Shing Chi Leung's Learning Pandas from Zero

Chapter 2: Creating your first dataframe

In this chapter, we will go through the fundamental steps in constructing your own tables. We will examine the command **DataFrame** and **read_csv** and go through how we can convert different types of data into a dataframe.

Why DataFrame?

Before we start, one may question what is so special about **DataFrame**. Dataframe is the main type of object used in Pandas. Tables are stored and processed as different dataframes. In order to benefit the many powerful options available in Pandas,
the table data needs to be stored in a dataframe variable.

In Pandas, another main datatype is called **Series**. It is in general a one-dimensional array data with each entry corresponding to a key. Therefore a series can be viewed as a slice of a dataframe. Since most options available in series are a subset of that
of the DataFrame, we do not explicitly discuss series in this book.

Build a table from a dictionary

Now let us assume this scenario. You have a table as following (Table 1). The table contains data of the amount of fruits in different shops. Instead of using the more familiar Microsoft Excel to handle, you want to experiment with Pandas and see what
you can do with this package.

Table 1: Fruit amount surveyed in different shops

Shop	Apple	Orange	Kiwi
A	18	23	30
B	23	18	29
C	31	37	30

The first step is to pass this set of data. The first way is to make this set of data into a dictionary. Then we make a dataframe object by passing the dictionary into this object, namely

```
df = pandas.DataFrame(dictionary, index=..., columns=...)
```

The first input is compulsory, it can be an iterable such as a list, array or dictionary. Here we consider a dictionary first. There are a number of auxillary setting you can pass during declaration, or can be set after the declaration. Here we consider the simplest case first.

To make the above Table 1 into a dictionary which can be converted into a dataframe directly, we used the following code:

```
import pandas as pd       # if not imported Python has no idea what DataFrame
means!
data = {
  "Shop":["A","B","C"],
  "Apple":[18,21,31],
  "Orange":[23,18,37],
  "Kiwi":[30,29,30]
}
df = pd.DataFrame(data)
```

You might notice that in preparing the dictionary, we set the keys to be the header of all columns. The values for each key correspond to all values under that header. And notice that the dataframe can contain numerical and alphabetical input.

The results of df can be immediately checked by printing the object out (Figure 1).

```
In [82]: print(df)
   Shop  Apple  Orange  Kiwi
0     A     18      23    30
1     B     21      18    29
2     C     31      37    30
```

Figure 1: Output of the dataframe df

One thing to notice is that Pandas by default assign integers 0, 1, 2 and so on as the keys for the entry when there is no specification during declaration. In some context they are referred as indices. We will return to this later in this chater.

If you get the same as that shown in Figure 1, congratulations! This is the very first step of your first experience in Pandas.

When we check the variable type of df by typing

```
type(df)
```

we will receive the class 'pandas.core.frame.DataFrame'.

Build a table from a list

The flexibility of Pandas allows us to choose other types of iterable for passing the data. Now we repeat the process by creating a list containing the data. We will use the following code to construct a list and then pass the list to the dataframe object.

```
import pandas as pd
data = [
  ["A",18,23,30],
  ["B",21,18,29],
  ["C",31,37,30]
]
column_names = ["Shop", "Apple", "Orange", "Kiwi"]
df = pd.DataFrame(data, columns=column_names)
```

By glancing at the code, there are four major differences compared to the above code for using dictionary, including:

1. The square bracket (obvious but easy to miss)
2. Each entry in the list corresponds to the data from the same row
3. No column names passed in the list
4. The column names stored in a separate list

The result dataframe will be identical to Figure 1. Again, this will be very helpful to practice once on your own the whole dataframe declaration!

What if we omit the column setting?

In the version using list, you might have noticed the notation *columns=column_names* is used in order to obtain a dataframe which is identical to the version using dictionary. Then, what will happen if we do not pass this setting during declaration?

To experiment, let us modify the code a little by writing

```
import pandas as pd
data = [
  ["A",18,23,30],
  ["B",21,18,29],
  ["C",31,37,30]
]
column_names = ["Shop", "Apple", "Orange", "Kiwi"]
df = pd.DataFrame(data)
```

When we print the dataframe again, this time we obtain something listed in Figure 2.

```
In [87]: print(df)
```

```
   0   1   2   3
0  A  18  23  30
1  B  21  18  29
2  C  31  37  30
```

Figure 2: Output of the dataframe df without column names

From this it becomes clear that Pandas by default name all rows and columns by integers 0, 1, 2. When the user offers lists of column names or row names, Pandas will overwrite these integers by the elements from those lists.

In light of that, one may be tempted to question if a similar treatment can be applied to the keys. The answer is... YES! By passing a list which contains the names to be used for the indices, the output dataframe will use the elements from the list automatically. In fact, giving meaningful keys and column names will be very useful for processing the dataframe effectively as we will show in later chapters.

To do so, we first modified the list version of the code as follows:

```
import pandas as pd
data = [
  ["A",18,23,30],
  ["B",21,18,29],
  ["C",31,37,30]
]
column_names = ["Shop", "Apple", "Orange", "Kiwi"]
index_names = ["Shop 1", "Shop 2", "Shop 3"]
df = pd.DataFrame(data, columns=column_names, index=index_names)
```

When we examine the dataframe, we will obtain the result shown in Figure 3.

```
In [89]: print(df)
```

```
       Shop  Apple  Orange  Kiwi
Shop 1    A     18      23    30
Shop 2    B     21      18    29
Shop 3    C     31      37    30
```

Figure 3: Output of the dataframe df with index and column names

Other types of data available

It is also possible to pass a JSON type data, as it has a structure very similar to a dictionary, to create a dataframe. However, we do not discuss the detailed process as treating JSON type data will involve further notation used in *json* module and other web fetching interface in order to achieve a complete discussion.

Build a dataframe from a csv file

In the above examples, we have paid attention to how to create a table from scratch. However, this will mostly be applicable to very small scale table or table for your own practice. In general, we use Pandas to read a large dataset or datafile. Here let us examine the case when the datafile is a csv file.

For those who have not heard about csv file, a csv file is the short form of comma-separated values file. It is a very compact form to store an array of data. Each line corresponds to one row of data, with a comma to separate between consecutive data. The first row is usually used as the header for the column names. Notice that no space is needed after each comma or otherwise the machine might confuse the comma as a necessary part in the string. In Figure 4 we show an example where we convert the table used in this chapter into a csv file.

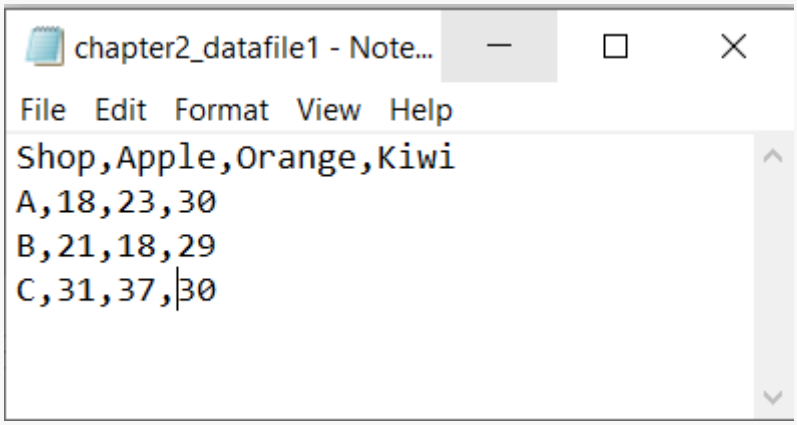

Figure 4: An example of a csv file

Python has package specific for processing and reading csv file. For Pandas, we do need to call that module explicitly as Pandas contains all tools necessary for configuring and accessing csv file. We use the command

```
df = pandas.read_csv(file, header=..., index_col=...)
```

Again, the filename (or with the path to the file) is the compulsory input for this command. Others are auxillary and can be set according to the user's specific needs. Some useful ones include:

- header: An integer which tells Pandas not to read the line before that, and make the header line as column names
- index_col: An integer which tells Pandas not to set that column as part of the dataframe, but to use that column as keys

There are much more fine tuning setting available in this command. Since this book serves as the introductory guide to grasp
the essential technique useful in Pandas, we will not discuss each configuration one-by-one.

By using the above csv file example, we can set up the code to read the csv file by

```
import pandas as pd
filename = "chapter2_datafile1.csv"
df = pd.read_csv(filename)
```

The procedure is simpler than using list and dictionary because most data input effort is done while preparing the csv file. We will obtain a dataframe identical to Figure 1 of this chapter.

Build a dataframe from an excel file

The last command we want to discuss in this chapter is to ask pandas to read an excel file. It usually becomes a dilemma when the data is already available in an excel file, why bothers to use Pandas again? There are two reasons here. First, besides the reason of pedagogy, when the table is large (10000 lines or above), the performance of Pandas is much better than standard spreadsheet software because Python does not need to handle the expensive graphical user interface in the operation. User also does not need to scroll up and down in the table to process the data.

Second, in standard spreadsheet software like Excel, graph plotting and other calculation can be limited by the design. The setting in Pandas can be fine-tuned: for example, the colour template in each plot or choice of partial data to be plotted, can be
adjusted one by one.

Again, we will use the same table (Table 1) as a reference to demonstrate how to read the excel file. We will first prepare an excel file containing all entry in the table (Figure 5).

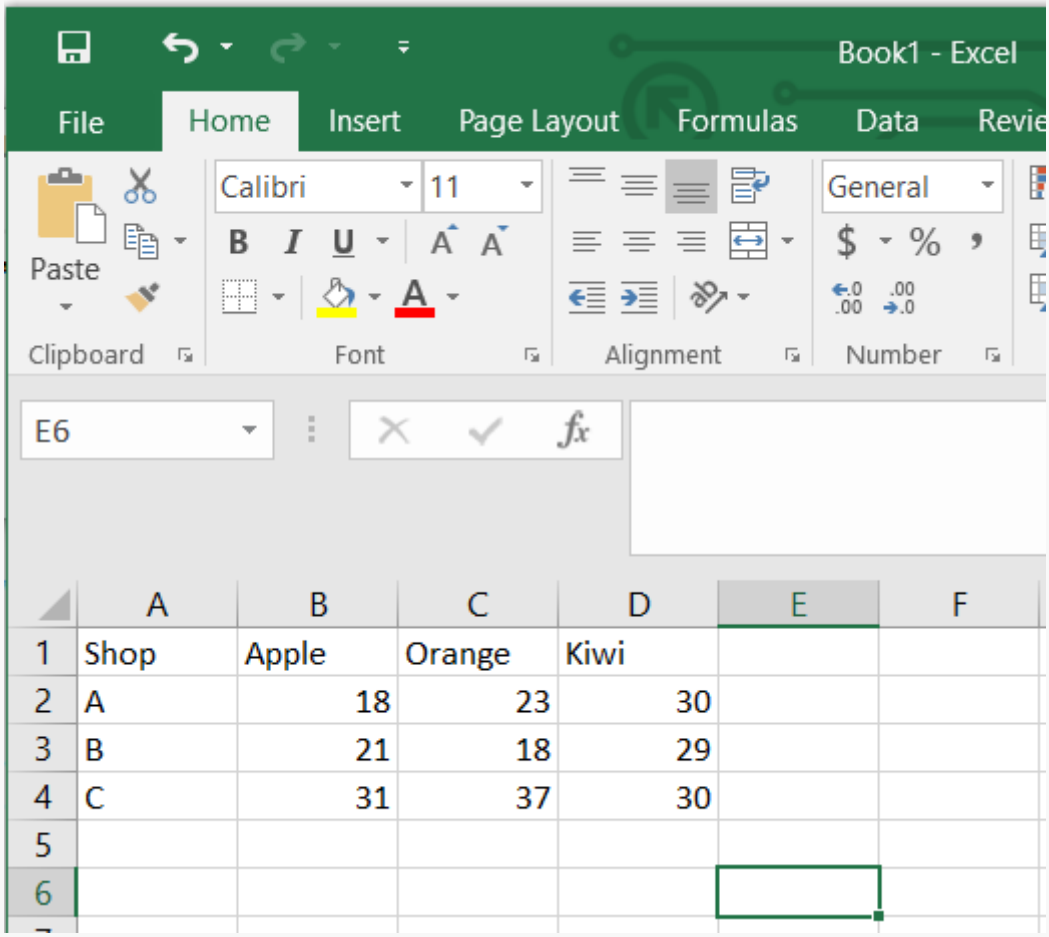

Figure 5: An example of an excel file

Then we will use the command

```
df = pandas.read_excel(file, sheet_name=..., header=..., index_col=...)
```

to convert the data in the excel table to a dataframe. Again, the first entry is compulsory which is the file we want to pass, others are again non-compulsory setting which depend on the scenario.

Some useful settings include:

- sheet_name: integer or a list of integer, the name of the sheet to be converted in the excel file
- header: integer, similar to read_csv, it controls Pandas to skip previous lines and set the header line for column names
- index_col: integer, similar to read_csv, it controls Pandas to pick a selected columns as the keys of each entry

```
import pandas as pd
filename = "chapter2_datafile2.xlsx"
df = pd.read_excel(filename)
```

When we output the datafile to check, we will obtain identical results as Figure 1.

Summary

In this chapter we have explored multiple ways to generate a dataframe by using data available in a list, a dictionary, a csv file and an excel file. We have learnt that the index (key) and column names are changeable in Pandas, and are not necessarily letters or integers. We have covered three essential methods in the Pandas package, including:

```
# Passing iterables (list, dictionary) to a dataframe
df = pandas.DataFrame(data, ...)

# Ask Pandas to read a csv file and generate a dataframe
df = pandas.read_csv(csv_file, ...)

# Ask Pandas to read an excel file and generate a dataframe
df = pandas.read_excel(excel_file, ...)
```

Exercises

The exercises of this eBook has two functions. First, they serve as a reinforcement opportunities to practice some of the coding skills presented in that chapter. Second, they serve as a prompt for readers to explore different functionality available in Pandas with real-life examples as a drive. The solutions of the exercises are available in the **solution** chapter.

1. Your friend gave you a table like Table 2 (below).

Name	ID	Age	Gender
Ann	A01	20	F
Ben	A02	35	M
Carla	A03	28	F
Doug	A04	31	M

Try to use

- a dictionary,
- a list,
- a csv file and
- an excel file

to pass the data to a dataframe object.

2. If we want to make the output similar to Figure 3 for the case of a csv or an excel file in Exercise 1, which setting should we try to use? Can you do it by adding a new list or modifying the csv file?

3. Your friend tried to copy the code and made a csv file for that. Now your friend wants to make the table shorter by skipping the first two entries (No Ann and Ben). But your friend wants to keep the column names. What will you suggest?