Our ability to communicate effectively hinges on our ability, as a species, to recognize patterns. In verbal communication, our brains are interpreting auditory signals and breaking them down into patterns that allow us to decipher the message being conveyed by the speaker. It’s interesting to contemplate the exact origin of human speech. It may have been a caveman who noticed a large saber-toothed cat sneaking up on his fellow caveman brother from a distance, and felt the need to yelp aloud about the impending cat-attack to warn him. We make little chicken scratches on pieces of paper and other people are able to read these chicken scratches because they’re able to recognize patterns in the writing. Unless, of course, you’re dealing with a prescription from a doctor, in which case you need an advanced Pharmacy degree in order to determine exactly what the heck to the doctor wrote on that little piece of paper.

Fortunately, we have advanced beyond the need to warn fellow peers about the dangers of tar pits or Wooly Mammoth stampedes. However, your survival as an online merchant does still depend largely on your ability to communicate with your customers. You need to be able to discern patterns in their shopping behavior, by looking at exactly how they are using your site, and then use any information they provide you to your own advantage by helping talk to them as individuals.

In this chapter, we’re going to build the home page. While we haven’t created every aspect of our site, we’ve got enough data to go on that we can create product recommendations based on a user’s browsing habits. We’re also going to help them while they’re browsing our site by setting up intelligent product cross-sales on each product page.

The name of the game is always increasing revenue. Since this hinges on our ability to talk to each customer, with as personal a touch as we can manage, let’s look at what our customers are saying to us already.

Product Page Recommendations

Whether or not you’ve considered it yet, the customers shopping on your site are providing you with quite a bit of information about themselves. Sure, you may not know their name or where they live quite yet, but as soon as they hit your site, they are giving you some feedback about what kind of person they are. It’s up to you to recognize what information they’re providing, and how best to use this information in order to help them use your site more effectively. In this way, we hope to increase the value our site is providing the customer, enhance their shopping experience, and hopefully get more sales revenue out of each customer.

So what are our customers telling us? Think about the ways in which your customer currently interacts with your site. They can browse categories, browse products, add products to a shopping cart, place orders, and search the product catalog. There is actually quite a lot that we can do with this data,
even though it’s a fairly short list. Let’s look at each of these one at a time, starting with what we can do on the product page itself.

Order-Based Filtering

One option we have in creating product recommendations is to create relationships between products manually. We could create a new “relationship” model and make recommendations based on the products that we specify by hand. However, this approach is tedious and won’t scale as the number of products we have on our site grows very large.

The easiest thing for us to do is look at our order history. A customer can create an order with more than one item. It makes perfect sense, then, that if a customer ordered a Backwater Boondocks Banjo along with a set of Superior Brand Banjo strings, other customers who view the Backwater Boondocks Banjo might be interested in that same set of Superior Brand Banjo strings. It stands to reason that we should display that set of banjo strings on the banjo page since there’s a high probability that future customers might also want to purchase this product as well. If you were to label these product recommendations on the product page, you might use the text: “Also purchased with this product…”

Doing this in code is a pretty simple process. We merely need to look up any orders that included the given product we’re interested in evaluating. Say we’re looking up cross-sells for the Ruby Axe Guitar:

```python
my_product = Product.active.filter(name="Ruby Axe Guitar")
orders = Order.objects.filter(orderitem__product=my_product)
```

Then, we look up any order items that we in those same orders, but that were not the Ruby Axe Guitar:

```python
order_items = OrderItem.objects.filter(order__in=orders).exclude(product=my_product)
```

From this, we just need to get the associated products from the list of order items. We call the `distinct()` method on the result set in order to eliminate any duplicates:

```python
products = Product.active.filter(orderitem__in=order_items).distinct()
```

It makes the most sense to retrieve these product recommendations as a method we can call on a model instance. Open `models.py` and add this method to your `Product` model:

```python
def cross_sells(self):
    from ecomstore.checkout.models import Order, OrderItem
    orders = Order.objects.filter(orderitem__product=self)
    order_items = OrderItem.objects.filter(order__in=orders).exclude(product=self)
    products = Product.active.filter(orderitem__in=order_items).distinct()
    return products
```

Notice that we’re doing the import of the `Order` and `OrderItem` models down in the method instead of at the top of the file. In this particular case, that’s intentional. You see, the `OrderItem` model is dependent on the `Product` model itself, which we import at the top of the file. If we try to import the `OrderItem` model at the top of the file that contains the `Product` model, we get an import error, since we have a series of imports that result in a circle. For example, let’s say you have class A in one module called A and class B in another module called B. Module A imports class B, and module B import class A. What we have here is a circular dependency, and when this code is run, Python gets very confused about what it’s supposed to do. These kinds of situations result in runtime errors that are very difficult to track down in your code. The best solution is to either change where you are performing the import statement in your code, or move stuff around into different modules.