4.6.2

Aggregate the Data

A dataset often has natural aggregations, or groups, that we can analyze both separately and in comparison to one another. By using `loc` and `iloc`, we could divide a DataFrame into variables, one for each group, and then do an analysis on each. But, that method is error prone and prohibitively time consuming, depending on the number of needed groups. Enter the Pandas `groupby` function. We can use `groupby` to automatically subdivide a DataFrame into groups based on a column in the DataFrame.

For example, consider a sales DataFrame that has 10 rows and 4 columns of data, as the following code shows:

```
sales_df = pd.DataFrame({
    "item": ["tomatoes", "onions", "potatoes", "tomatoes", "mushrooms", "potatoes", "onions", "tomatoes", "on
    "store": ["Downtown", "Downtown", "Midtown", "Midtown", "Downtown", "Midtown", "Downtown", "Midtown", "Mi
    "quantity": [15, 26, 10, 12, 2, 3, 17, 16, 20, 9],
    "total_price": [50.25, 39.52, 50.10, 40.20, 8.46, 15.03, 25.84, 53.60, 30.40, 30.15]})
sales_df
```

Running the preceding code produces the output that the following image shows:

	item	store	quantity	total_price
0	tomatoes	Downtown	15	50.25
1	onions	Downtown	26	39.52
2	potatoes	Midtown	10	50.10
3	tomatoes	Midtown	12	40.20
4	mushrooms	Downtown	2	8.46
5	potatoes	Midtown	3	15.03
6	onions	Downtown	17	25.84
7	tomatoes	Midtown	16	53.60
8	onions	Midtown	20	30.40
9	tomatoes	Midtown	9	30.15

To use `groupby`, we must first find a column that contains the values we want to use for grouping. Such a column should contain repeated values so that the groups will be meaningful. For example, in the "item" column of the sales DataFrame, "tomatoes", "onions", and "potatoes" all occur more than once.

NOTE

Each value in the chosen grouping column will get its own row in a grouped DataFrame. That's why we might not want to choose a column that contains many unique values.

The following code uses `groupby` to group the DataFrame by the "item" column:

```
item_group_df = sales_df.groupby("item")
item_group_df
```

Running the preceding code produces the output that the following image shows:

```
item_group_df = sales_df.groupby("item")
item_group_df
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fade1fa4a30>
```

In the preceding image, notice that the output is as follows:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fade1fa4a30>
```

The output is a string of characters that has no obvious meaning! That's because the groups are packed together, waiting for a Pandas aggregation function to display the data. We use these aggregation functions to be able to compare the groups.

Now that you've learned how to aggregate the data, you'll next learn how to compare the data aggregations.