```
'Output':
               open    high     low   close  close_4_ahead
date
2017-07-23  555.89  578.97  411.78  413.06         385.48
2017-07-24  412.58  578.89  409.21  440.70         406.05
2017-07-25  441.35  541.66  338.09  406.90         384.77
2017-07-26  407.08  486.16  321.79  365.82         345.66
2017-07-27  417.10  460.97  367.78  385.48         294.46
```

Observe that the tail of the column **close_4_head** contains **NaNs**.

```
data_subset_BCH.tail()
'Output':
                open     high      low    close  close_4_ahead
date
2018-01-06  2583.71  2829.69  2481.36  2786.65        2895.38
2018-01-07  2784.68  3071.16  2730.31  2786.88            NaN
2018-01-08  2786.60  2810.32  2275.07  2421.47            NaN
2018-01-09  2412.36  2502.87  2346.68  2391.56            NaN
2018-01-10  2390.02  2961.20  2332.48  2895.38            NaN
```

Rolling Windows

Pandas provides a function called **rolling()** to find the rolling or moving statistics of values in a column over a specified window. The window is the "number of observations used in calculating the statistic." So we can find the rolling sums or rolling means of a variable. These statistics are vital when working with timeseries datasets. Let's see some examples.

Let's find the rolling means for the closing variable over a 30-day window.

```
# find the rolling means for Bitcoin cash
rolling_means = data_subset_BCH['close'].rolling(window=30).mean()
```

The first few values of the **rolling_means** variable contain **NaNs** because the method computes the rolling statistic from the earliest time to the latest time in the dataset. Let's print out the first five values using the **head** method.

```
rolling_means.head()
Out[75]:
date
2017-07-23    NaN
2017-07-24    NaN
2017-07-25    NaN
2017-07-26    NaN
2017-07-27    NaN
```

Now let's observe the last five values using the **tail** method.

```
rolling_means.tail()
'Output':
date
2018-01-06     2403.932000
2018-01-07     2448.023667
2018-01-08     2481.737333
2018-01-09     2517.353667
2018-01-10     2566.420333
Name: close, dtype: float64
```

Let's do a quick plot of the rolling means using the Pandas plotting function. The output of the plot is shown in Figure 11-2.

```
# plot the rolling means for Bitcoin cash
data_subset_BCH['close'].rolling(window=30).mean().plot(label='Rolling
Average over 30 days')
```

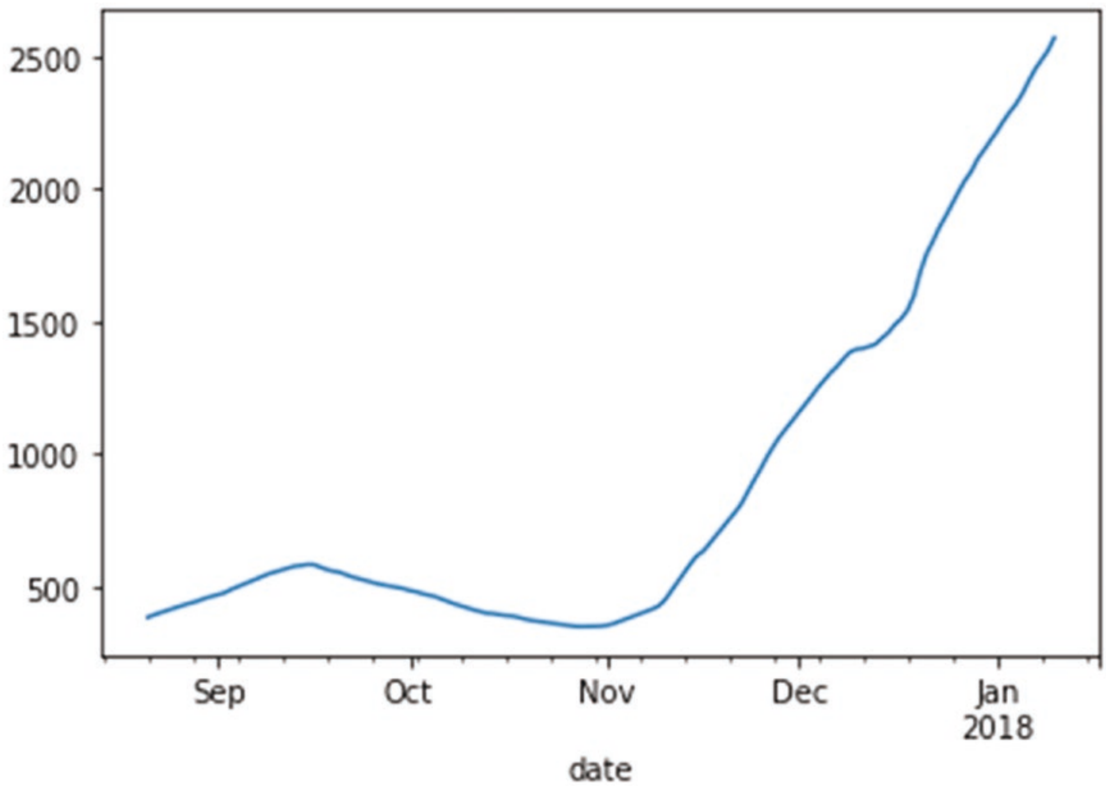

***Figure 11-2.** Rolling average closing price over 30 days for Bitcoin Cash*

More on plotting in the next chapter.

CHAPTER 12

Matplotlib and Seaborn

It is critical to be able to plot the observations and variables of a dataset before subjecting the dataset to some machine learning algorithm or another. Data visualization is essential to understand your data and to glean insights into the underlying structure of the dataset. These insights help the scientist in deciding with statistical analysis or which learning algorithm is more appropriate for the given dataset. Also, the scientist can get ideas on suitable transformations to apply to the dataset.

In general, visualization in data science can conveniently be split into **univariate** and **multivariate** data visualizations. Univariate data visualization involves plotting a single variable to understand more about its distribution and structure, while multivariate plots expose the relationship and structure between two or more variables.

Matplotlib and Seaborn

Matplotlib is a graphics package for data visualization in Python. Matplotlib has arisen as a key component in the Python data science stack and is well integrated with NumPy and Pandas. The **pyplot** module mirrors the MATLAB plotting commands closely. Hence, MATLAB users can easily transit to plotting with Python.

Seaborn, on the other hand, extends the Matplotlib library for creating beautiful graphics with Python using a more straightforward set of methods. Seaborn is more integrated for working with Pandas DataFrames. We will go through creating simple essential plots with Matplotlib and seaborn.

Pandas Plotting Methods

Pandas also has a robust set of plotting functions which we will also use for visualizing our dataset. The reader will observe how we can easily convert datasets from NumPy to Pandas and vice versa to take advantage of one functionality or the other. The plotting features of Pandas are found in the **plotting** module.

© Ekaba Bisong 2019
E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*,
https://doi.org/10.1007/978-1-4842-4470-8_12