Quantity` and `UnitPrice` attributes, as those are the attributes we will be using in our analysis.

```
In [14]: cs_df.Quantity.describe()
Out[14]:
count    541909.000000
mean          9.552250
std         218.081158
min      -80995.000000
25%           1.000000
50%           3.000000
75%          10.000000
max       80995.000000
Name: Quantity, dtype: float64

In [15]: cs_df.UnitPrice.describe()
Out[15]:
count    541909.000000
mean          4.611114
std          96.759853
min      -11062.060000
25%           1.250000
50%           2.080000
75%           4.130000
max       38970.000000
Name: UnitPrice, dtype: float64
```

We can observe from the preceding output that both of these attributes are having negative values, which may mean that we may have some return transactions in our data also. This scenario is quite common for any retailer but we need to handle these before we proceed to our analysis. These are some of the data quality issues we found in our dataset. In the real world, the datasets are generally messy and have considerable data quality issues, so it is always a good practice to explicitly verify information at hand before performing any kind of analysis. We encourage you to try and find similar issues with any dataset which you might want to analyze in the future.

Customer Segmentation

Segmentation is the process of segregating any aggregated entity into separate parts or groups (segments). These parts may or may not share something common across them. Customer segmentation is similarly the process of dividing an organization's customer bases into different sections or segments based on various customer attributes. It is driven by the belief that customers are inherently different and this difference is exemplified by their behavior. A deep understanding of an organization's customer base and their behavior is often the focus of any customer segmentation project. The process of customer segmentation is based on the premise of finding differences among the customers' behavior and patterns. These differences can be on the basis of their buying behavior, their demographic information, their geographic information, their psychographic attributes, and so on.

Objectives

Customer segmentation can help an organization in multitude of ways. Before we describe the various ways it can be done, we want to enumerate the major objectives and benefits behind the motivation for customer segmentation.

Customer Understanding

One of the primary objectives of a customer segmentation process is a deeper understanding of a firm's customers and their attributes and behavior. These insights into the customer base can be used in different ways, as we will discuss shortly. But the information is useful by itself. One of the mostly widely accepted business paradigms is "know your customer" and a segmentation of the customer base allows for a perfect dissection of this paradigm. This understanding and its exploitation is what forms the basis of the other benefits of customer segmentation.

Target Marketing

The most visible reason for customer segmentation is the ability to focus marketing efforts effectively and efficiently. If a firm knows the different segments of its customer base, it can devise better marketing campaigns which are tailor made for the segment. Consider the example of any travel company, if it knows that the major segments of its customers are the budget travelers and the luxury travelers, it can have two separate marketing campaigns for each of the group. One can focus on the higher value aspects of the company's offerings relevant to budget deals while the other campaign deals with luxurious offerings. Although the example seems quite trivial, the same logic can be extended in a number of ways to arrive at better marketing practices. A good segmentation model allows for better understanding of customer requirements and hence increases the chances of the success of any marketing campaign developed by the organization.

Optimal Product Placement

A good customer segmentation strategy can also help the firm with developing or offering new products. This benefit is highly dependent on the way the segmentation process is leveraged. Consider a very simple example in which an online retailer finds out that a major section of its customers are buying makeup products together. This may prompt him to bundle those product together as a combined offering, which may increase sales margin for the retailer and make the buying process more streamlined for the customer.

Finding Latent Customer Segments

A customer segmentation process helps the organization with knowing its customer base. An obvious side effect of any such practice is finding out which segment of customers it might be missing. This can help in identifying untapped customer segments by focused on marketing campaigns or new product development.

Higher Revenue

This is the most obvious requirement of any customer segmentation project. The reason being that customer segmentation can lead to higher revenue due to the combined effects of all the advantages identified in this section.

Strategies

The easy answer to the question of "How to do customer segmentation?" would be "In any way you deem fit" and it would be a perfectly acceptable answer. The reason this is the right answer, is because of the original definition of customer segmentation. It is just a way of differentiating between the customers. It can be as simple as making groups of customers based on age groups or other attributes using a manual process or as complex as using a sophisticated algorithm for finding out those segments in an automated fashion. Since our book is all about Machine Learning, we describe how customer segmentation can be translated into a core Machine Learning task. The detailed code for this section is available in the notebook titled `Customer Segmentation.ipynb`.

Clustering

We are dealing with an unlabeled, unsupervised transactional dataset from which we want to find out customer segments. Thus, the most obvious method to perform customer segmentation is using unsupervised Machine Learning methods like clustering. Hence, this will be the method that we use for customer segmentation in this chapter. The method is as simple as collecting as much data about the customers as possible in the form of *features* or *attributes* and then finding out the different clusters that can be obtained from that data. Finally, we can find traits of customer segments by analyzing the characteristics of the clusters.

Exploratory Data Analysis

Using exploratory data analysis is another way of finding out customer segments. This is usually done by analysts who have a good knowledge about the domain relevant to both products and customers. It can be done flexibly to include the top decision points in an analysis. For example, finding out the range of spends by customers will give rise to customer segments based on spends. We can proceed likewise on important attributes of customers until we get segments of customer that have interesting characteristics.

Clustering vs. Customer Segmentation

In our use case, we will be using a clustering based model to find out interesting segments of customers. Before we go on to modify our data for the model there is an interesting point we would like to clarify. A lot of people think that clustering is equivalent to customer segmentation. Although it is true that clustering is one of the most suitable techniques for segmentation, it is not the only technique. Besides this, it is just a method that is "applied" to extract *segments*.

Customer segmentation is just the task of segmenting customers. It can be solved in several ways and it need not always be a complex model. Clustering provides a mathematical framework which can be leveraged for finding out such segment boundaries in the data. Clustering is especially useful when we have a lot of attributes about the customer on which we can make different segments. Also, often it is observed that a clustering-based segmentation will be superior to an arbitrary segmentation process and it will often encompass the segments that can be devised using such an arbitrary process.

Clustering Strategy

Now that we have some information about what customer segmentation is, various strategies and how it can be useful, we can start with the process of finding out customer segments in our online retail dataset. The dataset that we have consists of only the sales transactions of the customers and no other information about them, i.e., no other additional attributes. Usually in larger organizations we will usually have more attributes of information about the customers that can help in clustering. However, it will be interesting and definitely a challenge to work with this limited attribute dataset! So we will use a **RFM—R**ecency, **F**requency and **M**onetary Value—based model of customer value for finding our customer segments.

RFM Model for Customer Value

The RFM model is a popular model in marketing and customer segmentation for determining a customer's value. The RFM model will take the transactions of a customer and calculate three important informational attributes about each customer:

- **Recency**: The value of how recently a customer purchased at the establishment
- **Frequency**: How frequent the customer's transactions are at the establishment
- **Monetary value**: The dollar (or pounds in our case) value of all the transactions that the customer made at the establishment

A combination of these three values can be used to assign a value to the customer. We can directly think of some desirable segments that we would want on such model. For example, a high value customer is one who buys frequently, just bought something recently, and spends a high amount whenever he buys or shops!

Data Cleaning

We hinted in the "Exploratory Data Analysis" section about the return transactions that we have in our dataset. Before proceeding with our analysis workflow, we will find out all such transactions and remove them. Another possibility is to remove the matching buying transactions also from the dataset. But we will assume that those transactions are still important hence we will keep them intact. Another data cleaning operation is to separate transactions for a particular geographical region only, as we don't want data from Germany's customers to affect the analysis for another country's customers. The following snippet of code achieves both these tasks. We focus on UK customers, which are notably the largest segment (based on country!).

```
In [16]: # Separate data for one geography
    ...: cs_df = cs_df[cs_df.Country == 'United Kingdom']
    ...:
    ...: # Separate attribute for total amount
    ...: cs_df['amount'] = cs_df.Quantity*cs_df.UnitPrice
    ...:
    ...: # Remove negative or return transactions
```

```
    ...: cs_df = cs_df[~(cs_df.amount<0)]
    ...: cs_df.head()
    ...:
Out[16]:
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country  amount
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom   15.30
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom   20.34
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom   22.00
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom   20.34
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom   20.34
```

The data now has only buying transactions from United Kingdom. We will now remove all the transactions that have a missing value for the `CustomerID` field as all our subsequent transactions will be based on the customer entities.

```
In [17]: cs_df = cs_df[~(cs_df.CustomerID.isnull())]
In [18]: cs_df.shape
Out[18]: (354345, 9)
```

The next step is creating the recency, frequency, and monetary value features for each of the customers that exist in our dataset.

Recency

To create the recency feature variable, we need to decide the reference date for our analysis. For our use case, we will define the reference date as one day after the last transaction in our dataset.

```
In [19]: refrence_date = cs_df.InvoiceDate.max()
    ...: refrence_date = refrence_date + datetime.timedelta(days = 1)
    ...: refrence_date
Out[20]: Timestamp('2011-12-10 12:49:00')
```

We will construct the recency variable as the number of days before the reference date when a customer last made a purchase. The following snippet of code will create this variable for us.

```
In [21]: cs_df['days_since_last_purchase'] = refrence_date - cs_df.InvoiceDate
    ...: cs_df['days_since_last_purchase_num'] = cs_df['days_since_last_purchase'].
astype('timedelta64[D]')

In [22]: customer_history_df = cs_df.groupby("CustomerID").min().reset_index()
[['CustomerID', 'days_since_last_purchase_num']]
    ...: customer_history_df.rename(columns={'days_since_last_purchase_num':'recency'},
inplace=True)
```

Before we proceed, let's examine how the distribution of customer recency looks for our data (see Figure 8-2).

```
x = customer_history_df.recency
mu = np.mean(customer_history_df.recency)
sigma = math.sqrt(np.var(customer_history_df.recency))
n, bins, patches = plt.hist(x, 1000, facecolor='green', alpha=0.75)

# add a 'best fit' line
y = mlab.normpdf( bins, mu, sigma)
l = plt.plot(bins, y, 'r--', linewidth=2)

plt.xlabel('Recency in days')
plt.ylabel('Number of transactions')
plt.title(r'$\mathrm{Histogram\ of\ sales\ recency}\ $')
plt.grid(True)
```

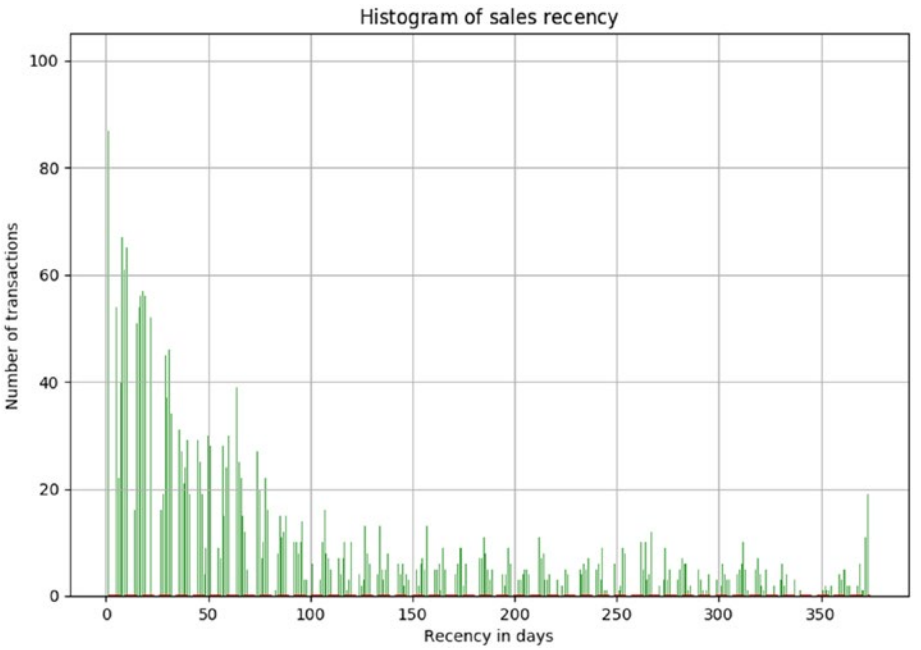

Figure 8-2. *Distribution of sales recency*

The histogram in Figure 8-2 tells us that we have skewed distribution of sales recency with a much higher number of frequent transactions and a fairly uniform number of less recent transactions.

Frequency and Monetary Value

Using similar methods, we can create a frequency and monetary value variable for our dataset. We will create these variables separately and then merge all the dataframes to arrive at the customer value dataset. We will perform our clustering-based customer segmentation on this dataframe. The following snippet will create both these variables and the final merged dataframe.

```
In [29]: customer_monetary_val = cs_df[['CustomerID',
                                        'amount']].groupby("CustomerID").sum().reset_index()
    ...: customer_history_df = customer_history_df.merge(customer_monetary_val, how='outer')
    ...: customer_history_df.amount = customer_history_df.amount+0.001
    ...:
    ...: customer_freq = cs_df[['CustomerID',
                                'amount']].groupby("CustomerID").count().reset_index()
    ...: customer_freq.rename(columns={'amount':'frequency'},inplace=True)
    ...: customer_history_df = customer_history_df.merge(customer_freq, how='outer')
```

The input dataframe for clustering will look like the dataframe depicted in Figure 8-3. Notice that we have added a small figure `0.001` to the customer monetary value, as we will be transforming our values to the log scale and presence of zeroes in our data may lead to an error.

	CustomerID	recency	amount	frequency
0	12346.0	326.0	77183.601	1
1	12747.0	2.0	4196.011	103
2	12748.0	1.0	33719.731	4596
3	12749.0	4.0	4090.881	199
4	12820.0	3.0	942.341	59

Figure 8-3. *Customer value dataframe*

Data Preprocessing

Once we have created our customer value dataframe, we will perform some preprocessing on the data. For our clustering, we will be using the *K-means clustering* algorithm, which we discussed in the earlier chapters. One of the requirements for proper functioning of the algorithm is the mean centering of the variable values. Mean centering of a variable value means that we will replace the actual value of the variable with a standardized value, so that the variable has a mean of 1 and variance of 0. This ensures that all the variables are in the same range and the difference in ranges of values doesn't cause the algorithm to not perform well. This is akin to feature scaling.

Another problem that you can investigate about is the huge range of values each variable can take. This problem is particularly noticeable for the monetary amount variable. To take care of this problem, we will transform all the variables on the log scale. This transformation, along with the standardization, will ensure that the input to our algorithm is a homogenous set of scaled and transformed values.

An important point about the data preprocessing step is that sometimes we need it to be reversible. In our case, we will have the clustering results in terms of the log transformed and scaled variable. But to make inferences in terms of the original data, we will need to reverse transform all the variable so that we get back the actual RFM figures. This can be done by using the preprocessing capabilities of Python.

```
In [30]: from sklearn import preprocessing
    ...: import math
    ...: customer_history_df['recency_log'] = customer_history_df['recency'].apply(math.log)
    ...: customer_history_df['frequency_log'] = customer_history_df['frequency'].apply(math.log)
    ...: customer_history_df['amount_log'] = customer_history_df['amount'].apply(math.log)
    ...: feature_vector = ['amount_log', 'recency_log','frequency_log']
```

```
...: X = customer_history_df[feature_vector].as_matrix()
...: scaler = preprocessing.StandardScaler().fit(X)
...: X_scaled = scaler.transform(X)
```

The previous code snippet will create a log valued and mean centered version of our dataset. We can visualize the results of our preprocessing by inspecting the variable with the widest range of values. The following code snippet will help us visualize this.

```
In [31]: x = customer_history_df.amount_log
    ...: n, bins, patches = plt.hist(x, 1000, facecolor='green', alpha=0.75)
    ...: plt.xlabel('Log of Sales Amount')
    ...: plt.ylabel('Probability')
    ...: plt.title(r'$\mathrm{Histogram\ of\ Log\ transformed\ Customer\ Monetary\ value}\ $')
    ...: plt.grid(True)
    ...: plt.show()
```

The resulting graph is a distribution resembling normal distribution with mean 0 and variance of 1, as clearly depicted in Figure 8-4.

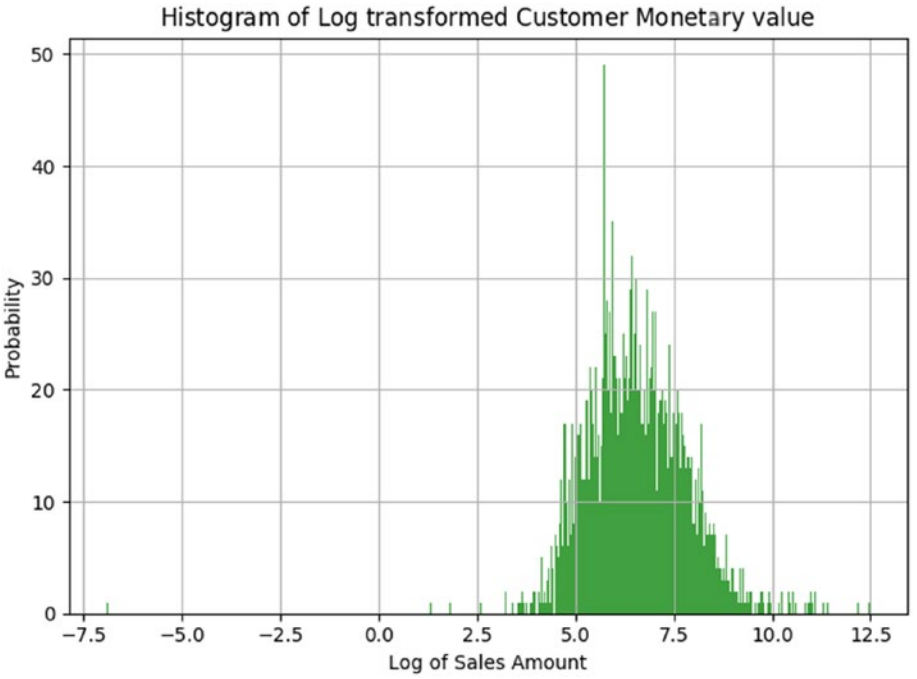

Figure 8-4. *Scaled and log transformed sales amount*

Let's try to visualize our three main features (R, F, and M) on a three-dimensional plot to see if we can understand any interesting patterns that the data distribution is showing.

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

xs =customer_history_df.recency_log
ys = customer_history_df.frequency_log
zs = customer_history_df.amount_log
ax.scatter(xs, ys, zs, s=5)

ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')
```

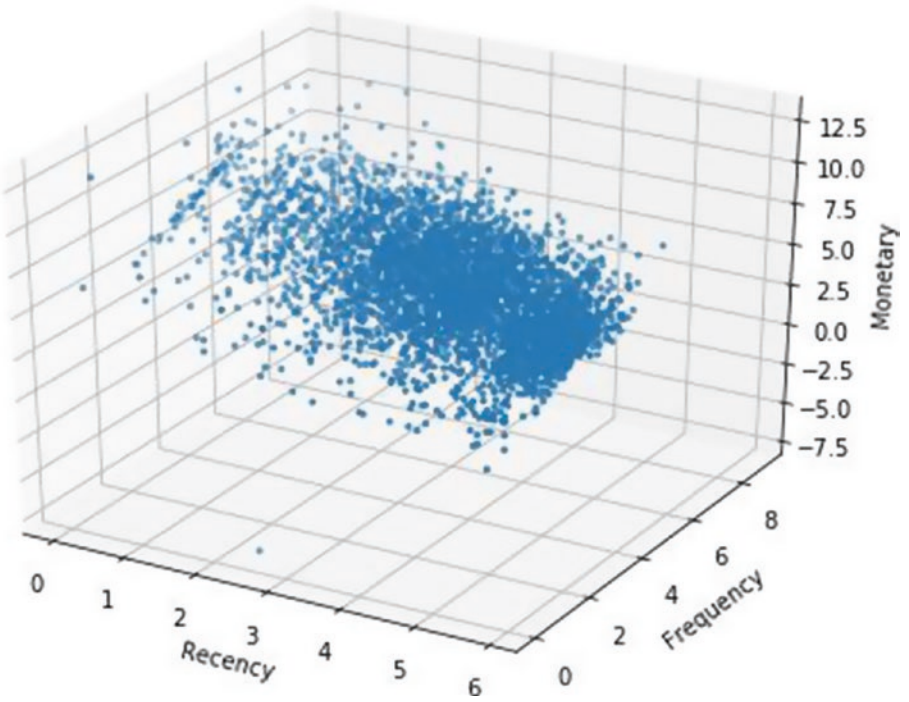

Figure 8-5. *Customer value dataframe*

The obvious patterns we can see from the plot in Figure 8-5 is that people who buy with a higher frequency and more recency tend to spend more based on the increasing trend in Monetary value with a corresponding increasing and decreasing trend for Frequency and Recency, respectively. Do you notice any other interesting patterns?

Clustering for Segments

We will be using the K-means clustering algorithm for finding out clusters (or segments in our data). It is one of the simplest clustering algorithms that we can employ and hence it is widely used in practice. We will give you a brief primer of the algorithm before we go on to using the same, for finding segments in our data.

K-Means Clustering

The K-means clustering belongs to the partition based\centroid based clustering family of algorithms. The steps that happen in the K-means algorithm for partitioning the data are as given follows:

1. The algorithm starts with random point initializations of the required number of centers. The "K" in K-means stands for the number of clusters.
2. In the next step, each of the data point is assigned to the center closest to it. The distance metric used in K-means clustering is normal Euclidian distance.
3. Once the data points are assigned, the centers are recalculated by averaging the dimensions of the points belonging to the cluster.
4. The process is repeated with new centers until we reach a point where the assignments become stable. In this case, the algorithm terminates.

We will adapt the code given in the `scikit-learn` documentation at `http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html` and use the silhouette score for finding out the optimal number of clusters during our clustering process. We leave it as an exercise for you to adapt the code and create the visualization in Figure 8-6. You are encouraged to modify the code in the documentation to not only build the visualization, but also to capture the centers and the silhouette score of each cluster in a dictionary, as we will need to refer to those for performing our analysis of the customer segments obtained. Of course, in case you find it overwhelming, you can always refer to the detailed code snippet in the `Customer Segmentation.ipynb` notebook.

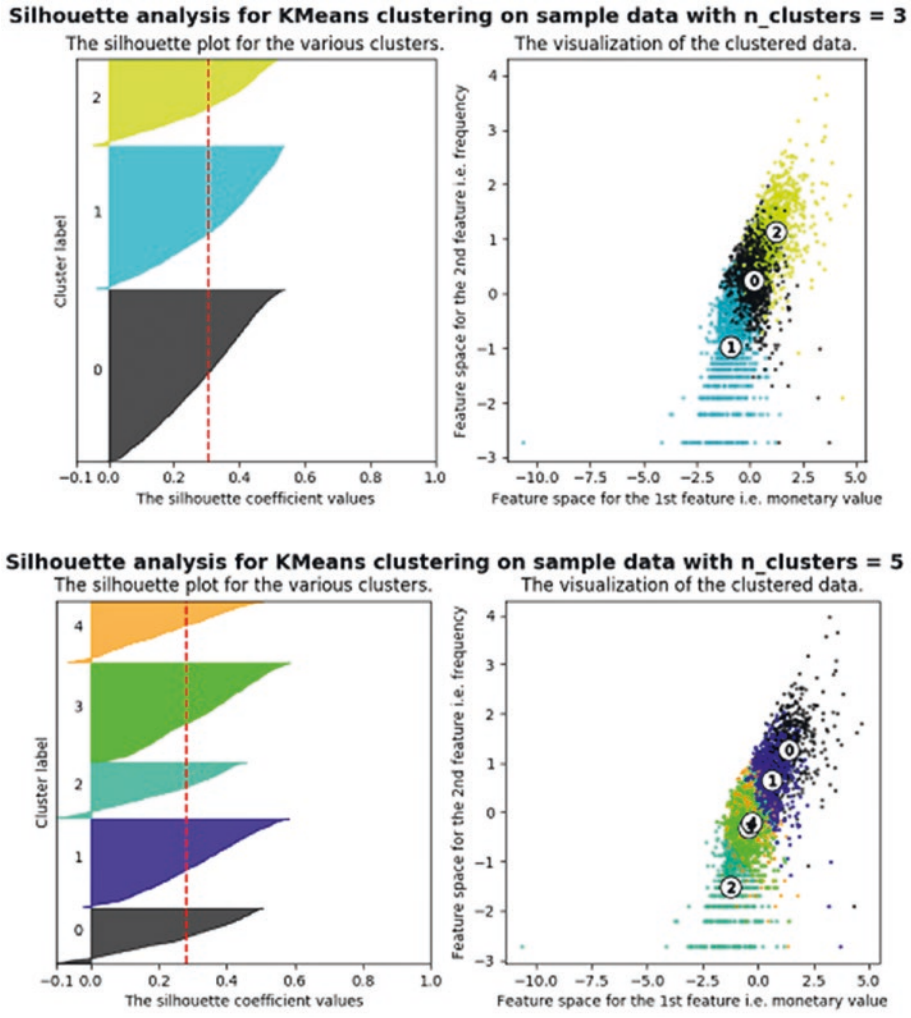

Figure 8-6. *Silhouette analysis with three and five clusters*

In the visualization depicted in Figure 8-6, we plotted the silhouette score of each cluster along with the center of each of the cluster discovered. We will use this information in the next section on cluster analysis. Although we have to keep in mind that in several cases and scenarios, sometimes we may have to drop the mathematical explanation given by the algorithm and look at the business relevance of the results obtained.

Cluster Analysis

Before we proceed to the analysis of clusters such obtained, let's look at the cluster center values after retransforming them to normal values from the log and scaled version. The following code helps us convert the center values to the reversed transformed values.

```
In [38]: for i in range(3,6,2):
    ...:     print("for {} number of clusters".format(i))
```

```
    ...:     cent_transformed = scaler.inverse_transform(cluster_centers[i]['cluster_center'])
    ...:     print(pd.DataFrame(np.exp(cent_transformed),columns=feature_vector))
    ...:     print("Silhouette score for cluster {} is {}". format(i,
                                                  cluster_centers[i]['silhouette_score']))
    ...:     print()

for 3 number of clusters
    amount_log  recency_log  frequency_log
0   843.937271    44.083222      53.920633
1   221.236034   121.766072      10.668661
2  3159.294272     7.196647     177.789098
Silhouette score for cluster 3 is 0.30437444714898737

for 5 number of clusters
    amount_log  recency_log  frequency_log
0  3905.544371     5.627973     214.465989
1  1502.519606    46.880212      92.306262
2   142.867249   126.546751       5.147370
3   408.235418   139.056216      25.530424
4   464.371885    13.386419      29.581298
Silhouette score for cluster 5 is 0.27958641427323727
```

When we look at the results of the clustering process, we can infer some interesting insights. Consider the three-cluster configuration and try to understand the following insights.

- We get three clusters with stark differences in the Monetary value of the customer.
- Cluster 2 is the cluster of high value customer who shops frequently and is certainly an important segment for each business.
- In the similar way we obtain customer groups with low and medium spends in clusters with labels 1 and 0, respectively.
- Frequency and Recency correlate perfectly to the Monetary value based on the trend we talked about in Figure 8-3 (High Monetary-Low Recency-High Frequency).

The five-cluster configuration results are more surprising! When we go looking for more segments, we find out that our high valued customer base is comprised of two subgroups:

- Those who shop often and with high amount (represented by cluster 0).
- Those who have a decent spend but are not as frequent (represented by cluster 1).

This is in direct conflict with the result we obtain from the silhouette score matrix which says the five-cluster segments are less optimal then the three cluster segments. Of course, remember you must not strictly go after mathematical metrics all the time and think about the business aspects too. Besides this, there could be more insights that are uncovered as you visualize the data based on these segments which might prove that in-fact the three-cluster segmentation was far better. For instance, if you check the right-side plot in Figure 8-6, you can see that segments with five clusters have too much overlap among them, as compared to segments with three clusters.

Cluster Descriptions

On the basis of eyeballing the cluster centers, we can figure out that we have a good difference in the customer value in the segments as defined in terms of recency, amount and frequency. To further drill down on this point and find out the quality of these difference, we can label our data with the corresponding cluster label and then visualize these differences. We will do this visualization for probably one of the most important customer value indicators, the total dollar value sales.

To arrive at such distinction based summary computations, we will first label each data row in our customer summary dataframe with the corresponding label as returned by our clustering algorithm. Note that you have to modify the code you are using if you want to try a different configuration of let's say two or four clusters. We will have to make changes so that we capture the labels for each different cluster configuration. We encourage you to try other cluster configurations to see if you get even better segments! The following code will extract the clustering label and attach it with our customer summary dataframe.

```
labels = cluster_centers[5]['labels']
customer_history_df['num_cluster5_labels'] = labels
labels = cluster_centers[3]['labels']
customer_history_df['num_cluster3_labels'] = labels
```

Once we have the labels assigned to each of the customers, our task is simple. Now we want to find out how the summary of customer in each group is varying. If we can visualize that information we will able to find out the differences in the clusters of customers and we can modify our strategy on the basis of those differences. We have used a lot of `matplotlib` and `seaborn` so far; we will be using `plotly` in this section for creating some interactive plots that you can play around with in your jupyter notebook!

■ **Note** While `plotly` provides excellent interactive visualizations in jupyter notebooks, you might come across some notebook rendering issues that come up as popups when you open the notebook. To fix this problem, upgrade your `nbformat` library by running the `conda update nbformat` command and re-open the notebook. The problem should disappear.

The following code leverages `plotly` and will take the cluster labels we got for the configuration of five clusters and create boxplots that will show how the median, minimum, maximum, highest, and lowest values are varying in the five groups. Note that we want to avoid the extremely high outlier values of each group, as they will interfere in making a good observation (due to noise) around the central tendencies of each cluster. So we will restrict the data such that only data points which are less than 0.8th percentile of the cluster is used. This will give us good information about the majority of the users in that cluster segment. The following code will help us create this plot for the total sales value.

```
import plotly as py
import plotly.graph_objs as go
py.offline.init_notebook_mode()

x_data = ['Cluster 1','Cluster 2','Cluster 3','Cluster 4', 'Cluster 5']
cutoff_quantile = 80
field_to_plot = 'amount'

y0=customer_history_df[customer_history_df['num_cluster5_labels']==0][field_to_plot].values
y0 = y0[y0<np.percentile(y0, cutoff_quantile)]
y1=customer_history_df[customer_history_df['num_cluster5_labels']==1][field_to_plot].values
```

```
y1 = y1[y1<np.percentile(y1, cutoff_quantile)]
y2=customer_history_df[customer_history_df['num_cluster5_labels']==2][field_to_plot].values
y2 = y2[y2<np.percentile(y2, cutoff_quantile)]
y3=customer_history_df[customer_history_df['num_cluster5_labels']==3][field_to_plot].values
y3 = y3[y3<np.percentile(y3, cutoff_quantile)]
y4=customer_history_df[customer_history_df['num_cluster5_labels']==4][field_to_plot].values
y4 = y4[y4<np.percentile(y4, cutoff_quantile)]
y_data = [y0,y1,y2,y3,y4]

colors = ['rgba(93, 164, 214, 0.5)', 'rgba(255, 144, 14, 0.5)', 'rgba(44, 160, 101, 0.5)',
'rgba(255, 65, 54, 0.5)', 'rgba(207, 114, 255, 0.5)', 'rgba(127, 96, 0, 0.5)']
traces = []

for xd, yd, cls in zip(x_data, y_data, colors):
        traces.append(go.Box(
            y=yd,      name=xd,     boxpoints=False,
            jitter=0.5,      whiskerwidth=0.2,
            fillcolor=cls,       marker=dict(size=2,),
            line=dict(width=1), ))

layout = go.Layout(
    title='Difference in sales {} from cluster to cluster'.format(field_to_plot),
    yaxis=dict(
        autorange=True,        showgrid=True,
        zeroline=True,        dtick=1000,
        gridcolor='black',        gridwidth=0.1,
        zerolinecolor='rgb(255, 255, 255)',
        zerolinewidth=2,    ),
    margin=dict(l=40,r=30, b=80, t=100,    ),
    paper_bgcolor='white',    plot_bgcolor='white',  showlegend=False
)

fig = go.Figure(data=traces, layout=layout)
py.offline.iplot(fig)
```

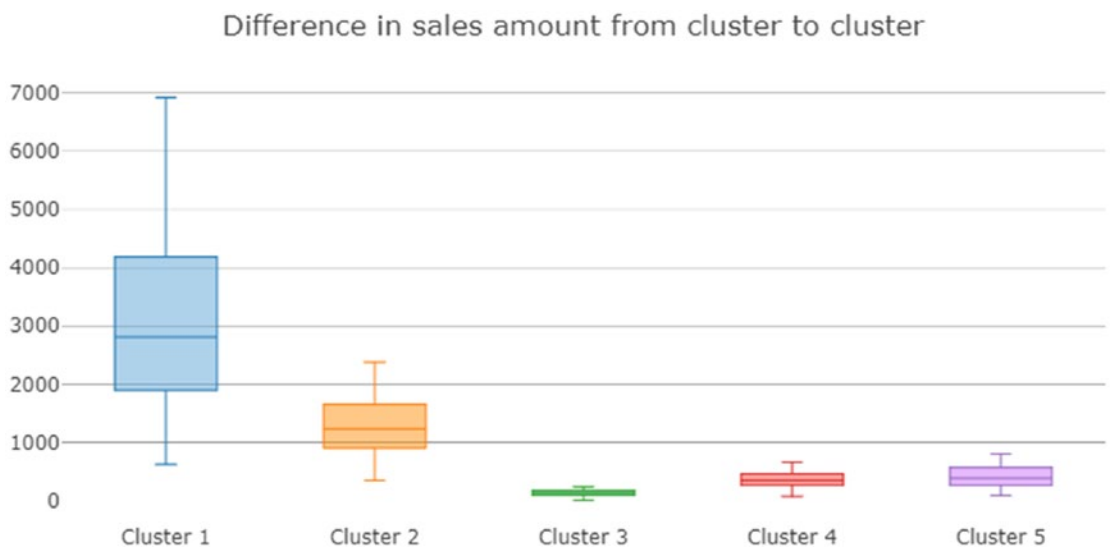

Figure 8-7. *Difference in sales amount values across the five segments*

Let's also take a look at the plot that is generated as a result of this code snippet. In Figure 8-7 we can see that the Clusters 1 and 2 have a higher average sales amount, thus being the highest spenders. Although we don't see much difference in the sales values of Clusters 4 and 5, we do see a markedly smaller sales amount in Cluster 3. This gives us an indication that we can merge the candidates of Clusters 4 and 5 together, at least on the basis of sales amount. You can plot similar figures for recency and frequency also to figure out differences in the cluster on the basis of those values. Detailed code and visualizations are present in the notebook for three- and five-cluster based segments. Plotly enables us to interact with the plots to see the central tendency values in each boxplot in the notebook. We show the difference in sales amount across three segments based on the three-cluster configuration just so you can compare it with Figure 8-7.

The detailed visualization is shown in Figure 8-8, which talks about difference in the sales\revenue across the three segments. It is clear that this is much more distinguishable and we also have lesser overlap as compared to Figure 8-7 where Clusters 4 and 5 were very similar. This doesn't mean that the previous method was wrong; it's just one of the dimensions where some segments were similar to each other.

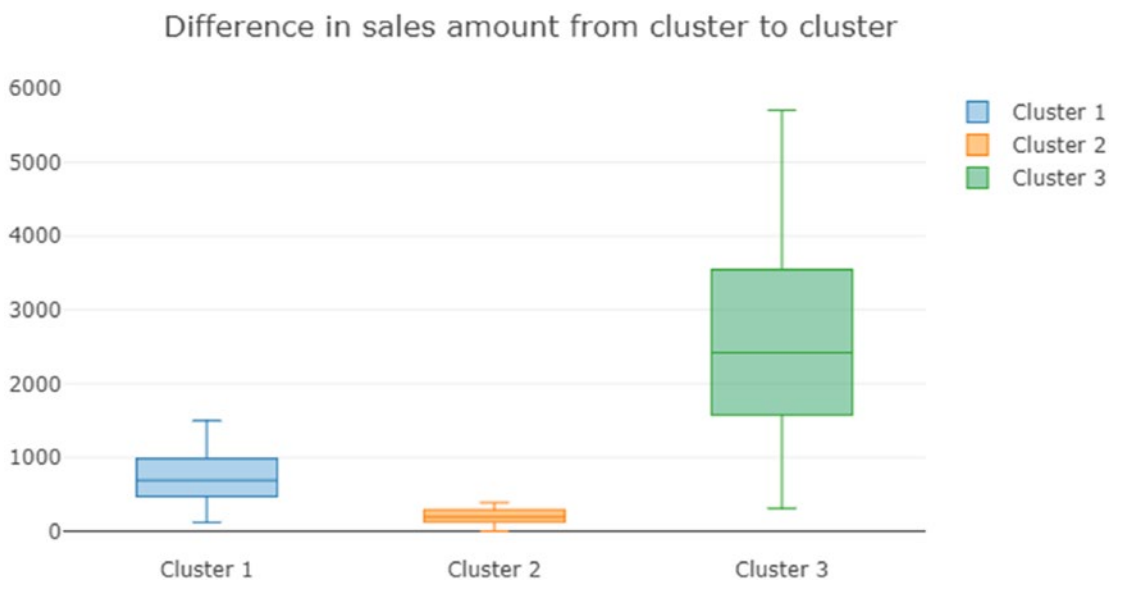

Figure 8-8. *Difference in sales amount values across the three segments*

We can further improve the quality of clustering by adding relevant features to the dataset that we have created. Often firms will buy data regarding their customers from external data vendors and use it to enhance the segmentation process. We had a limitation of only having around a year worth of transaction data but even a mid-size organization may have multiple years of transaction data which can improve the results. Another dimension to explore can be trying out different algorithms for performing the segmentation for instance hierarchical clustering, which we explored in some of the earlier chapters. A good segmentation process will encompass all these avenues to arrive at optimal segments that provide valuable insight. We encourage you to get creative with the process and build your own examples.

The important caveat from this analysis is that when it comes to correlating the actual value of results with the mathematical metrics we cannot always rely on the metrics. We need to include this habit of including business metrics and domain insights in our modeling process as often times this becomes the difference between an implemented high value project and a forgotten data-focused solution. Once we obtain these results, we can further discuss them with the marketing team of the organization to come up with appropriate practices for each of the segment identified.

Cross Selling

What is cross selling? In the simplest terms, cross selling is the ability to sell more products to a customer by analyzing the customer's shopping trends as well as general shopping trends and patterns which are in common with the customer's shopping patterns. This simple definition captures the essential idea of cross selling, but it is not as descriptive as we would like it to be.

We will illustrate the idea with an example. Say we are concerned about our health (which everyone should be) and decide to buy a protein supplement on our favorite e-commerce site. In most scenarios, the moment you get to your product page, you will have a section that will tell you other products that you can buy along with the product(s) of your choice. See Figure 8-9.

Figure 8-9. *Cross selling example*

More often than not, these recommended products would be very appealing. For example, if I am in the market for a protein supplement it will definitely be a good idea for me to buy a vitamin supplement too. The retailer will often offer you a bundle of products with some attractive offer and it is highly likely that we will end up buying the bundled products instead of just the original item. This is the simple but powerful concept of cross selling. We research the customer transactions and find out potential additions to the customer's original needs and offer it to the customer as a suggestion in the hope and intent that they buy them benefiting both the customer as well as the retail establishment.

Cross selling is ubiquitous in both the online and offline retail worlds. The simplicity and the effectiveness of the idea make it an essential and powerful marketing tool for all types of retailers. The idea of cross selling can be extended to any organization, irrespective of whether it is an online or offline retailer or whether it is selling its products to the end users of whole sellers.

In this section, we explore *association rule-mining,* a powerful technique that can be used for cross selling, then we illustrate the concept of market basket analysis for a toy dataset, and finally we apply the same concepts to our retail transactions dataset.

Market Basket Analysis with Association Rule-Mining

Before we go on to understand how to generate the association rules from our transactional data, let's try to understand how we will use those rules. A famous story in data analytics circles is the story of "Beer and Diapers". The basic crux of the story is the concept that a major retailer upon analyzing their customer transaction behavior discovered a strong association between sales of beer and diapers. The retailer was able to exploit this association by moving the beer section close to the diapers section leading to higher sales volume (The origins of the story may have some fact but the whole concept is debatable. We encourage you to go through the discussion at `http://www.dssresources.com/newsletters/66.php` for learning more about this story).

Although the beer and diaper story may or may not have been a myth, the concept of finding sales association in customer behavior is an important and inspiring one. Suppose we can mathematically capture the significance of these associations, then it would be a good idea to try and exploit the rules which are likely to be correct. The whole concept of association rule-mining is based on the concept that customer purchase behavior has a pattern which can be exploited for selling more items to the customer in the future. An association rule usually has the structure as depicted in this equation:

$$\{item_1,\ item_2, item_3 \rightarrow item_k\}$$

This rule can be read in the obvious manner that when the customer bought items on the (left hand side) LHS of the rule he is likely to buy the $item_k$. In the later sections, we define mathematical metrics that will capture the strength of such rules.

We can use these rules in a variety of ways. The most obvious way to use these rules is to develop bundles of products which make it convenient for the customer to buy these items together. Another way to use these rules is to bundle products along with some discounts for other relevant products in the bundle, hence ensuring that the customer becomes more likely to buy more items. Some unlikely ways to use association rules is for designing a better web site navigational structure, intrusion detection, bioinformatics, and so on.

Association Rule-Mining Basics

Before proceeding to explore the association rules in our dataset, we will go through some essential concepts for association rule-mining. These terms and concepts will help you in the later analysis and also in understanding the rules that the algorithm will generate. Consider Table 8-1 with some toy transaction data.

Table 8-1. *Example of a Transaction Set*

Trans.ID	Items
1	{milk, bread}
2	{butter}
3	{beer, diaper}
4	{milk, bread, butter}
5	{bread}

Each row in the table consists of a transaction. For example, the customer bought milk and bread in the first transaction. Following are some vital concepts pertaining to association rule-mining.

- **Itemset**: Itemset is just a collection of one or more items that occur together in a transaction. For example, here {milk, bread} is example of an itemset.
- **Support**: Support is defined as number of times an itemset appears in the dataset. Mathematically it is defined as:

$$supp\left(\{beer, diaper\}\right) = \frac{\text{number of transactions with beer and diaper}}{total\ \ transactions}$$

  In the previous example, support (beer, diaper) = 1/5 = 0.2.
- **Confidence**: Confidence is a measure of the times the number of times a rule is found to exist in the dataset. For a rule which states $\{beer \rightarrow diaper\}$ the confidence is mathematically definedas:

$$confidence\left(\{beer, diaper\}\right) = \frac{supp\left(beer\ and\ diaper\right)}{supp\left(beer\right)}$$

- **Lift**: Lift of the rule is defined as the ratio of observed support to the support expected in the case the elements of the rule were independent. For the previous set of transactions if the rule is defined as $\{X \rightarrow Y\}$, then the lift of the rule is defined as:

$$lift(X \rightarrow Y) = \frac{supp(X \cup Y)}{(supp(X) * supp(Y))}$$

- **Frequent itemset**: Frequent itemsets are itemsets whose support is greater than a user defined support threshold.

FP Growth

The most famous algorithm for association rule-mining is the Apriori algorithm, for which you will find a lot of code and resources on the web and in standard data mining literature. However, here we will use a different and more efficient algorithm, the FP growth algorithm for finding our association rules. The major bottleneck in any association rule-mining algorithm is the generation of frequent itemsets. If the transaction dataset is having k unique products, then potentially we have 2^k possible itemsets. The Apriori algorithm will first generate these itemsets and then proceed to finding the frequent itemsets.

This limitation is a huge performance bottleneck as even for around 100 unique products the possible number of itemsets is huge. This limitation makes the Apriori algorithm prohibitively computationally expensive. The FP growth algorithm is superior to Apriori algorithm as it doesn't need to generate all the candidate itemsets. The algorithm uses a special data structure that helps it retains itemset association information. An example of the data structure is depicted in Figure 8-10.

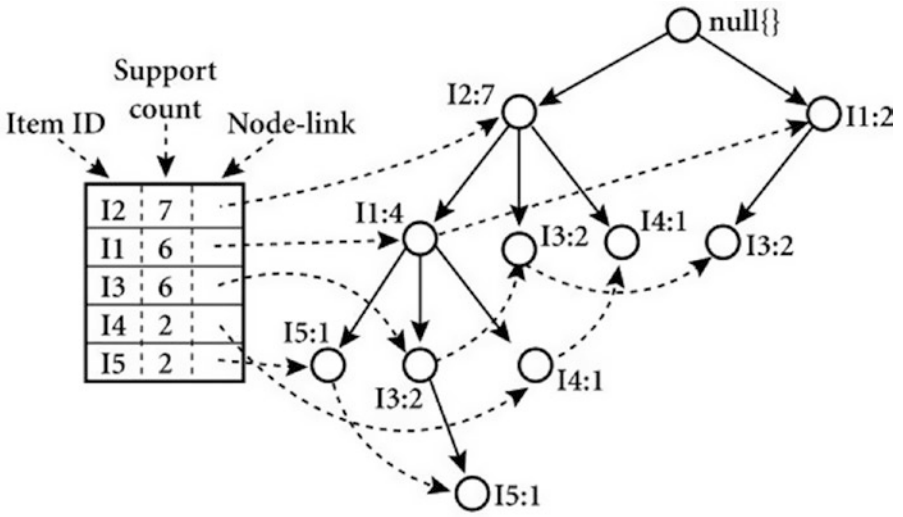

Figure 8-10. *An example of an FP-tree*

We will not go into detailed mathematical descriptions of the algorithm here, as the intent is to not to keep this section math heavy but to focus on how it can be leveraged to find patterns in this data. However, we will explain it in brief so you can understand the core concepts in this method. The FP growth algorithm uses a divide-and-conquer strategy and leverages a special data structure called the FP-tree, as depicted in Figure 8-10, to find frequent itemsets without generating all itemsets. The core steps of the algorithm are as follows:

1. Take in the transactional database and create an FP-tree structure to represent frequent itemsets.
2. Divide this compressed representation into multiple conditional datasets such that each one is associated with a frequent pattern.
3. Mine for patterns in each such dataset so that shorter patterns can be recursively concatenated to longer patterns, hence making it more efficient.

If you are interested in finding about it, you can refer to the Wikibooks link at https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm, which talks about FP growth and the FP tree structure in detail.

Association Rule-Mining in Action

We will illustrate association rule-mining using the famous grocery dataset. The dataset is available by default in the R language's base package. To use in in Python, you can obtain it from https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/groceries.csv or even from our official GitHub repository mentioned at the start of this chapter. This dataset consists of a collection of transactions that are sourced from a grocery retailer. We will use this data as the basis of our analysis and build our rule-mining work flow using this data.

Once we have grasped the basics of association rule-mining on the grocery dataset, we will leave it as an exercise for you to apply the same concepts on our transaction dataset that we used in the customer segmentation section. Remember to load the following dependencies before getting started.

```
import csv
import pandas as pd
import matplotlib.pyplot as plt
import Orange
from Orange.data import Domain, DiscreteVariable, ContinuousVariable
from orangecontrib.associate.fpgrowth import *

%matplotlib inline
```

Check out the "Mining Rules" section for details on how to install the `Orange` framework dependencies. The code for this section is available in the `Cross Selling.ipynb` notebook.

Exploratory Data Analysis

The grocery dataset that we mentioned earlier is arranged so that each of the line occurring in the dataset is a transaction. The items given in each row are comma-separated and are the items in that particular transaction. Take a look at the first few lines of the dataset depicted in Figure 8-11.

```
citrus fruit,semi-finished bread,margarine,ready soups
tropical fruit,yogurt,coffee
whole milk
pip fruit,yogurt,cream cheese ,meat spreads
other vegetables,whole milk,condensed milk,long life bakery product
whole milk,butter,yogurt,rice,abrasive cleaner
rolls/buns
other vegetables,UHT-milk,rolls/buns,bottled beer,liquor (appetizer)
pot plants
whole milk,cereals
tropical fruit,other vegetables,white bread,bottled water,chocolate
citrus fruit,tropical fruit,whole milk,butter,curd,yogurt,flour,bottled water,dishes
```

Figure 8-11. *Grocery dataset transactions*

The first observation that we make from this dataset is that it is not available in a completely structured, easy-to-analyze format. This limitation will mean that the first thing we will have to do is to write custom code that will convert the raw file into a data structure we can use. Since we have done most of our analysis until now using the `pandas` dataframe, we will convert this data into a similar data structure. The following code snippet will perform the conversion for us.

```
grocery_items = set()
with open("grocery_dataset.txt") as f:
    reader = csv.reader(f, delimiter=",")
    for i, line in enumerate(reader):
        grocery_items.update(line)
output_list = list()
with open("grocery_dataset.txt") as f:
    reader = csv.reader(f, delimiter=",")
    for i, line in enumerate(reader):
        row_val = {item:0 for item in grocery_items}
        row_val.update({item:1 for item in line})
        output_list.append(row_val)

grocery_df = pd.DataFrame(output_list)

In [3]: grocery_df.shape
Out[3]: (9835, 169)
```

The conversion gives us a dataframe of dimension (`num_transaction`, `total_items`), where each transaction row has columns corresponding to its constituent items as 1. For example, for row 3 in Figure 8-11, we will have the column for `whole milk` as 1 and the rest of columns will be all 0. Although this data structure is sparse, meaning it has a lot of zeros, our framework that extracts association rules will take care of this sparseness.

Before we proceed to building association rules on our dataset, we will explore some salient features of our dataset. We already know that we have 9,835 total transactions and a total of 169 items in the dataset. But what are the top 10 items that occur in the dataset and how much of the total sales they account for. We can plot a simple histogram that will help us extract this information.

```
In [4]: total_item_count = sum(grocery_df.sum())
   ...: print(total_item_count)
   ...: item_summary_df = grocery_df.sum().sort_values(ascending = False).reset
   ...: _index().head(n=20)
   ...: item_summary_df.rename(columns={item_summary_df.columns[0]:'item_name',
```

```
   ...: item_summary_df.columns[1]:'item_count'}, inplace=True)
   ...: item_summary_df.head()
43367
Out[4]:
          item_name  item_count
0        whole milk        2513
1  other vegetables        1903
2        rolls/buns        1809
3              soda        1715
4            yogurt        1372
```

For creating the histogram, we will create a summary dataframe using the previous code. This tells us that we have a total of 43,000+ items occurring in total in all those transactions and we also see the top five most sold items. Let's use this dataframe to plot the top 20 most sold items. The following snippet of code will help us create the required bar graph.

```
objects = (list(item_summary_df['item_name'].head(n=20)))
y_pos = np.arange(len(objects))
performance = list(item_summary_df['item_count'].head(n=20))
plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects, rotation='vertical')
plt.ylabel('Item count')
plt.title('Item sales distribution')
```

The bar graph depicting the item sales is depicted in Figure 8-12. It indicates that a surprisingly large share of total items is claimed by only these 20 items.

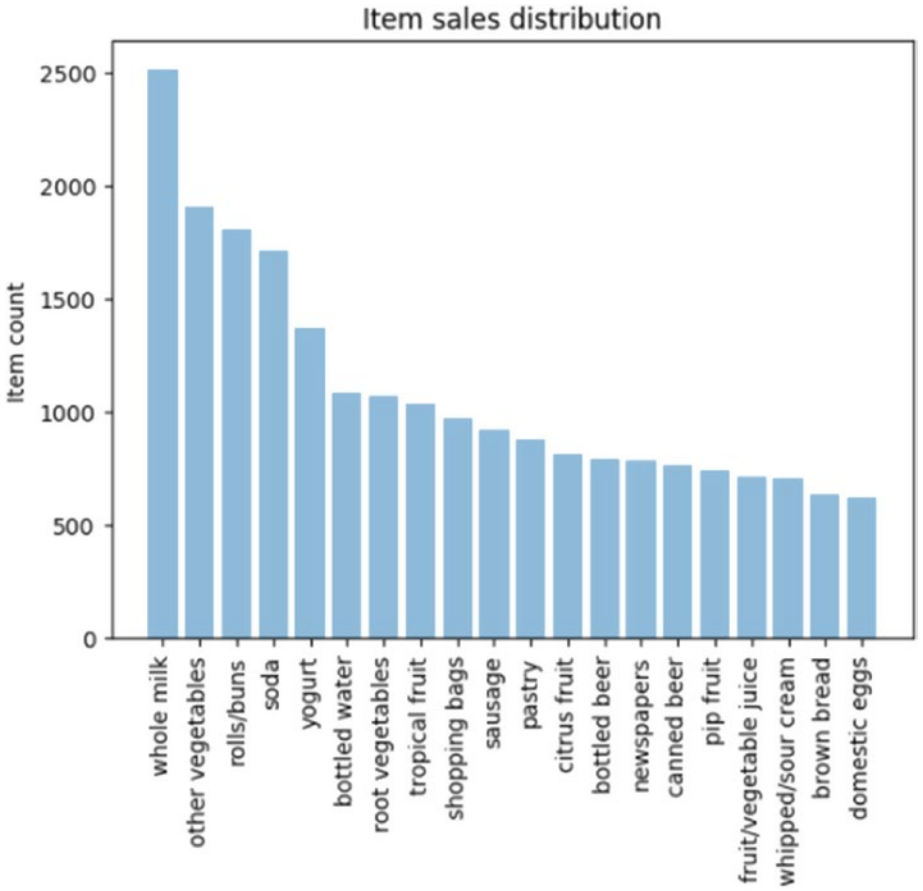

Figure 8-12. *Grocery dataset top 20 items based on sales*

Let's also find out how much percentage of total sales is explained by these 20 items alone. We will use the cumulative sum function offered by pandas (cumsum) to find this out. We will create two columns in our dataframe. One will tell how much percentage of total sales can be attributed to a particular item and the other will keep a cumulative sum of this sales percentage.

```
item_summary_df['item_perc'] = item_summary_df['item_count']/total_item_count
item_summary_df['total_perc'] = item_summary_df.item_perc.cumsum()
item_summary_df.head(10)
```

```
          item_name  item_count  item_perc  total_perc
0        whole milk        2513   0.057947    0.057947
1  other vegetables        1903   0.043881    0.101829
2        rolls/buns        1809   0.041714    0.143542
3              soda        1715   0.039546    0.183089
4            yogurt        1372   0.031637    0.214725

In [12]: item_summary_df[item_summary_df.total_perc <= 0.5].shape
Out[12]: (19, 4)
```

This shows us that the top five items are responsible for 21.4% of the entire sales and only the top 20 items are responsible for over 50% of the sales! This is important for us, as we don't want to find association rules for items which are bought very infrequently. With this information we can limit the items we want to explore for creating our association rules. This also helps us in keeping our possible itemset number to a manageable figure.

Mining Rules

We will be using the `Orange` and the `Orange3-Associate` frameworks, which can be installed using the commands `conda install orange3` and `pip install orange3-associate`. The `orange3-associate` package contains the implementation of FP growth provided by the group who developed for the `Orange` data mining package, The Bioinformatics Laboratory at the University of Ljubljana, Slovenia (**https://fri.uni-lj.si/en/laboratory/biolab**).

■ **Note** We encourage you to experiment with the `Orange` package, which is available at `https://orange.biolab.si/`. It is a GUI-driven data mining framework written in Python and highly conducive for learning data analysis in an interactive way.

Before we go on using the package for finding association rules, we will discuss about the way data is represented in `Orange` library. The data representation is a little tricky but we will help you modify the existing data into the format required by `Orange`. Primarily we will focus on how to convert our `pandas` dataframes to the `Orange Table` data structure.

Orange Table Data Structure

The `Table` data structure is the primary way to represent any tabular data in `Orange`. Although it is similar in some way to a `numpy` array or a `pandas` dataframe, it differs from them in the way it stores metadata about the actual data. In our case we can easily convert our `pandas` dataframe to the `Table` data structure by providing the metadata about our columns. We need to define the domain for each of our variables. The domain means the possible set of values that each of our variables can use. This information will be stored as metadata and will be used in later transformation of the data. As our columns are only having binary values—i.e. either 0 or 1—we can easily create the domain by using this information. The following code snippet helps us convert our dataframe to an `Orange` table.

```
from Orange.data import Domain, DiscreteVariable, ContinuousVariable
from orangecontrib.associate.fpgrowth import *

input_assoc_rules = grocery_df
domain_grocery = Domain([DiscreteVariable.make(name=item, values=['0', '1']) for item in
                                                                    input_assoc_rules.columns])
data_gro_1 = Orange.data.Table.from_numpy(domain=domain_grocery,
                                          X=input_assoc_rules.as_matrix(),Y= None)
```

Here we defined the domain of our data by specifying each variable as a `DiscreteVariable` having values as `(0, 1)`. Then using this domain, we created our `Table` structure for our data.

Using the FP Growth Algorithm

Now we have all the pieces required to perform our rule-mining. But before proceeding, we want to take care of one more important aspect of the analysis. We saw in the earlier section how only a handful of items are responsible for bulk of our sales so we want to prune our dataset to reflect this information. For this we have created a function `prune_dataset` (check out the notebook), which will help us reduce the size of our dataset based on our requirements. The function can be used for performing two types of pruning:

- **Pruning based on percentage of total sales**: The parameter `total_sales_perc` will help us select the number of items that will explain the required percentage of sales. The default value is 50% or 0.5.
- **Pruning based on ranks of items**: Another way to perform the pruning is to specify the starting and the ending rank of the items for which we want to prune our dataset.

By default, we will only look for transactions which have at least two items, as transactions with only one item are counter to the whole concept of association rule-mining. The following code snippet will help us select only that subset of data which explain **40%** of the total sales by leveraging our pruning function.

```
output_df, item_counts = prune_dataset(input_df=grocery_df, length_trans=2,total_sales_perc=0.4)
print(output_df.shape)
print(list(output_df.columns))
```

```
(4585, 13)
['whole milk', 'other vegetables', 'rolls/buns', 'soda', 'yogurt', 'bottled water', 'root
vegetables', 'tropical fruit', 'shopping bags', 'sausage', 'pastry', 'citrus fruit',
'bottled beer']
```

So we find out that we have only 13 items responsible for 40% of sales and 4585 transactions that have those items along with other items and we can also see what those items are. The next step is to convert this selected data into the required `Table` data structure.

```
input_assoc_rules = output_df
domain_grocery = Domain([DiscreteVariable.make(name=item,values=['0', '1']) for item in
                                                                    input_assoc_rules.columns])
data_gro_1 = Orange.data.Table.from_numpy(domain=domain_grocery,
                                          X=input_assoc_rules.as_matrix(), Y= None)
data_gro_1_en, mapping = OneHot.encode(data_gro_1, include_class=False)
```

The new addition to the previous code is the last line. This is required for coding our input so that the entire domain is represented as binary variables. This will complete all the parsing and data manipulation required for our rule-mining. Phew!

The final step is creating our rules. We need to specify two pieces of information for generating our rules: support and confidence. We have already defined both of them conceptually earlier, so we will not be defining them again. An important piece of information is to start with a higher support, as lower support will mean a higher number of frequent itemsets and hence a longer execution time. We will specify a min-support of **0.01**—**45** transactions at least—and see the number of frequent itemsets that we get before we specify confidence and generate our rules.

```
min_support = 0.01
print("num of required transactions = ", int(input_assoc_rules.shape[0]*min_support))
num_trans = input_assoc_rules.shape[0]*min_support
itemsets = dict(frequent_itemsets(data_gro_1_en, min_support=min_support))
```

```
num of required transactions =  45

In [25]: len(itemsets)
Out[25]: 166886
```

So we get a whopping 166,886 itemsets for a support of only 1%! This will increase exponentially if we decrease the support or if we increase the number of items in our dataset. The next step is specifying a confidence value and generating our rules. We have written a code snippet that will take a confidence value and generate the rules that fulfill our specified support and confidence criteria. The rules generated are then decoded using the mapping and variable names. `Orange3-Associate` also provides a helper function that will help us extract metrics about each of these rules. The following code snippet will perform rule generation and decoding of rules, and then compile it all in a neat dataframe that we can use for further analysis.

```
confidence = 0.3
rules_df = pd.DataFrame()

if len(itemsets) < 1000000:
    rules = [(P, Q, supp, conf)
    for P, Q, supp, conf in association_rules(itemsets, confidence)
       if len(Q) == 1 ]

    names = {item: '{}={}'.format(var.name, val)
        for item, var, val in OneHot.decode(mapping, data_gro_1, mapping)}
    eligible_ante = [v for k,v in names.items() if v.endswith("1")]
    N = input_assoc_rules.shape[0] * 0.5
    rule_stats = list(rules_stats(rules, itemsets, N))
    rule_list_df = []
    for ex_rule_frm_rule_stat in rule_stats:
        ante = ex_rule_frm_rule_stat[0]
        cons = ex_rule_frm_rule_stat[1]
        named_cons = names[next(iter(cons))]
        if named_cons in eligible_ante:
            rule_lhs = [names[i][:-2] for i in ante if names[i] in eligible_ante]
            ante_rule = ', '.join(rule_lhs)
            if ante_rule and len(rule_lhs)>1 :
                rule_dict = {'support' : ex_rule_frm_rule_stat[2],
                             'confidence' : ex_rule_frm_rule_stat[3],
                             'coverage' : ex_rule_frm_rule_stat[4],
                             'strength' : ex_rule_frm_rule_stat[5],
                             'lift' : ex_rule_frm_rule_stat[6],
                             'leverage' : ex_rule_frm_rule_stat[7],
                             'antecedent': ante_rule,
                             'consequent':named_cons[:-2] }
                rule_list_df.append(rule_dict)
    rules_df = pd.DataFrame(rule_list_df)
    print("Raw rules data frame of {} rules generated".format(rules_df.shape[0]))
    if not rules_df.empty:
        pruned_rules_df = rules_df.groupby(['antecedent','consequent']).max().reset_index()
    else:
        print("Unable to generate any rule")

Raw rules data frame of 16628 rules generated
```

The output of this code snippet consists of the association rules dataframe that we can use for our analysis. You can play around with the item number, consequent, antecedent, support, and confidence values to generate different rules. Let's take some sample rules generated using transactions that explain 40% of total sales, min-support of 1% (required number of transactions >=45) and confidence greater than 30%. Here, we have collected rules having maximum lift for each of the items that can be a consequent (that appear on the right side) by using the following code.

```
(pruned_rules_df[['antecedent','consequent',
                  'support','confidence','lift']].groupby('consequent')
                                                 .max()
                                                 .reset_index()
                                                 .sort_values(['lift',
                                                 'support','confidence'],
                                                              ascending=False))
```

	consequent	antecedent	support	confidence	lift
4	root vegetables	yogurt, whole milk, tropical fruit	228	0.463636	2.230611
5	sausage	shopping bags, rolls/buns	59	0.393162	2.201037
8	tropical fruit	yogurt, root vegetables, whole milk	92	0.429907	2.156588
1	citrus fruit	whole milk, other vegetables, tropical fruit	66	0.333333	2.125637
10	yogurt	whole milk, tropical fruit	199	0.484211	1.891061
2	other vegetables	yogurt, whole milk, tropical fruit	228	0.643836	1.826724
6	shopping bags	sausage, soda	50	0.304878	1.782992
0	bottled water	yogurt, soda	59	0.333333	1.707635
9	whole milk	yogurt, tropical fruit	228	0.754098	1.703222
3	rolls/buns	yogurt, whole milk, tropical fruit	97	0.522222	1.679095
7	soda	yogurt, sausage	95	0.390625	1.398139

Figure 8-13. *Association rules on the grocery dataset*

Let's interpret the first rule, which states that:

$$\{yogurt, whole\ milk, tropical\ \ fruit \rightarrow \text{root vegtables}\}$$

The pattern that the rule states in the equation is easy to understand—people who bought yogurt, whole milk, and tropical fruit also tend to buy root vegetables. Let's try to understand the metrics. Support of the rule is **228**, which means, all the items together appear in **228** transactions in the dataset. Confidence of the rule is **46%**, which means that **46%** of the time the antecedent items occurred we also had the consequent in the transaction (i.e. 46% of times, customers who bought the left side items also bought root vegetables). Another important metric in Figure 8-13 is Lift. ***Lift*** means that the probability of finding root vegetables in the transactions which have yogurt, whole milk, and tropical fruit is greater than the normal probability

of finding root vegetables in the previous transactions (**2.23**). Typically, a lift value of 1 indicates that the probability of occurrence of the antecedent and consequent together are independent of each other. Hence, the idea is to look for rules having a lift much greater than 1. In our case, all the previously mentioned rules are good quality rules.

This is a significant piece of information, as this can prompt a retailer to bundle specific products like these together or run a marketing scheme that offers discount on buying root vegetables along with these other three products.

We encourage you to try similar analyses with your own datasets in the future and also with the online retail transactions dataset that we used for our market segmentation case study. Considering the dataset `Online Retail` from market segmentation, the workflow for that particular analysis will be very similar. The only difference among these two datasets is the way in which they are represented. You can leverage the following code snippets to analyze patterns from the United Kingdom in that dataset.

```
cs_mba = pd.read_excel(io=r'Online Retail.xlsx')
cs_mba_uk = cs_mba[cs_mba.Country == 'United Kingdom']
# remove returned items
cs_mba_uk = cs_mba_uk[~(cs_mba_uk.InvoiceNo.str.contains("C") == True)]
cs_mba_uk = cs_mba_uk[~cs_mba_uk.Quantity<0]

# create transactional database
items = list(cs_mba_uk.Description.unique())
grouped = cs_mba_uk.groupby('InvoiceNo')
transaction_level_df_uk = grouped.aggregate(lambda x: tuple(x)).reset_index()
[['InvoiceNo','Description']]
transaction_dict = {item:0 for item in items}
output_dict = dict()
temp = dict()
for rec in transaction_level_df_uk.to_dict('records'):
    invoice_num = rec['InvoiceNo']
    items_list = rec['Description']
    transaction_dict = {item:0 for item in items}
    transaction_dict.update({item:1 for item in items if item in items_list})
    temp.update({invoice_num:transaction_dict})

new = [v for k,v in temp.items()]
tranasction_df = pd.DataFrame(new)
del(tranasction_df[tranasction_df.columns[0]])
```

Once you build the transactional dataset, you can choose your own configuration based on which you want to extract and mine rules. For instance, the following code mines for patterns on the top **15** most sold products with min-support of **0.01** (min transactions **49**) and minimum confidence of **0.3**

```
output_df_uk_n, item_counts_n = prune_dataset(input_df=tranasction_df, length_trans=2,
                                              start_item=0, end_item=15)
input_assoc_rules = output_df_uk_n
domain_transac = Domain([DiscreteVariable.make(name=item,values=['0', '1']) for item in
                                                                    input_assoc_rules.
                                                                    columns])
data_tran_uk = Orange.data.Table.from_numpy(domain=domain_transac,  X=input_assoc_rules.
as_matrix(),Y= None)
data_tran_uk_en, mapping = OneHot.encode(data_tran_uk, include_class=True)
```

```
support = 0.01
num_trans = input_assoc_rules.shape[0]*support
itemsets = dict(frequent_itemsets(data_tran_uk_en, support))
confidence = 0.3
rules_df = pd.DataFrame()
 ...  # rest of the code similar to what we did earlier
```

The rest of the analysis can be performed using the same workflow which we used for the groceries dataset. Feel free to check out the `Cross Selling.ipynb` notebook in case you get stuck. Figure 8-14 shows some patterns from the previous analysis on our `Online Retail` dataset.

	consequent	antecedent	support	confidence	lift
8	PACK OF 72 RETROSPOT CAKE CASES	WHITE HANGING HEART T-LIGHT HOLDER, REGENCY CAKESTAND 3 TIER, NATURAL SLATE HEART CHALKBOARD	145	0.971014	5.394404
9	PAPER CHAIN KIT 50'S CHRISTMAS	WHITE HANGING HEART T-LIGHT HOLDER, REGENCY CAKESTAND 3 TIER, NATURAL SLATE HEART CHALKBOARD	94	0.597701	4.341428
3	JUMBO SHOPPER VINTAGE RED PAISLEY	WHITE HANGING HEART T-LIGHT HOLDER, PAPER CHAIN KIT 50'S CHRISTMAS	384	0.879310	4.218819
5	LUNCH BAG BLACK SKULL.	WHITE HANGING HEART T-LIGHT HOLDER, PACK OF 72 RETROSPOT CAKE CASES, LUNCH BAG RED RETROSPOT	227	0.852459	4.078157
4	JUMBO STORAGE BAG SUKI	WHITE HANGING HEART T-LIGHT HOLDER, SET OF 3 CAKE TINS PANTRY DESIGN , JUMBO BAG PINK POLKADOT	405	0.852459	4.016191

Figure 8-14. *Association rules on the online retail dataset for UK customers*

It is quite evident from the metrics in Figure 8-14 that these are excellent quality rules. We can see that items relevant to baking are purchased together and items like bags are purchased together. Try changing the previously mentioned parameters and see if you can find more interesting patterns!

Summary

In this chapter, we read about some simple yet high-value case studies. The crux of the chapter was to realize that the most important part about any analytics or Machine Learning-based solution is the value it can deliver to the organization. Being an analytics or data science professional, we must always try to balance the value aspect of our work with its technical complexity. We learned some important methods that have the potential to directly contribute to the revenue generation of organizations and retail establishments. We looked at ideas pertaining to customer segmentation, its impact, and explored a novel way of using unsupervised learning to find out customer segments and view interesting patterns and behavior. Cross selling introduced us to the world of pattern-mining and rule-based frameworks like association rule-mining and principles like market basket analysis. We utilized a framework that was entirely different from the ones that we have used until now and understood the value of data parsing and pre-processing besides regular modeling and analysis. In subsequent chapters of this book, we increase the technical complexity of our case studies, but we urge you to always have an eye out for defining the value and impact of these solutions. Stay tuned!