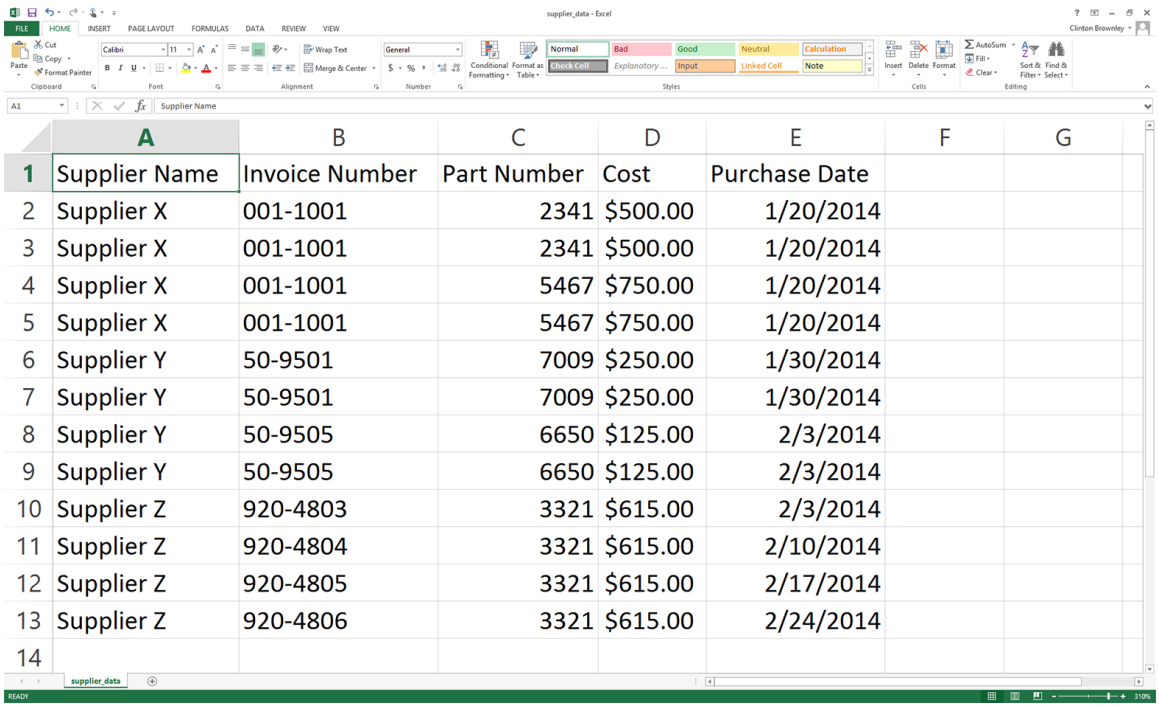
2Python 读写 CSV 文件

CSV 文件格式是非常简单的贮存和共享数据的方法。CSV 文件以明文方式贮存数据表，表格的每一格是数字或字串。CSV 文件与 Excel 文件相比的一个重要优点是有许多程序可以贮存、转换、处理明文文件；而只有少量的程序可以处理 Excel 文件。任何表格程序，word 处理程序，文本编辑器都可以处理明文文件，但是它们并不都能处理 Excel 文件。尽管 Excel 的功能很强大，但是当你处理 Excel 文件时，你基本上仅限以 Excel 可以完成的任务。CSV 文件给予你发送数据到合适工具以进行你想要的工作的自由--你可以用 Python 构建自己的工具。你失去一些 Excel 的特征，如 Excel 的每个单元格有特定的类型（数值，文本，货币，日期等），CSV 文件只是原始数据。非常感谢，python 可以聪明的识别不同的数据类型，如我们在第一章所见。另一个妥协是使用 CSV 文件不能贮存公式，只有数据。但是通过数据贮存和数据处理，你可以应用你的程序来处理不同的数据集。你也很容易发现--更难传递（在程序和在数据文件中）！--错误，当处理和贮存分开时。

为了能处理这种格式，你需要创建 CSV 文件(你也可以从 https://github.com/cbrownley/foundations-for-analyticswith-python/blob/master/csv/supplier_data.csv 下载)：

1. 打开电子表格并添加以下数据，如图 2-1.



	A	B	C	D	E	F	G
1	Supplier Name	Invoice Number	Part Number	Cost	Purchase Date		
2	Supplier X	001-1001	2341	$500.00	1/20/2014		
3	Supplier X	001-1001	2341	$500.00	1/20/2014		
4	Supplier X	001-1001	5467	$750.00	1/20/2014		
5	Supplier X	001-1001	5467	$750.00	1/20/2014		
6	Supplier Y	50-9501	7009	$250.00	1/30/2014		
7	Supplier Y	50-9501	7009	$250.00	1/30/2014		
8	Supplier Y	50-9505	6650	$125.00	2/3/2014		
9	Supplier Y	50-9505	6650	$125.00	2/3/2014		
10	Supplier Z	920-4803	3321	$615.00	2/3/2014		
11	Supplier Z	920-4804	3321	$615.00	2/10/2014		
12	Supplier Z	920-4805	3321	$615.00	2/17/2014		
13	Supplier Z	920-4806	3321	$615.00	2/24/2014		
14							

图 2-1. 在 supplier_data.csv 文件中添加数据

2. 在桌面将文件另存为 supplier_data.csv.

要确认 supplier_data.csv 实际上是明文文件:

1. 最小化所有的打开的窗口，在桌面定位到 supplier_data.csv 文件。
2. 右击文件。
3. 选择“打开方式”然后选择文本编辑器如 Notepad, Notepad++, 或 Sublime Text。

当你用文本编辑器打开文件时，它看起来像图 2-2。

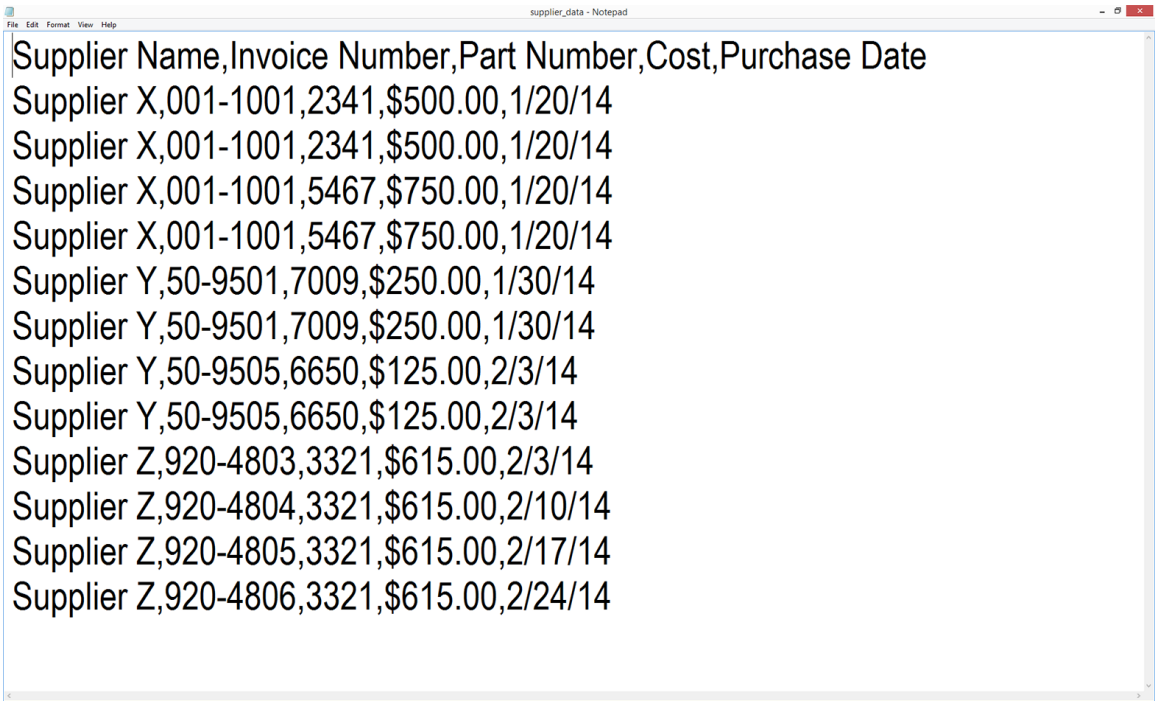


图 2-2. Notepad 中的 supplier_data.csv

如你所见，文件是简单的明文文件。每一行有 5 个值由逗号分隔。另一种想法是在电子表格中，逗号勾画了 5 列。你现在可以关闭文件了。

基础 Python 与 pandas

本章的每一节都有两个版本的代码，来完成特定的数据分析工作。第一种版本是用基础 pthon 来完成任务。第二个版本是用 pandas 来完成。你会看到 pandas 更容易完成任务，使用的代码相对少。但是我以基础 python 开始，你可以学习如何用一般的编程概念和操作来完成特定的任务。通过两种版本，你可以选择用 pandasa 快速的完成任务或学习一般的编程和解决问题的技巧。我并不像解释基础 python 版本一样详细的解释 pandas 版本。你可以把例子当作“cookbook”来用 Pandas 完成任务。但是你想成为 pandas 的高手，我推荐你读 Wes McKinney 的书 Python for Data Analysis(O’Reilly) 。

读和写 CSV 文件(Part 1)

基础 Python,没有 csv 模块

我们来学习如何用基础 python 读，处理，写 CSV 文件（不使用内置的 csv 模块）。首先看一下例子，然后你会理解 CSV 模块到底如何。

要处理 CSV 文件，我们先创建新的 python 脚本，1csv_read_with_simple_Parsing_and_write.py.

在 Spyder 或文本编辑器中输入如下代码:

```
1 #!/usr/bin/env python3
2 import sys
3
4 input_file = sys.argv[1]
5 output_file = sys.argv[2]
6
7 with open(input_file, 'r', newline='') as filereader:
8 with open(output_file, 'w', newline='') as filewriter:
9 header = filereader.readline()
10 header = header.strip()
11 header_list = header.split(',')
12 print(header_list)
13 filewriter.write(','.join(map(str,header_list))+'\n')
14 for row in filereader:
15 row = row.strip()
16 row_list = row.split(',')
17 print(row_list)
18 filewriter.write(','.join(map(str,row_list))+'\n')
```

将脚本在桌面另存为1csv_read_with_simple_parsing_and_write.py。图2-3, 2-4,和 2-5展示脚本 Anaconda Spyder, Notepad++(Windows), 和 TextWrangler (macOS)中的样子。

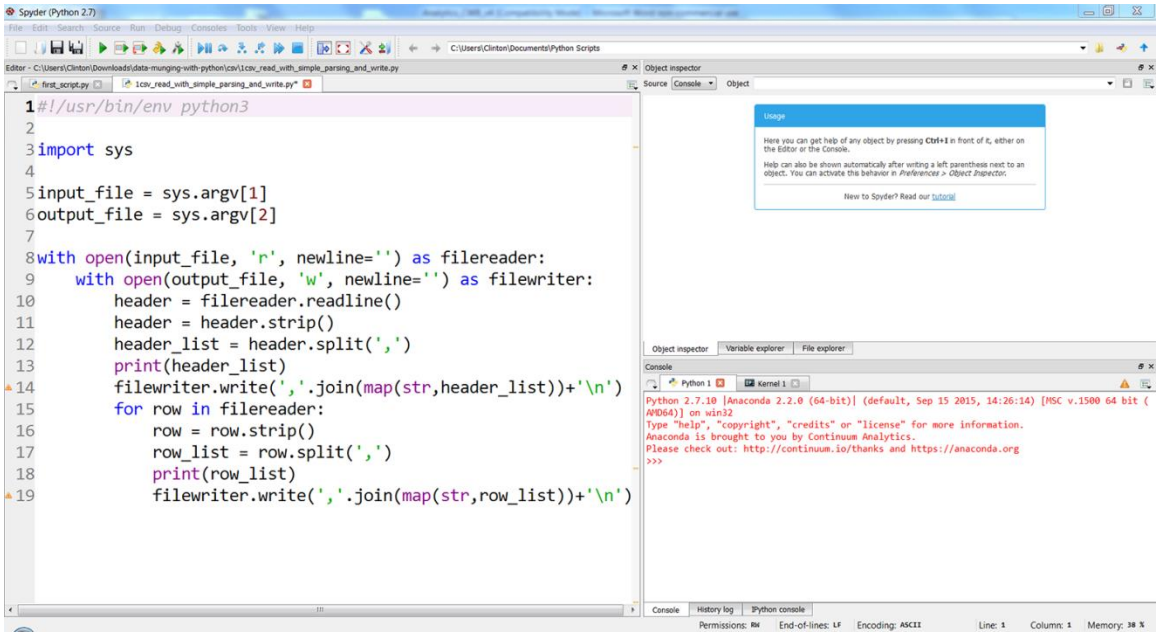


图 2-3. 1csv_read_with_simple_parsing_and_write.py 脚本在 Anaconda Spyder 中

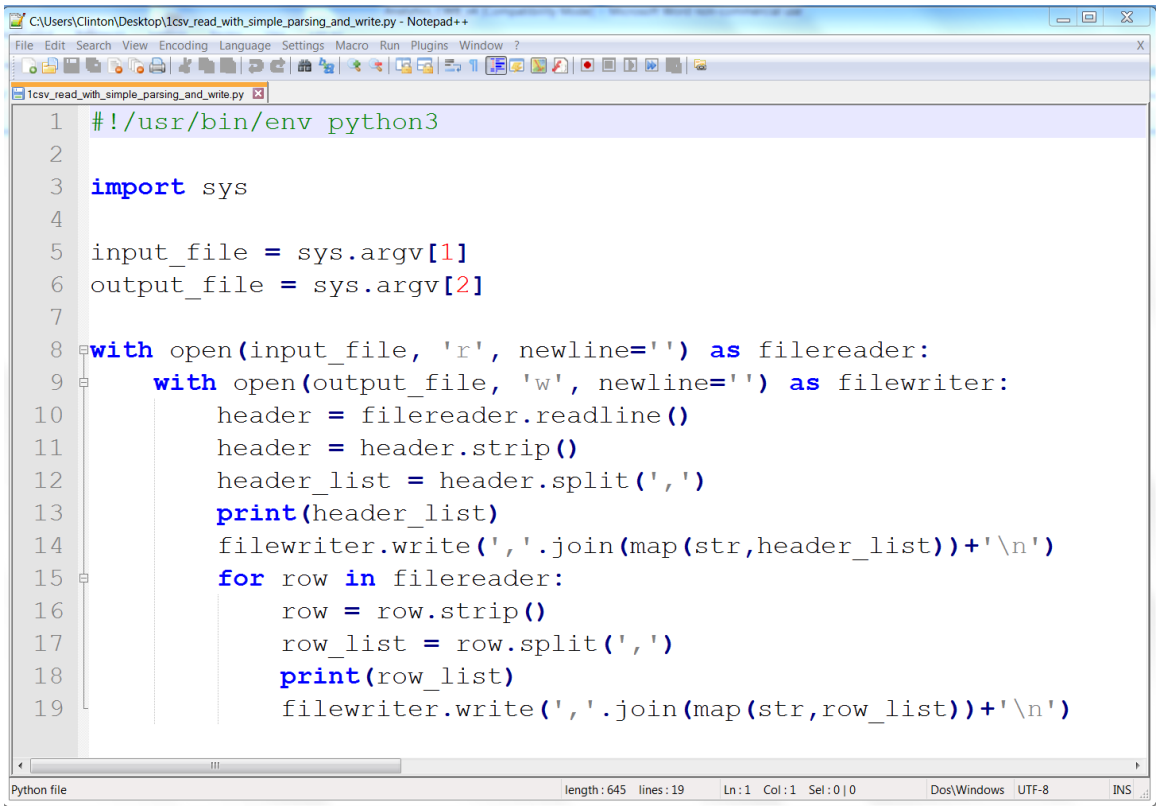


图 2-4. 1csv_read_with_simple_parsing_and_write.py 脚本在 Notepad++ (Windows)中

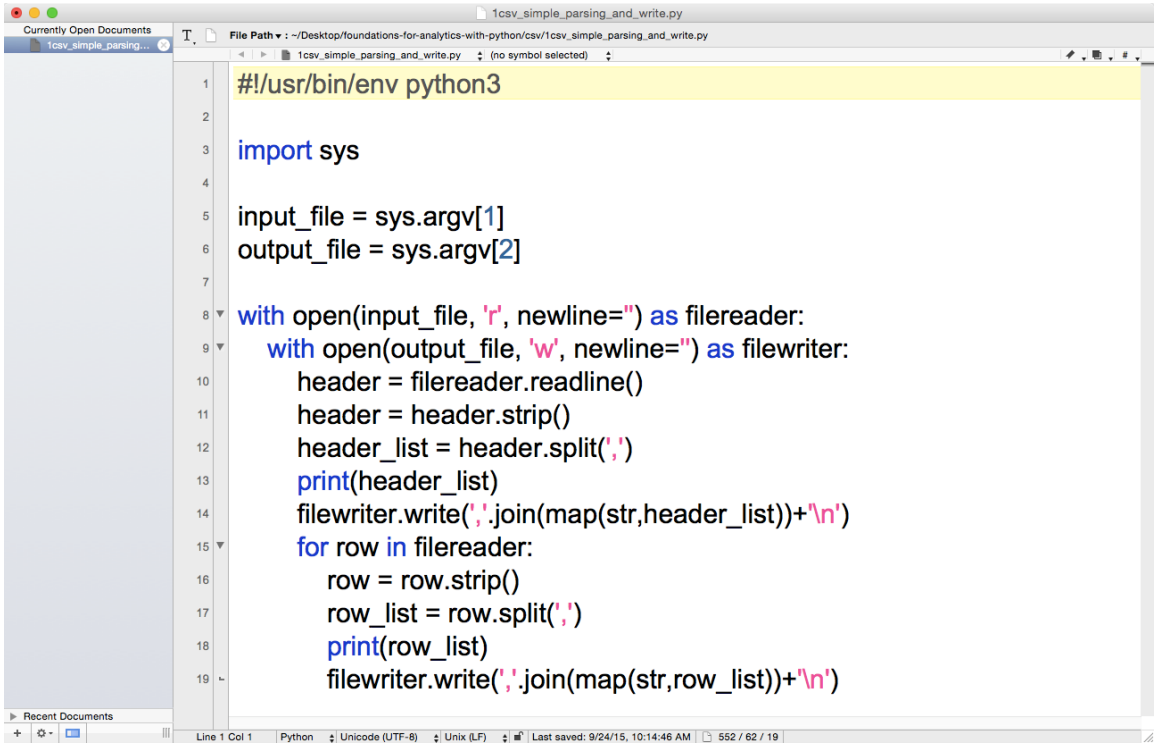


图 2-5. 1csv_read_with_simple_parsing_and_write.py 脚本在 Text - Wrangler (macOS)中

在我们运行脚本和观察输出之前，我们来看一下代码想做什么。我们会一行一行的讨论代码。

```
#!/usr/bin/env python3
import sys
```

正如第一章所讨论的，第一行是注释行，使脚本可以在不同的操作系统转换。第二行导入 python 的内置模块，sys 模块，使你可以从命令行发送额外的输入到你的脚本。

```
input_file = sys.argv[1]
output_file = sys.argv[2]
```

第 4 和第 5 行用 sys 模块的 argv 参数,它是传递到脚本的命令行参数列表，即当你运行脚本时在命令行中输入东西。这是一个通用的命令行参数版本，我们在 Windows 计算机用来读取 CSV 输入文件并写入 CSV 输出文件：

```
python script_name.py "C:\path\to\input_file.csv" "C:\path\to\output_file.csv"
```

第一个词，python，告诉你的计算机使用 python 程序来处理后面的命令行参数。Python 收集余下的参数到特定的列表中，称为 argv。它保存列表中的第一个位置，argv[0]，作为脚本名，所以 argv[0]指向 script_name.py。下一个命令行参数是"C:\path\to\input_file.csv"，是 CSV 输入文件的路径。Python 贮存这个值于 argv[1],所以在第 4 行将这个值赋给这量名 input_file。最后一个命令行参数"C:\path\to\output_file.csv"是 CSV 输出文件的路径。python 将它贮存于 argv[2]，第 5 行将这个值赋给变量名 output_file。

```
with open(input_file, 'r', newline='') as filereader:
with open(output_file, 'w', newline='') as filewriter:
```

第 7 是 with 语句，它打开 input_file 作为文件对象。'r'指明读模式，即 input_file 是打开来读的。第 8 行是另一个 with 语句，它打开 output_file 作为文件对象， filewriter。 'w'指明写模式，即 output_file 是打开来写的。如我们在“Modern File-Reading Syntax”所见，with 符号很有帮助，因为它可以自动的关闭文件对象，当语句退出时。

```
header = filereader.readline()
header = header.strip()
header_list = header.split(',')
```

第 9 行用文件对象的 readline 方法来读输入文件的第一行，本例中是头行，作为字串， 赋值给变量 header。第 10 行用 string 模块的 strip 函数自 header 字串尾部去除空格，tabs， 和 newline 字符，并赋值给 stripped 版本的字串 header。第 11 行用 string 模块的 split 函数按逗号将字串分割为列表，值是列的标题，将列表赋值给变量 header_list。

```
 print(header_list)
filewriter.write(','.join(map(str,header_list))+'\n')
```

第 12 是打印语句，它打印 header_list 的值 (如，列标题)到屏幕。

第 13 用 filewriter 对象的 write 方法来将 header_list 值写到 output file。因为这一行有很多东西,我们彻底的看看。Map 应用 str 函数到 header_list 的每一个值以确证每个值是字串。然后 join 函数将逗号放入 header_list 的两个值之间，转换列表到字串。接着，在字串尾部添加

```
for row in filereader:
row = row.strip()
row_list = row.split(',')
Baseprint(row_list)
filewriter.write(','.join(map(str,row_list))+'\n')
```

第 14 行创建 for 循环来遍历输入文件的余下各行。第 15 行用 strip 函数自行字串的头部和尾部去除空格，tabs， 和 newline 符号。第 16 行用 split 函数按逗号将字串分割为列表，每个值是一行中的列的值，将列表赋值给变量 row_list。第 17 行打印 row_list 的值到屏幕，第 18 行将值写入输出文件。脚本为输入文件的每一行执行第 15 到 18 行，因为这四行在第 14 行的 for 循环的下面。你可以在命令行窗口或终端窗口中检查脚本：

命令行 (Windows)

1. 打开命令行窗口。
2. 导航到你的桌面(Python 脚本保存的地方)。

   输入下面的一行并按回车键即可完成:

```
cd "C:\Users\[Your Name]\Desktop"
```

3. 运行 Python 脚本。

   输入下面的一行并按回车键即可完成:

```
python lcsv_simple_parsing_and_write.py supplier_data.csv\
output_files\loutput.csv
```

终端 (macOS)

1. 打开终端窗口。
2. 导航到你的桌面 (你保存 Python 脚本的地方)。

输入下面的一行并按回车键即可完成:

```
cd /Users/[Your Name]/Desktop
```

3. 使 Python 脚本可执行。

输入如下一行并按回车键即可:

```
chmod +x 1csv_simple_parsing_and_write.py
```

4. 运行 Python 脚本。

输入如下一行并按回车键即可:

```
./1csv_simple_parsing_and_write.py supplier_data.csv\
output_files/1output.csv
```

你可以看到如图 2-6 所示的命令行中终端输出。

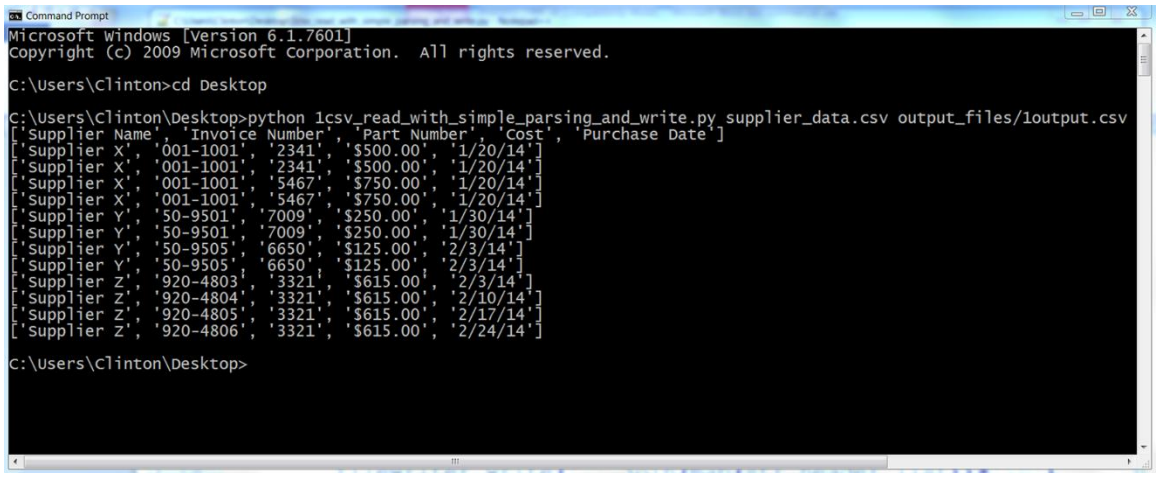


图 2-6. 1csv_read_with_simple_parsing_and_write.py 脚本的运行输出

输入文件的所有行被打印到屏幕，并写入到输出文件。大多数情况下，你不需要自输入文件重写文件到输出文件，因为你在输入文件已有所有的数据，但是这个例子是有用的，因为它预示你可以在条件逻辑中用 filewriter.writ 语句来确保你只定入特的行到输出文件中。

Pandas

要用 pandas 处理 CSV 文件，在编辑器中输入如下代码并将文件保存为 pandas_parsing_and_write.py (这个脚本读 CSV 文件，打印它的内容到屏幕，写入内容到输出文件):

```
#!/usr/bin/env python3
import sys
import pandas as pd
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
print(data_frame)
```

```
data_frame.to_csv(output_file, index=False)
```

要运行脚本，输入如下的命令，取决于你的操作系统:

Windows 中:

```
python pandas_parsing_and_write.py supplier_data.csv\
output_files\pandas_output.csv
```

macOS 中:

```
chmod +x pandas_parsing_and_write.py
./pandas_parsing_and_write .py supplier_data.csv\
output_files/pandas_output.csv
```

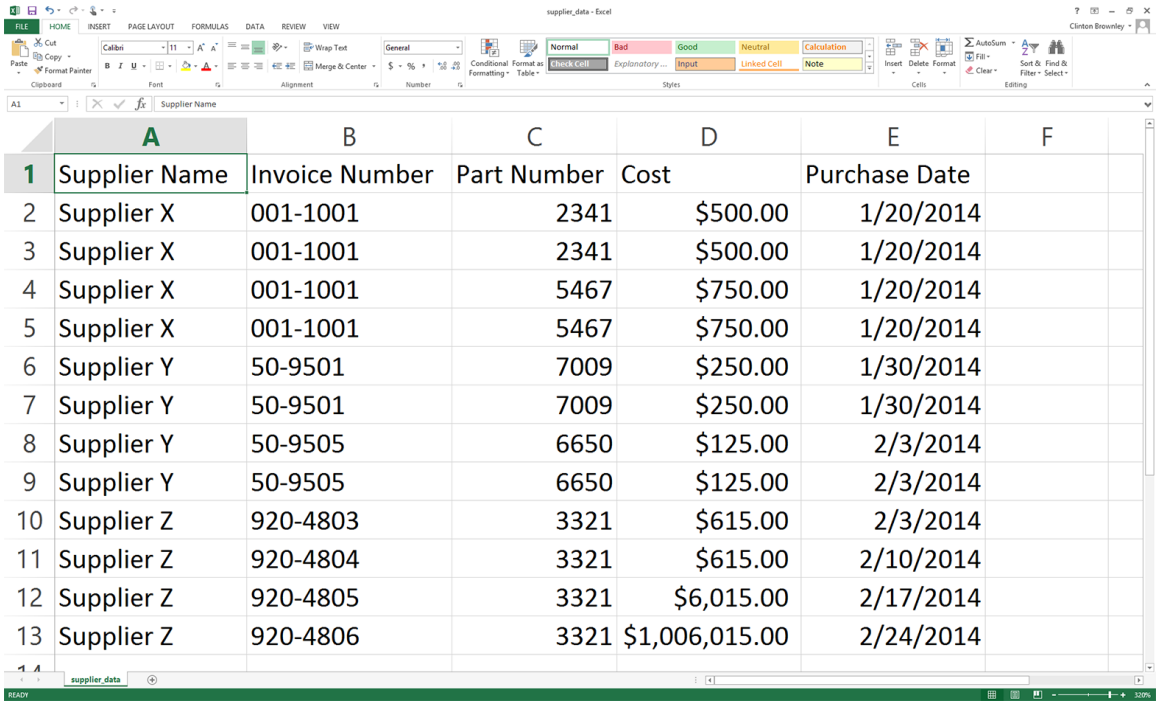
你会注意到 pandas 版本的脚本中，我们创建了变量 data_frame。像列表，字典和元组，DataFrame 是存贮数据的一种方法。它们保存表格形式的数据而不需要将数据解析为列表的列表。DataFrames 是 pandas 包的一部分；你只有导入 padans 后它们才存在。虽然我们称变量为 data_frame，就像用变量列表一样，它在学习阶段很有用，但是你在后面可能需要更多的描述变量名。

脏数据 Dirty Data

现实的世界通常是“脏的”。有些值有时会缺失，手工输入数据可能不正确，传感器记录的数据可能不正确。某些情况下，人们故意记录不正确的数据，因为那是他们的记录工作的一部分。要记住每个人都必须处理“脏”数据，这是数据分析人员的工作，它可能是最没趣也可能是最有趣的工作，但通常是最有趣的工作。

为什么字串的解析会失败

CSV 解析失败的一种原因是列值包含逗号。打开 supplier_data.csv 并在 Cost 列加入两个价格$6,015.00 和 $1,006,015.00。改变之后输入文件看起来像图 2-7.



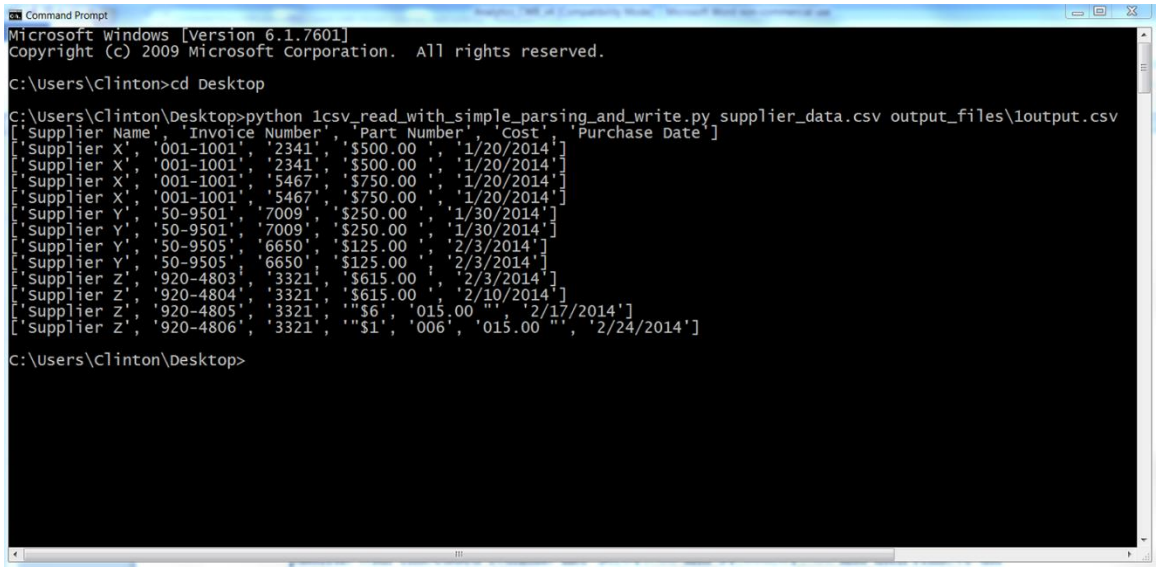
	A	B	C	D	E	F
1	Supplier Name	Invoice Number	Part Number	Cost	Purchase Date	
2	Supplier X	001-1001	2341	$500.00	1/20/2014	
3	Supplier X	001-1001	2341	$500.00	1/20/2014	
4	Supplier X	001-1001	5467	$750.00	1/20/2014	
5	Supplier X	001-1001	5467	$750.00	1/20/2014	
6	Supplier Y	50-9501	7009	$250.00	1/30/2014	
7	Supplier Y	50-9501	7009	$250.00	1/30/2014	
8	Supplier Y	50-9505	6650	$125.00	2/3/2014	
9	Supplier Y	50-9505	6650	$125.00	2/3/2014	
10	Supplier Z	920-4803	3321	$615.00	2/3/2014	
11	Supplier Z	920-4804	3321	$615.00	2/10/2014	
12	Supplier Z	920-4805	3321	$6,015.00	2/17/2014	
13	Supplier Z	920-4806	3321	$1,006,015.00	2/24/2014	

图 2-7. 修改的输入文件（supplier_data.csv）

看一下改变输入文件之后我们的解析脚本如何失败，对修改后的输入文件重新运行脚本。即保存修改，然后用向上箭头恢复之前的命令，重新输入如下命令并按回车键：

```
python 1csv_simple_parsing_and_write.py supplier_data.csv\
output_files\1output.csv
```

你可以看到如图 2-8 所示的打印输出到屏幕。



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Clinton>cd Desktop

C:\Users\Clinton\Desktop>python 1csv_read_with_simple_parsing_and_write.py supplier_data.csv output_files\1output.csv
['Supplier Name', 'Invoice Number', 'Part Number', 'Cost', 'Purchase Date']
['Supplier X', '001-1001', '2341', '$500.00 ', '1/20/2014']
['Supplier X', '001-1001', '2341', '$500.00 ', '1/20/2014']
['Supplier X', '001-1001', '5467', '$750.00 ', '1/20/2014']
['Supplier X', '001-1001', '5467', '$750.00 ', '1/20/2014']
['Supplier Y', '50-9501', '7009', '$250.00 ', '1/30/2014']
['Supplier Y', '50-9501', '7009', '$250.00 ', '1/30/2014']
['Supplier Y', '50-9505', '6650', '$125.00 ', '2/3/2014']
['Supplier Y', '50-9505', '6650', '$125.00 ', '2/3/2014']
['Supplier Z', '920-4803', '3321', '$615.00 ', '2/3/2014']
['Supplier Z', '920-4804', '3321', '$615.00 ', '2/10/2014']
['Supplier Z', '920-4805', '3321', '"$6', '015.00 "', '2/17/2014']
['Supplier Z', '920-4806', '3321', '"$1', '006', '015.00 "', '2/24/2014']

C:\Users\Clinton\Desktop>
```

图 2-8. 对修改后的 supplier_data.csv 文件运行 Python 脚本

如你所见，脚本按逗号解析每一行。脚本处理标题行和前 10 个数据行都正确因为它们不包括嵌入的逗号。但是脚本不能正确的解析最后两行，因为它们包含嵌入的逗号。

有很多办法来改进脚本的代码来处理嵌入逗号的值。例如，我用正则表达式查找嵌入逗号的模式如$6,015.00 和 $1,006,015.00，然后去除值中的逗号，然后分割行，按余下的逗号进行。然而，不是让脚本变得复杂，我们用 python 内置的 csv 模块，它可以处理任意复杂的 CSV 文件。

读写 CSV 文件(Part 2)

基础 Python，使用 csv 模块

使用 python 内置的 csv 模块的优点是它可以正确的处理数据中嵌入逗号和其它复杂的模式。它可以识别这些模式并正确解析，所以你可以花时间来管理数据，进行计算，写输出而不是设计正则表达式和条件逻辑来正确清洗你的数据。我们导入 Python 的内置 csv 模块并用它来处理含有 $6,015.00 和$1,006,015.00 的 CSV 文件。你将学习使用 csv 模块并看到它是如何处理数据内的逗号的。在文本编辑器中输入以下代码并保存文件为 2csv_reader_parsing_and_write.py:

```
1 #!/usr/bin/env python3
2 import csv
3 import sys
4 input_file = sys.argv[1]
5 output_file = sys.argv[2]
6 with open(input_file, 'r', newline='') as csv_in_file:
7 with open(output_file, 'w', newline='') as csv_out_file:
8 filereader = csv.reader(csv_in_file, delimiter=',')
9 filewriter = csv.writer(csv_out_file, delimiter=',')
10 for row_list in filereader:
11 print(row_list)
12 filewriter.writerow(row_list)
```

如你所见，大部分代码与第一个版本本的脚本相似。因此我只讨论不同的行。第 2 行导入 csv 模块，用它的函数来解析输入文件并写到输出文件。第 8 行，即 with 语句下的行，用 csv 模块的 reader 函数创建 reader 对象 filereader，我们用它来读输入文件的行。相似的，第 9 行用 csv 模块的 write 函数来创建 writer 对象称为 filewriter，我们用它来写输出文件。这些函数的第二个参数(如, delimiter=',')是默认是的 delimiter，所以你不需要指明，如果你的输出输入文件是逗号分隔的。我包括 delimiter 参数以便你需要处理有不同 delimiter 的输入或输出文件，如(;)或 tab (\t)。第 12 行用 filewriter 对象的 writerow 函数来写值列表到输出文件。假定输入文件和 Python 脚本在桌面，你不改变命令行窗口或终端窗口的目录，输入下面的命令行并按回车键运行脚本。 (如果你用，你要先运行 chmod 命令以新的脚本以使它可执行):

```
python 2csv_reader_parsing_and_write.py supplier_data.csv\
output_files\2output.csv
```

你可以看到如图 2-9 所示的打印输出到屏幕。

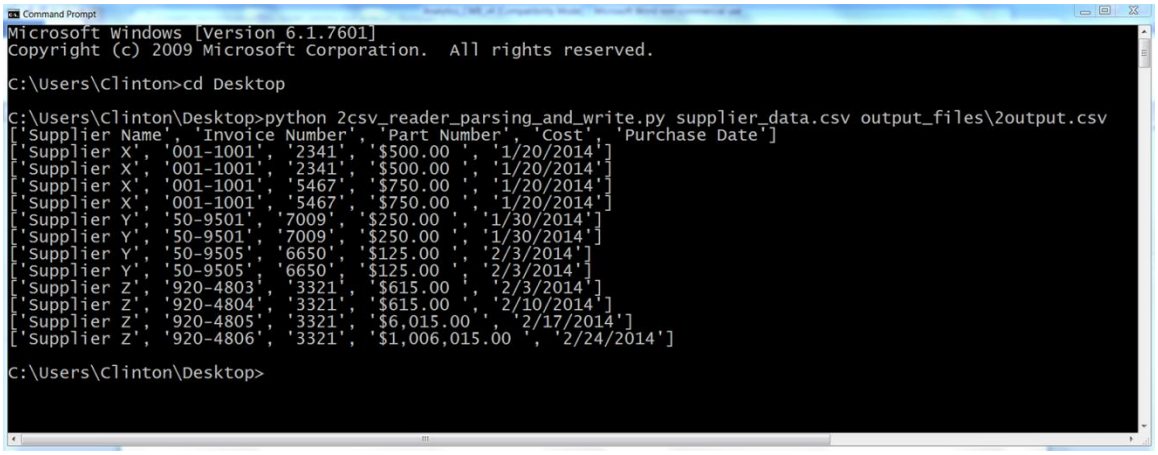


图 2-9. Python 脚本的运行输出

输入文件的所有行被打印到屏幕并写入到输出文件。如你所见，Python 的内置 CSV 模块处理嵌入的逗号并正确的解析每一行到 5 个值的列表中。现以我们知道用 csv 模块来读，处理，写 csv 文件了，我们来学习一下过滤器来指定行和选择定的列以便有效的提取数据。

行过滤器 Filter for Specific Rows

有时候文件包含很多行超出你的需求。例如你只需要包含特定单词或数字的行的子集，或你只需要与物定日期相关的行的子集。这些情况，你可以用 Python 来过滤得到你需要保留的特定的行。你可能习惯了 Excel 手工过滤，但是本章要扩展你的能务以你可以处理 CSV 文件，它太大了 Excel 无打开，而且 CSV 文件的集合手工处理很耗时。下面的子节展示三种不同的方法来过滤输入文件中的特定行:

- 行中的值满足条件
- 行中的值是感兴趣的集合
- 行中的值匹配感兴趣的模式（正则表达式）

你会发现这些子节的代码有一致的结构和格式。我要指出这种共同的结构使你容易识别哪个地方需要修改代码以整合你的商务规则。关注下面的三个子节的以下结构来理解如何过滤输入文件的特定行：

```
for row in filereader:
***if value in row meets some business rule or set of rules:***
do something
else:
do something else
```

这个假代码展示我们将使用过过滤输入文件特定行的共同结构。下面的子节，我们将看到如何修改***内的行来指定商务规则并提取你需要的行。

行中的值满足条件 Value in Row Meets a Condition

基础 Python

有时候你需要保留哪些行中的值满足条件的行。如，你要保留数据集中所有的行其中 cost 大于特定的阀值。或你可能想要所有的行其中采购日期在特定日期之前。这些情况，你可以测试行的值与特定的条件，并过滤到满足条件的行。下面的例子说明如何用两种条件检测行的值并将满足条件的行的子集写入到输出文件。本例中，我们想要保留行的子集，其中供应商的名字为 Supplier Z 或者 cost 大于 $600.00，并将结果写入到输出文件。要过滤到满足条件的行的子集，输入下面的代码到文本编辑器并将文件保存为 3csv_reader_value_meets_condition.py。

```
1 #!/usr/bin/env python3
2 import csv
3 import sys
4 input_file = sys.argv[1]
5 output_file = sys.argv[2]
6 with open(input_file, 'r', newline='') as csv_in_file:
7 with open(output_file, 'w', newline='') as csv_out_file:
8 filereader = csv.reader(csv_in_file)
9 filewriter = csv.writer(csv_out_file)
10 header = next(filereader)
11 filewriter.writerow(header)
12 for row_list in filereader:
13 supplier = str(row_list[0]).strip()
14 cost = str(row_list[3]).strip('$').replace(',', '')
15 if supplier == 'Supplier Z' or float(cost) > 600.0:
16 filewriter.writerow(row_list)
```

第 10 行用 csv 模块的 next 函数读取输入文件的第一行到列表变量 header 中。第 11 行写标题行到输出文件中。第 13 行每一行捕获供应商的名字并赋值给变量 supplier。它用列表索引捕获行的第一个值，row[0]，然后用 str 函数转换值到字串。然后，用 strip 函数从子串的尾部去除空格，tabs,和 newline 符号。最后，将删截的字串赋值给变量 supplier。第 14 行捕获每一行的 cost 并赋值给变量 cost。它用列表索引捕获每一行的第四个值，row[3]，然后用 str 函数转换值到字串。然后，它用 strip 函数去除字串中的美元符号。然后用 replace 函数去除字串中的逗号。最后将结果赋值给变量 cost。第 15 行，创建一个 if 语句，用两个条件检测行中的两个值。特别的，我们想要过滤到供应商名为 Supplier Z 或 cost 大于$600.00 的行。第一个条件，在 if 和 or 之间，检测 supplier 中的值是否为 Supplier Z。第二个条件，在 or 和帽号之间，检测变量 cost 的值，转换成浮点数之后是否大于 600.0。第 16 行，用 filewriter 的 writerow 函数将满足条件的行写入到输出文件。要运行脚本，输入以下命令行并回车：

```
python 3csv_reader_value_meets_condition.py supplier_data.csv\
output_files\3output.csv
```

你看不到屏幕中有任何输出，但是你可以打开输出文件 3output.csv,来检查结果。确保它们是你想要的并试着改为代码选择不同的数据，指明不同的供应商和不同的价格阀值。

Pandas

Pandas 提供了 loc 函数以同时选择特定的行和列。 在逗号之前你指明行过滤条件，在逗号之后你指明列过滤条件。loc 函数内的条件指出我们想要供应商的名字包含 Z 或 Cost 大于 600.0 的行。我们想要所有的列。输入下面的代码到文本编辑器并将文件保存为 pandas_value_meets_condition.py（这个脚本用 pandas 解析 CSV 文件并将满足条件的行写入到输出文件中。）

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
data_frame['Cost'] = data_frame['Cost'].str.strip('$').astype(float)
data_frame_value_meets_condition = data_frame.loc[(data_frame['Supplier Name']\
.str.contains('Z')) | (data_frame['Cost'] > 600.0), :]
data_frame_value_meets_condition.to_csv(output_file, index=False)
```

在命令行中运行脚本，为你的数据提供源和输出文件:

```
python pandas_value_meets_condition.py supplier_data.csv\
output_files\pandas_output.csv
```

你看不到屏幕中有任何输出，但是你可以打开输出文件 pandas_output.csv 检查结果。你可以改变 loc 函数的参数来选择不同的数据。

行中的值在感兴趣的集合中 Value in Row Is in a Set of Interest

基础 Python

有时候你想保留那些行，它们的值在感兴趣的集合内。例如，你想保留数据集中的所有行，其中供应商的名字在集合{Supplier X, Supplier Y}内。（这个花括号是集合标记，不是 python 的字典）。或者你相要所有的行，其中采购日期在集合 {'1/20/14', '1/30/14'}内。这种情况下，你可以检查行的值是否在集合内并过滤到值在集合内的行。下面的例子展示如何用集合的成员检查行的值并将在集合内的行的子集写到输出文件。本例，我们想保留行的子集，其中采购日期在集合{'1/20/14', '1/30/14'}内，并将结果写入输出文件。要过滤到行的值在这个集合内的行的子集，在文本编辑器内写入以下代码并将文件保存为 4csv_reader_value_in_set.py。

```
1 #!/usr/bin/env python3
2 import csv
3 import sys
4 input_file = sys.argv[1]
5 output_file = sys.argv[2]
```

```
important_dates = ['1/20/14', '1/30/14']
with open(input_file, 'r', newline='') as csv_in_file:
with open(output_file, 'w', newline='') as csv_out_file:
filereader = csv.reader(csv_in_file)
filewriter = csv.writer(csv_out_file)
header = next(filereader)
filewriter.writerow(header)
for row_list in filereader:
a_date = row_list[4]
if a_date in important_dates:
filewriter.writerow(row_list)
```

第 6 行创建名为 important_dates 的列表变量，它包含两个感兴趣的日期。这个变量定义我们的集合。创建包含你感兴趣的值的变量然后在代码中引用变量是有帮助的。这样，如查感兴趣的值改变，你只需要改变一个位置（例如，你定义变量的地方），变种变化可以在整个代码中传递，只要你想引用这个变量。第 14 捕获每一行的采购日期并赋值给名为 a_date 的变量。你从行表的索引，row[4],可以看到，采购日期为第四列。 第 15 行创建 if 语句来检测变名为 a_date 的采购日期是否在定义的兴趣集合 important_dates 内。如果值在集合内，下一行代码将行写入到输出文件。在命令行中运行这个脚本：

```
python 4csv_reader_value_in_set.py supplier_data.csv output_files/4output.csv
```

然后你打开输出文件 4output.csv 检查结果。

Pandas

要用 pandas 来过滤到值在感兴趣的集合中的行，输入以下代码到文本编辑器并将文件保存为 pandas_value_in_set.py （这个脚本解析 CSV 文件并将值在感兴趣集合内的行写入输出文件）：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
important_dates = ['1/20/14', '1/30/14']
data_frame_value_in_set = data_frame.loc[data_frame['Purchase Date'].\
isin(important_dates), :]
data_frame_value_in_set.to_csv(output_file, index=False)
```

这里的新的重要的命令是 isin。如前所述，我们从命令行运行脚本，提供源文件和输出文件名：

```
python pandas_value_in_set.py supplier_data.csv output_files\pandas_output.csv
```

然后你打开输出文件，pandas_output.csv，检查结果。

行中的值匹配模式或正则表达式 Value in Row Matches a Pattern/Regular Expression

基础 Python

有时候你想保留行的子集，其中行中的值匹配或包含特的模式（如，正则表达式）。例如你想要保留数据集中 invoice number 以“001-”开始的所有行或你想要供应商名包含“Y”的所有行。这种情况，你可以检查行中的值是否匹配或包含特定的模式并过滤到那些行。下面的例子展示如何用物定的模式检查值并将匹配模式的行的子集写入输出文件。本例中，我们想要保留行的子集，其中 invoice number 以“001-”开始，并将结果写入输出文件。要过滤到匹配这种模式的行的子集，输入下面的代码到文件编辑器并将文件保存为 5csv_reader_value_matches_pattern.py:

```
1 #!/usr/bin/env python3
2 import csv
3 import re
4 import sys
5 input_file = sys.argv[1]
6 output_file = sys.argv[2]
7 pattern = re.compile(r'(?P<my_pattern_group>^001-.*)', re.I)
8 with open(input_file, 'r', newline='') as csv_in_file:
9 with open(output_file, 'w', newline='') as csv_out_file:
10 filereader = csv.reader(csv_in_file)
11 filewriter = csv.writer(csv_out_file)
12 header = next(filereader)
13 filewriter.writerow(header)
14 for row_list in filereader:
15 invoice_number = row_list[1]
16 if pattern.search(invoice_number):
17 filewriter.writerow(row_list)
```

第 3 行导入正则表达式模块 (re)使我们可以访问 re 模块内的函数。第 7 行用 re 模块的 compile 函数创建名为 pattern 的正则表达式变量。如果你读过第一章，那么这个函数的内容相似。r 是说，考虑单引号内的模式作为原始字串。 ?P<my_pattern_group> metacharacter 捕获 <my_pattern_group>组中匹配的子串，以便必要时它们可以被打印到屏幕或写到文件。 我们查找的真实模式是^001-.*。caret 是特殊符号，是说只查找以字串开前的模式。所以，过滤的字串需要以“001-”开始。句号匹配任何除 newline 外的字串。所以任何除 newline 外的符号可以接“001-”。最后，*说重复前面的符号限定 0 次或多次。 .*的组合是说除 newline 外的符号可以在“001-”后出多次。更正式一些，“字串可以在‘-’后包含任何东西只要字串以‘001-’开始，它就匹配正则表达式。”最后 re.I 参数指示正则表达式进行大小写敏感匹配。本例中这个参数不太重要，因为模式是数字的，但是它说明可以在哪里加入参数，如果你的模式包含字符并且你要进行大小写敏感的匹配。第 15 行用列表索引提取行的 invoice number 并赋值给变量

invoice_number。下一行，我们看一下变量的模式。第 16 行用 re 模块的 search 函数查找贮存于 invoice_number 中的模式。如果模式出现在 invoice_number 的值中，则第 17 行将行写入输出文件。要运行脚本，输入下面的命令行并回车：

```
python 5csv_reader_value_matches_pattern.py supplier_data.csv\
output_files\5output.csv
```

你可以打开输出文件，5output.csv，检查结果。

Pandas

在 pandas 中过滤匹配模式的值的行，输入如下代码到文本编辑器中，将文件保存为 pandas_value_matches_pattern.py（这个脚本读取 CSV 文件，打印符合模式的值的行到屏幕，将行写入输出文件）：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
data_frame_value_matches_pattern = data_frame.loc[data_frame['Invoice Number'].\
str.startswith("001-"), :]
data_frame_value_matches_pattern.to_csv(output_file, index=False)
```

使用 pandas，我们可以用 startswith 来找到数据而不是麻人的正则表达式。要运行脚本，在命令行中输入以下并回车：

```
python pandas_value_matches_pattern.py supplier_data.csv\
output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv 检查结果。

选择特定列 Select Specific Columns

有时候文件包含很多你不想要的列。这种情况下，你可以用 Python 来选择你需要的列。有两种方法来选择 CSV 文件中的指定列。下一节说明这两种方法：

- 用列索引值
- 用列标题

列索引 Column Index Values

基础 Python

选择 CSV 文件中的特定列的一种方法是用你想要保留的列的索引。当你想要的列的索引易于识别时这种方法有效，或当你要处理多个输入文件，其中列的位置一致时（如，不改变）。例如，你只需要保留第一列和最后一列数据，你可以用 row[0]和 row[-1]将第一行和最后一行写到

文件中。本例中，我们只想保留 Supplier Name 和 Cost 列。要用索引值选择这两列，输入下面的代码到文本编辑器并将文件保存为 6csv_reader_column_by_index.py:

```
1 #!/usr/bin/env python3
2 import csv
3 import sys
4 input_file = sys.argv[1]
5 output_file = sys.argv[2]
6 my_columns = [0, 3]
7 with open(input_file, 'r', newline='') as csv_in_file:
8 with open(output_file, 'w', newline='') as csv_out_file:
9 filereader = csv.reader(csv_in_file)
10 filewriter = csv.writer(csv_out_file)
11 for row_list in filereader:
12 row_list_output = [ ]
13 for index_value in my_columns:
14 row_list_output.append(row_list[index_value])
15 filewriter.writerow(row_list_output)
```

第 6 行创建名为 my_columns 的列表变量，它包含我们想要保留的两列的索引值。本例中，这两个索引值对应 Supplier Name 和 Cost 列。 再次，创建包含你感兴趣的索引值的变量然后在代码中引用变量是有帮助的。这样，如果感兴趣的索引值改变你就只需要修改一个位置（如，你定义 my_columns 的地方），然后修改可以在整个代码中传递只要你引用 my_columns。第 12 到 15 在 for 循环的外面，它们为输入文件的每一行运行。第 12 行创建空的列表变量 row_list_outPut。这个变理贮存我们想要的每一行的值。第 13 行是 for 循环，遍历我们感兴趣的 my_columns 的索引值。第 14 行用 append 函数将 my_columns 定义的索引值的行中的 row_list_output 填写。合在一起，这三行代码创建了列表，它包含我们想要写入输出文件的每一行的值。创建列表是有用的，因为 filewriter 的 writerow 方法希望用一序列的字串或数字，而我们的列表变量 row_list_output 是一序列的字串。第 15 行将 row_list_output 的值写入输出文件。再次，脚本按输入文件的行执行代码。为了确保操作序列清楚，我们检查一下 for 循环的外面发生了什么。本例中，我们操作输入文件的第一行（如 header 行）。第 12 行创建空的列表变量 variable row_list_output。第 13 行是 for 循环，它遍历 my_columns 的值。第一次循环，index_value 等于 0，所以第 14 行，append 函数推入 row[0]（如字串 Supplier Name）到 row_list_output。接着，代码返回第 13 行 for 循环，但这次，index_value 等于 3。因为 index_value 等于 3，第 14 行 append 函数推入 row[3]（如字串 Cost）到 row_list_output。my_columns 中没有别的值，所以第 13 行的 for 循环终止，代码运行到第 15 行。第 15 行写 row_list_output 的列表值到输出文件。接着，代码返回 for 循环的外部，第 11 行，开始处理输入文件的下一行。

要运行脚本，在命令行中输入如下并回车:

```
python 6csv_reader_column_by_index.py supplier_data.csv output_files\6output.csv
```

你可以打开输出文件，6output.csv，检查结果。

Pandas

要用 pandas 来基于索引选择择列，在文本编辑器中加入以下代码并将文件保存为 pandas_column_by_index.py （这个脚本读 CSV 文件,打印索引值为 0 和 3 的列到屏幕，并将同样的列写入到输出文件)：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
data_frame_column_by_index = data_frame.iloc[:, [0, 3]]
data_frame_column_by_index.to_csv(output_file, index=False)
```

Here, we’re using the iloc command to select columns based on their index position. Run the script at the command line:

```
python pandas_column_by_index.py supplier_data.csv\
output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv 检查结果。

列标题 Column Headings

基础 Python

第二种选择 CSV 文件的特定列的方法是用列标题而不是索引位置。当你要保留的列的标题易于识别，或当你处理多个文件，列的位置变化但是标题不变时，这种方法有效。

例如，假如我们只想保留 Invoice Number 和 Purchase Date 列。要通过列标题选择这两列，在文本编辑器中输入如下代码并将文件保存为 7csv_reader_column_by_name.py:

```
1 #!/usr/bin/env python3
2 import csv
3 import sys
4 input_file = sys.argv[1]
5 output_file = sys.argv[2]
6 my_columns = ['Invoice Number', 'Purchase Date']
7 my_columns_index = []
8 with open(input_file, 'r', newline='') as csv_in_file:
9 with open(output_file, 'w', newline='') as csv_out_file:
10 filereader = csv.reader(csv_in_file)
11 filewriter = csv.writer(csv_out_file)
12 header = next(filereader, None)
13 for index_value in range(len(header)):
14 if header[index_value] in my_columns:
15 my_columns_index.append(index_value)
```

```
16 filewriter.writerow(my_columns)
17 for row_list in filereader:
18 row_list_output = [ ]
19 for index_value in my_columns_index:
20 row_list_output.append(row_list[index_value])
21 filewriter.writerow(row_list_output)
```

本例中的代码比前面的代码长一些，但是看起来很熟悉。代码更多的原因是我们先要处理标题行，分别识感兴趣的列标题的索引。然后用索引值保留每一行的值，其索引与我们想要保留的列标题索引相同。第 6 行创建列表变量 my_columns，它包含两个字串，是我们想要保留的两列的名称。第 7 行创建空的列表变量 my_columns_index，我们要用感兴趣的两列的索引来填写。第 12 行用 filereader 对象的 next 函数读输入文件的第一行到列表变量 header 中。第 13 行用列标题的索引初始化 for 循环。第 14 行用 if 语句和列表索引检查每列标题是否在 my_columns 里。例如，第一次 for 循环，index_value 等于 0，所以 if 语句检查 header[0]（如，第一列标，SupplierName）是否在 my_columns 里。因为 Supplier Name 不在 my_columns 里，第 15 行不执行这个值。代码返回第 13 行的 for 循环，这次 index_value 等于 1。接着，第 14 行的 if 语句检查 header[1]（如，第二列标题，Invoice Number）是否在 my_columns 里。因为 Invoice Number 在 my_col Umns 里，第 15 行执行，这一列的索引值推入到列表 my_columns_index。for 循环继续，最后将 Purchase Date 列的索引值推入 my_columns_index。一旦 for 循环终止，第 16 行写 my_columns_index 的两个字串到输出文件。 第 18 到 21 行操作输入文件的余下的行。第 18 行创建空的列表 row_list_output 来贮存我们想要保存的每一行的值。第 19 行 for 循环遍历 my_columns_index 的索引值，第 20 行添加有这些索引值的值到 row_list_output。最后第 21 行将 row_list_output 的值写入输出文件。

在命令行中运行代码：

```
python 7csv_reader_column_by_name.py supplier_data.csv output_files\7output.csv
```

你可以打开输出文件 7output.csv，检查结查。

Pandas

要用 pandas 来基于标题选择列，在文本编辑器中加入以下代码并将文件保存为 pandas_column_by_name.py（这个脚本读 CSV 文件，打印 Invoice Number 和 Purchase Date 列到屏幕，并将相同的列写入输出文件）：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
data_frame_column_by_name = data_frame.loc[:, ['Invoice Number', 'Purchase Date']]
data_frame_column_by_name.to_csv(output_file, index=False)
```

再一次，我们用 loc 命令选择列，这一次用的是列标题。运行脚本：

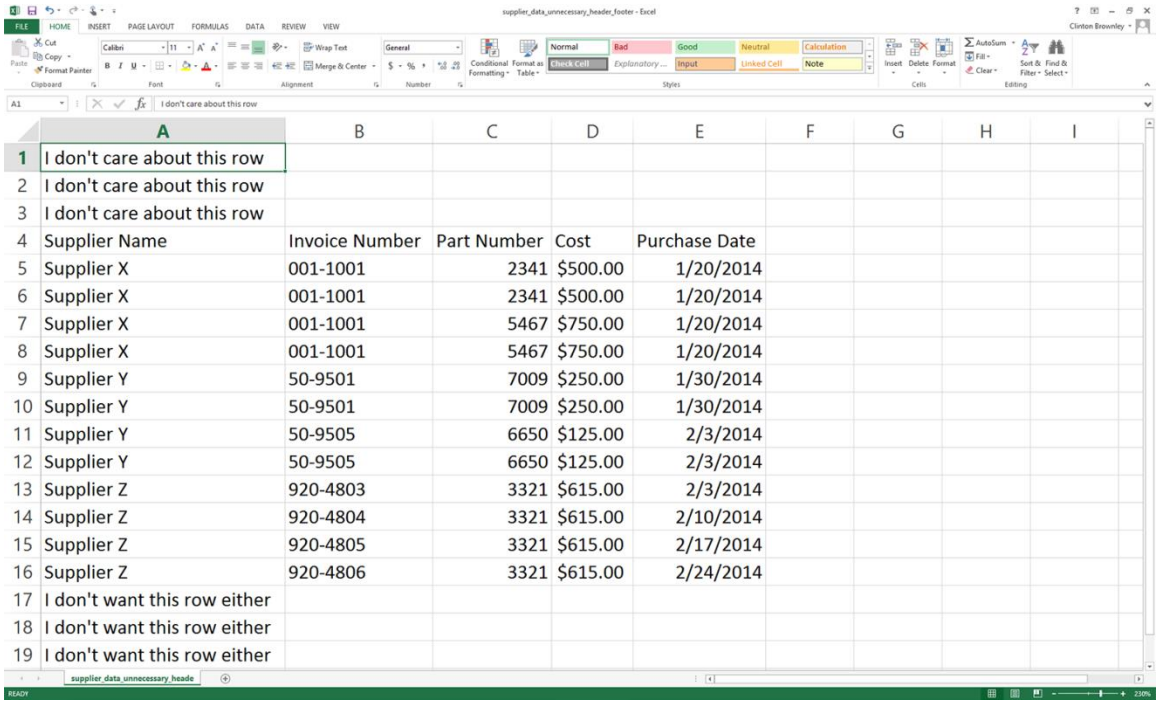
```
python pandas_column_by_name.py supplier_data.csv output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv,检查结果。

选择连续的行

有时候在文件的顶部或底部的内容你不想处理。例如，有很多的标题目和作者行在文件的顶部，或者资源，假定，警告，注意等列于文件的底部。很多时候，我们不想处理这些内容。为了说明如何选择 CSV 文件的连续的行，我们需要修改我们的输入文件。

1. 用电子表格打开 supplier_data.csv。
2. 在文件的顶部标题行上面插入三行。写入一些文本如 “I don’t care about this line” 在 A1:A3 单元格。
3. 在文件下部，最后一行下面添加三行。添加一些文本如 “I don’t want this line either” 在列 A 的三个单元格中，在最后一行下面。
4. 保存文件 supplier_data_unnecessary_header_footer.csv. 它看起来像图 2-10



	A	B	C	D	E
1	I don't care about this row				
2	I don't care about this row				
3	I don't care about this row				
4	Supplier Name	Invoice Number	Part Number	Cost	Purchase Date
5	Supplier X	001-1001	2341	$500.00	1/20/2014
6	Supplier X	001-1001	2341	$500.00	1/20/2014
7	Supplier X	001-1001	5467	$750.00	1/20/2014
8	Supplier X	001-1001	5467	$750.00	1/20/2014
9	Supplier Y	50-9501	7009	$250.00	1/30/2014
10	Supplier Y	50-9501	7009	$250.00	1/30/2014
11	Supplier Y	50-9505	6650	$125.00	2/3/2014
12	Supplier Y	50-9505	6650	$125.00	2/3/2014
13	Supplier Z	920-4803	3321	$615.00	2/3/2014
14	Supplier Z	920-4804	3321	$615.00	2/10/2014
15	Supplier Z	920-4805	3321	$615.00	2/17/2014
16	Supplier Z	920-4806	3321	$615.00	2/24/2014
17	I don't want this row either				
18	I don't want this row either				
19	I don't want this row either				

图 2-10. 你想要的行上面和下面有额外的行的 CSV 文件

现在输入文件有不想要的头部和脚部信息，我们修改一下我们的 Python 脚本以致不会读取这些行。

基础 Python

要用基础 python 选择特定的行，我们用 row_counter 变量追踪行数，以致我们可以识别和选择我们想要保留的行。从前面的列子，我们已经知道我们想要保留的数据有 13 行。你可以从下面的 if 块中看到我们想要写入输出文件的行是索引大于等于 3 和小于等于 15。

要用基础 python 选择这些行，在文本文件中输入以下代码将保存为 11csv_reader_select_contiguous_rows.py:

```
#!/usr/bin/env python3
import csv
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
row_counter = 0
with open(input_file, 'r', newline='') as csv_in_file:
with open(output_file, 'w', newline='') as csv_out_file:
filereader = csv.reader(csv_in_file)
filewriter = csv.writer(csv_out_file)
for row in filereader:
if row_counter >= 3 and row_counter <= 15:
filewriter.writerow([value.strip() for value in row])
row_counter += 1

```

我们用 row_counter 变量配合 if 语句来保留我们关注的行并跳过不想要的头部和脚部内容。对于输入文件的前三行，row_counter 小于 3，所以 if 语句不执行，并且 row_counter 的值加 1。对于输入文件的后三行，row_counter 大于 15，所以 if 块不执行，row_counter 的值加 1。我们要保留的行在不想要的头部和脚部之间。对于这些行， row_counter 的范围为 3-15。if 块处理这些行并写入输出文件。我们用 string 模块的 strip 函数在列表推导式中去除每一行前后的空格、tabs 和 newline 符号。你可以通过 print 语句看到 row_counter 变量和行的内容，如 print(row_counter, [value.strip() for value in row]) ，在已有的 writerow 语句前。

要运行脚本，在命令行中输入如下并回车。

```
python 11csv_reader_select_contiguous_rows.py supplier_data_unnecessary_header_\
footer.csv output_files\11output.csv
```

你可以打开输出文件 11output.csv，检查结果。

Pandas

Pandas 提供了 drop 函数来去除行或列，基于行索引或列标题。 下面的脚本中，drop 函数去除输入文件的前三行和后三行(如索引为 0, 1, 和 2 以及 16, 17,和 18 的行)。 Pandas 也提供了通用的 iloc 函数，我们可以用来基于索引选择一行并将它作为列索引。最后，按提供了 reindex 函数，我们可以一个或多个轴变为新的索引。

要用 pandas 保留标题行和数据行并去除不要的头部和脚部，在文本文件中加入以下代码并将文件保存为 pandas_select_contiguous_rows.py:

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
```

```
data_frame = pd.read_csv(input_file, header=None)
data_frame = data_frame.drop([0,1,2,16,17,18])
data_frame.columns = data_frame.iloc[0]
data_frame = data_frame.reindex(data_frame.index.drop(3))
data_frame.to_csv(output_file, index=False)
```

要运行脚本，你在命令行窗口中输入如下并回车：

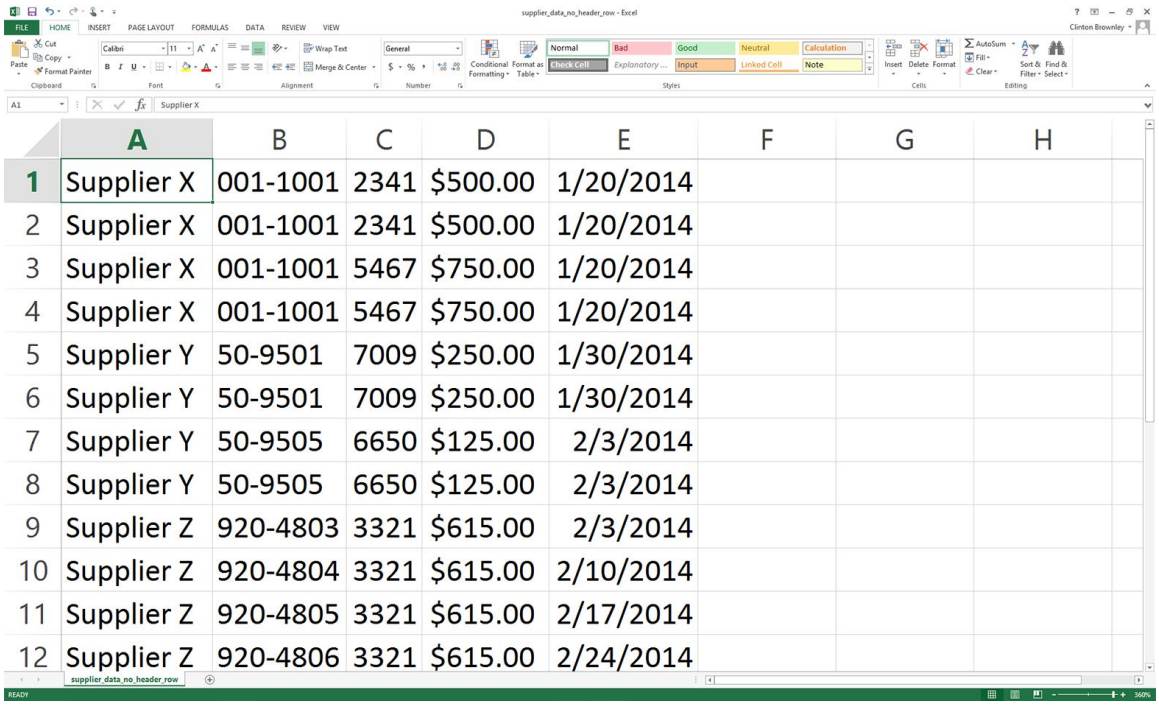
```
python pandas_select_contiguous_rows.py supplier_data_unnecessary_header_\
footer.csv output_files\pandas_output.csv
```

然后你打开输出文件 pandas_output.csv,检查结果。

添加标题行 Add a Header Row

有时候电子表格并没有标题行，即便你想要每列有列标题。这种情况，你可以用脚本增加列标题。为了展示如何用脚本增加列标题，我们需要修改输入文件：

1. 用电子表格打开 supplier_data.csv 文件。
2. 删除文件的第一行(如含有列标题目的标题行)。
3. 将文件保存为 supplier_data_no_header_row.csv。它看起来像图 2-11。



	A	B	C	D	E	F	G	H
1	Supplier X	001-1001	2341	$500.00	1/20/2014			
2	Supplier X	001-1001	2341	$500.00	1/20/2014			
3	Supplier X	001-1001	5467	$750.00	1/20/2014			
4	Supplier X	001-1001	5467	$750.00	1/20/2014			
5	Supplier Y	50-9501	7009	$250.00	1/30/2014			
6	Supplier Y	50-9501	7009	$250.00	1/30/2014			
7	Supplier Y	50-9505	6650	$125.00	2/3/2014			
8	Supplier Y	50-9505	6650	$125.00	2/3/2014			
9	Supplier Z	920-4803	3321	$615.00	2/3/2014			
10	Supplier Z	920-4804	3321	$615.00	2/10/2014			
11	Supplier Z	920-4805	3321	$615.00	2/17/2014			
12	Supplier Z	920-4806	3321	$615.00	2/24/2014			

图 2-11. 含数据行但没有标题行的 CSV 文件

基础 Python

要用基础 python 增加列标题，在文本文件中增加以下代码并将文件保存为 12csv_reader_add_header_row.py:

```
1 #!/usr/bin/env python3
```

```
2 import csv
3 import sys
4 input_file = sys.argv[1]
5 output_file = sys.argv[2]
6 with open(input_file, 'r', newline='') as csv_in_file:
7 with open(output_file, 'w', newline='') as csv_out_file:
8 filereader = csv.reader(csv_in_file)
9 filewriter = csv.writer(csv_out_file)
10 header_list = ['Supplier Name', 'Invoice Number',\
11 'Part Number', 'Cost', 'Purchase Date']
12 filewriter.writerow(header_list)
13 for row in filereader:
14 filewriter.writerow(row)
```

第 10 行创建列表变量 header_list，含有 5 个字串值作为列标题。第 12 行将列表的值作为输出文件的第一行写入。相似地，第 14 行写入所有的数据行到输出文件中，在标题行下。

要运行脚本，在命令行中输入以下并回车：

```
python 12csv_reader_add_header_row.py supplier_data_no_header_row.csv\
output_files\12output.csv
```

你可以打开文件 12output.csv 检查结果。

Pandas

Pandas 的 read_csv 函数可以直接显示输入文件没有标题行，并提供列标题列表。要增加标题行到原来没有标题行的数据集中，在文本文件中输入代码并将文件保存为 pandas_add_header_row.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
header_list = ['Supplier Name', 'Invoice Number',\
'Part Number', 'Cost', 'Purchase Date']
data_frame = pd.read_csv(input_file, header=None, names=header_list)
data_frame.to_csv(output_file, index=False)
```

要运行脚本，在命令行中输入如下并回车：

```
python pandas_add_header_row.py supplier_data_no_header_row.csv\
output_files\pandas_output.csv
```

你可以打开文件 pandas_output.csv，检查结果。

读多个 CSV 文件

到此我已经展示了如何处理单一个 CSV 文件。有时候，你只想处理一个文件，这种情况，前面的例子为你自动处理文件提供了想法。虽然你只想处理一个文件，但是文件太大，无法手工处理，程序处理可以减少人为错误的机率如复制粘贴的错误和书写错误。然而，许多情况下，你要处理很多个文件，手工处理效率太低或者不可能手工处理。这种情况，Python 更好，因为它让你自动化和放大数据处理过程。本节介绍 python 内置的 glog 模块并构建一些例子展示如何放大处理多个文件。为了处理多个 CSV 文件，我们需要创建多个 CSV 文件。我们创建三个文件用于以下的例子，但是要记住这里展示的技术可以放大到许多的文件，因为你的计算机可以处理成百上千个 CSV 文件。

CSV file #1

1. 打开电子表格。
2. 如图 2-12 所示添加数据。
3. 将文件保存为 sales_january_2014.csv.

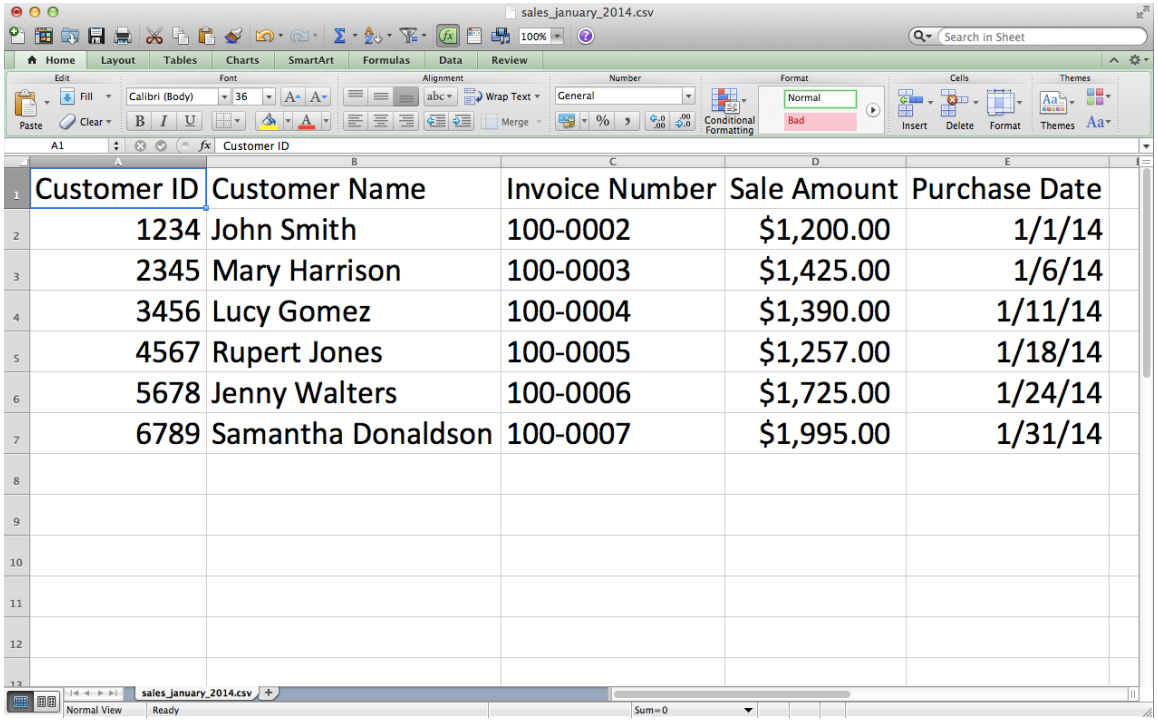
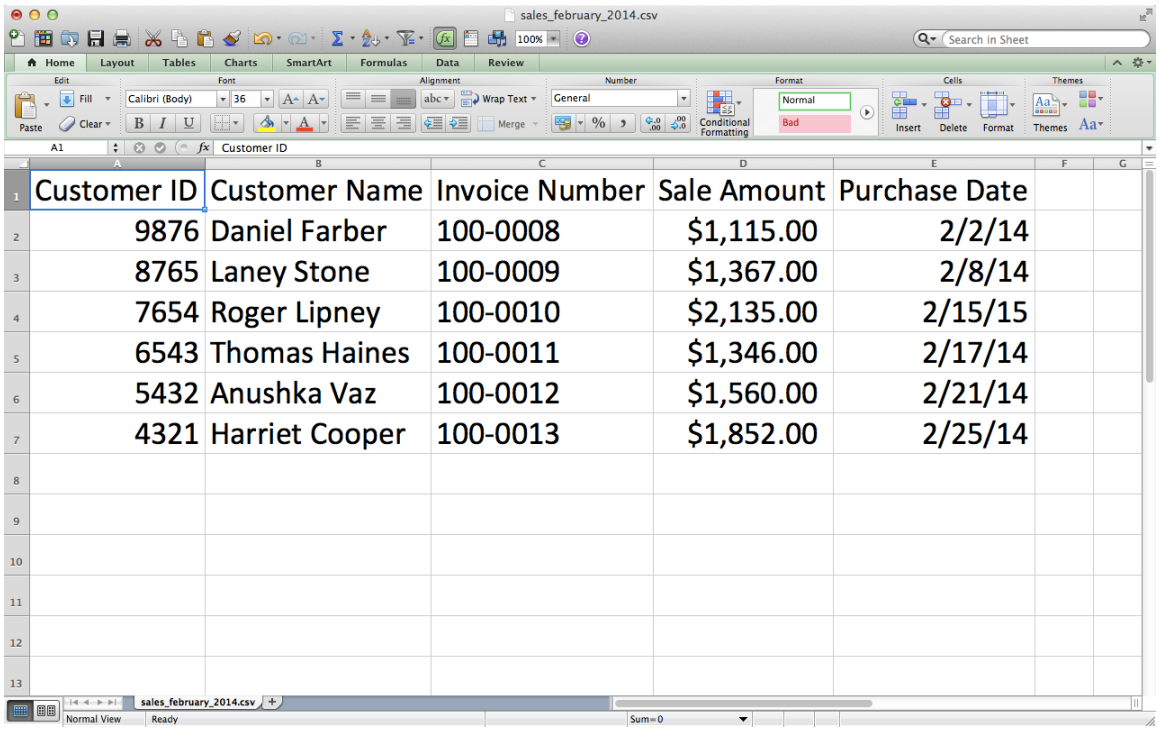


图 2-12. CSV file #1: sales_january_2014.csv

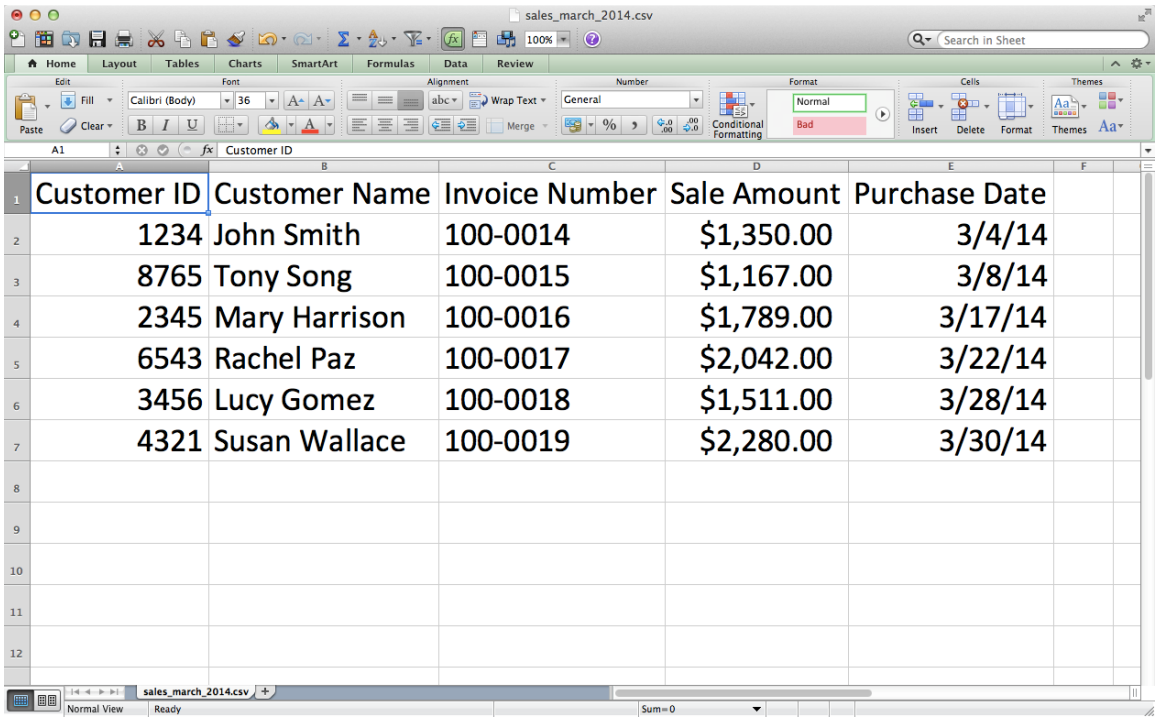
CSV file #2

1. 打开电子表格。
2. 如图 2-13 所示添加数据。
3. 将文件保存为 sales_february_2014.csv.



CSV file #3

1. 打开电子表格。
2. 如图 2-14 所示添加数据。
3. 将文件保存为 sales_march_2014.csv.



计算文件数目并计算每个文件的行数和列数

我们从简单的计算行或列数开始，这很基础，但是它是了解数据集的很好的方法。有时候，你想知道你在处量的文件的内容，有时某人给你一些文件你并不想马上知道内容。有时候计算你在处理文件的数目以及每个文件的行数和列数是有帮助的。要处理这些 CSV 文件，在文本编辑器中输入如下代码并将文件保存为 8csv_reader_counts_for_multiple_files.py:

```
1 #!/usr/bin/env python3
2 import csv
3 import glob
4 import os
5 import sys
6 input_path = sys.argv[1]
7 file_counter = 0
8 for input_file in glob.glob(os.path.join(input_path,'sales_*')):
9 row_counter = 1
10 with open(input_file, 'r', newline='') as csv_in_file:
11 filereader = csv.reader(csv_in_file)
12 header = next(filereader, None)
13 for row in filereader:
14 row_counter += 1
15 print('{0!s}: \t{1:d} rows \t{2:d} columns'.format(\
16 os.path.basename(input_file), row_counter, len(header)))
17 file_counter += 1
18 print('Number of files: {0:d}'.format(file_counter))
```

第 3 和第 4 行导入 Python 内置的 glob 和 os 模块，以便我们可以用它们的函数来列出和解析我们要处理的文件的路径。glob 模块定位所有的路径名，匹配特定的模式。模式可以含 Unix shell－风格的通配符如*。在这个例子里，我们要查找的模式为'sales_*'。这个模式意味着我们要查找所有的文件名以 sales_开始，后面可以有任意字符。因为你创建了 3 个输入文件，你知道我们要用这个代码来识别三个输入文件，它们的名字都以 sales_开头，后面接不同的月。以后，你可能想查找目录中的所有 CSV 文件而不仅是以 sales_开头的文件。你可以将脚本上的模式由'sales_*' 改为 '*.csv'。因为'.csv'是所有 CSV 文件名的结尾，这种模式可以有效的找到所有的 CSV 文件。os 模块包含有用的函数来解析路径名。例如，os.path.basename(path)返回 path 的基名。所以，如果 path 是 C:\Users\Clinton\Desktop\my_input_file.csv，则 os.path.basename(path) 返回 my_input_file.csv。第 8 行是多个输入文件放大数据处理的关键。第 8 行创建 for 循环遍历输入文件集，使用 glob 和 os 模块的函数创建输入文件的列表以进行处理。这一行代码有很多东西。我们来彻底看一下。 os 模块的 os.path.join() 函数合并函数的括号内的两个成分。input_path 是包含输入文件的目录路径，'sales_*'指任何以'sales_'开头的文件。glob 模块的 glob.glob()函数扩展'sales_*' 中的星号(*)到实际的文件名中。本例中，glob.glob() 和 os.path.join() 创建三个文件的列表。

```
['C:\Users\Clinton\Desktop\sales_january_2014.csv',
```

'C:\Users\Clinton\Desktop\sales_february_2014.csv',

'C:\Users\Clinton\Desktop\sales_march_2014.csv']

然后，for 循环为输入文件的列表执行行首缩进的代码。第 15 行是打印语句，打印文件名，文件中的行数，列数。打印语句中的 tab 符号，\t,不是必须的，但是有助于对齐三列，通过将 tab 放在列之间。这一行使用{}符号传递三个值到打印语句。对于第一个值，我们用 os.path.basename()函数提取完整路径的最后成份。第二个值，我们用 row_counter 变量计算输入文件的行数。最后第三个值，我们用内置的 len 函数计算列表变量 header 的值，它包含输入文件的列标题的列表。我们用这个值作为每一个输入文件的列数目。最后，第 15 行打印每个文件的信息，第 17 行用 file_counter 的值显示文件的数目。

要运行脚本，在命令行中输入如下 命令并回车:

```
python 8csv_reader_counts_for_multiple_files.py "C:\Users\Clinton\Desktop"
```

注意，命令行中脚本名后是目录路径。前面的例子中，那个位置输入的是文件名。这种情况，我们想要处理多个文件，所以我们要指出包含所有文件的目录。

你可以看到三个输入文件的名称和每个文件的行数以及列数打印到屏幕上。在行信息下面最后的打印语句显示输入文件的总数目。显示的信息如图 2-15。

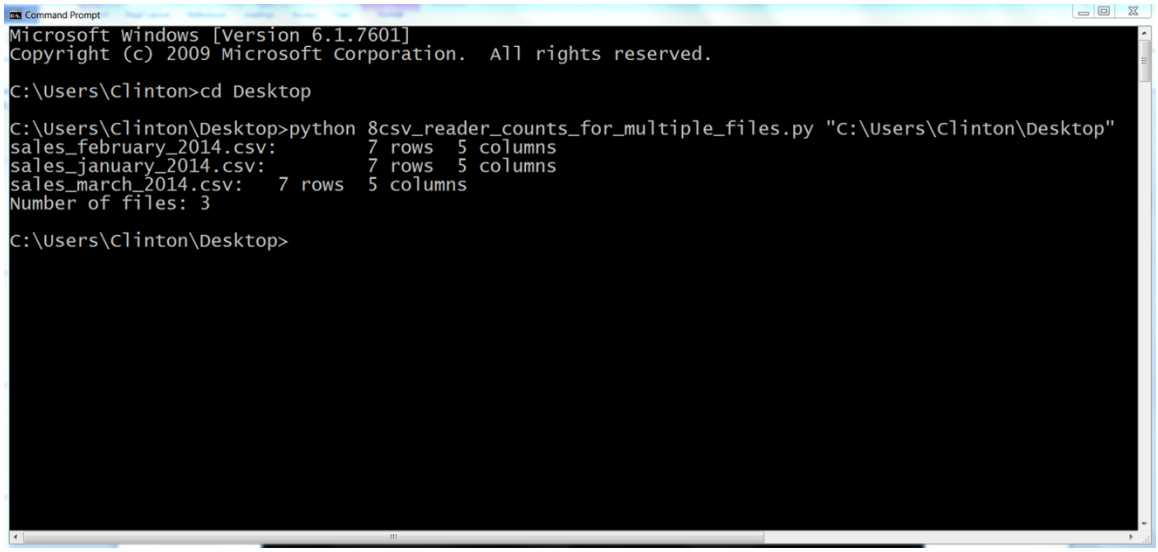


图 2-15. Python 脚本的输出:三个 CSV 文件的行数和列数

输出显示脚本处理三个文件，每个文件有 7 行和 5 列。本例说明如何读多个 CSV 文件并打印一些关于文件的基本信息到屏幕。当你不熟悉你想处理的文件时，打印你想要处理的文件的基本信息是很有用的。了解你要处理的文件数目以及每个文件中的行数和列数可以给你一些关于工作量的信息，以及文件的布局是否一致。

从多个文件拼接数据

当你有多个文件包含相似的数据时，你通常想拼接数据，使所有的数据在一个文件中。你以前通过打开文件，复制和粘贴各个表格中的数据到一个表格中。这种手工处理耗时还易于出

错。而且，有时候文件太大，手工处理不太可能。因为手工处理的缺点，我们看一下如何用 python 来完成任务。我们用创建的三个 CSV 文件来说明如何从多个文件拼接数据。

基础 Python

要用基础 python 来从多个输入文件垂直拼接数据到一个文件，在文本编辑器中输入以下代码并将文件保存为 9csv_reader_concat_rows_from_multiple_files.py:

```
1 #!/usr/bin/env python3
2 import csv
3 import glob
4 import os
5 import sys
6 input_path = sys.argv[1]
7 output_file = sys.argv[2]
8
9 first_file = True
10 for input_file in glob.glob(os.path.join(input_path,'sales_*')):
11 print(os.path.basename(input_file))
12 with open(input_file, 'r', newline='') as csv_in_file:
13 with open(output_file, 'a', newline='') as csv_out_file:
14 filereader = csv.reader(csv_in_file)
15 filewriter = csv.writer(csv_out_file)
16 if first_file:
17 for row in filereader:
18 filewriter.writerow(row)
19 first_file = False
20 else:
21 header = next(filereader, None)
22 for row in filereader:
23 filewriter.writerow(row)
```

第 13 行是个 with 语句，打开输出文件。早期的例子包括写入到输出文件，open 函数中的字串为'w'，意思为以写模式打开输出文件。本例中，我们用字母'a' 而不是 'w'，即以 append 模式打开输出文件。我们想要用 append 模式，使每一个输入文件的数据添加到输出文件。如果我们用写模式，每一个输入文件的数据会覆盖以前的输出文件中的数据。第 16 行的 if-else 语句依靠第 9 行创建的 first_file 变量来区别第一个输入文件和所有的后面的输入文件。我们这样区别输入文件以便标题行被写入到输出文件，只写一次。if 块处理第一个输入文件，并写所有的行，包括标题行，到输出文件。else 块处理所有余下的输入文件并用 next 方法将标题行赋值给变量（后面的处理有效的去除它），然后将余下的行写入输出文件。

要运行脚本，在命令行中输入以下并回车:

```
python 9csv_reader_concat_rows_from_multiple_files.py "C:\Users\Clinton\Desktop"\
output_files\9output.csv
```

你可以看到输入文件的名字打印到屏幕，如图 2-16。

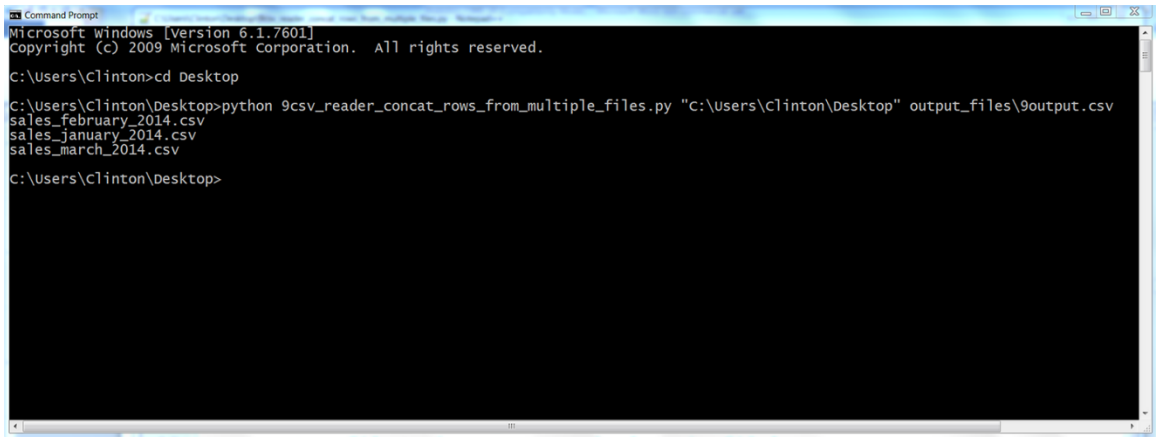
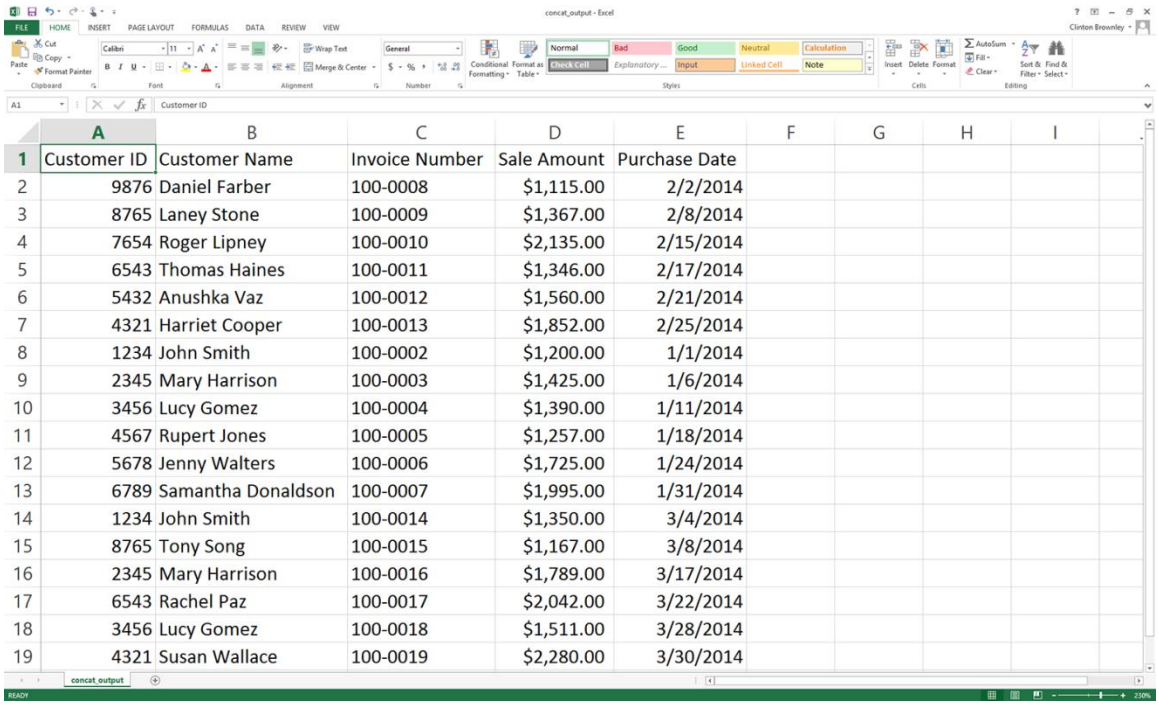


图 2-16. Python 脚本的输出：文件的名字拼接到输出文件

屏幕中的输出显示处理的文件的名字。另外脚本拼接了三个输入文件的数据到一个输出文件 9output.csv 中，它位于 桌面的 output_files 目录。图 2-17 显示拼接看起来的样子。



	A	B	C	D	E
1	Customer ID	Customer Name	Invoice Number	Sale Amount	Purchase Date
2	9876	Daniel Farber	100-0008	$1,115.00	2/2/2014
3	8765	Laney Stone	100-0009	$1,367.00	2/8/2014
4	7654	Roger Lipney	100-0010	$2,135.00	2/15/2014
5	6543	Thomas Haines	100-0011	$1,346.00	2/17/2014
6	5432	Anushka Vaz	100-0012	$1,560.00	2/21/2014
7	4321	Harriet Cooper	100-0013	$1,852.00	2/25/2014
8	1234	John Smith	100-0002	$1,200.00	1/1/2014
9	2345	Mary Harrison	100-0003	$1,425.00	1/6/2014
10	3456	Lucy Gomez	100-0004	$1,390.00	1/11/2014
11	4567	Rupert Jones	100-0005	$1,257.00	1/18/2014
12	5678	Jenny Walters	100-0006	$1,725.00	1/24/2014
13	6789	Samantha Donaldson	100-0007	$1,995.00	1/31/2014
14	1234	John Smith	100-0014	$1,350.00	3/4/2014
15	8765	Tony Song	100-0015	$1,167.00	3/8/2014
16	2345	Mary Harrison	100-0016	$1,789.00	3/17/2014
17	6543	Rachel Paz	100-0017	$2,042.00	3/22/2014
18	3456	Lucy Gomez	100-0018	$1,511.00	3/28/2014
19	4321	Susan Wallace	100-0019	$2,280.00	3/30/2014

图 2-17. 输出 CSV 文件，包含从输入文件拼接的行

该图显示脚本成功的拼接了三个输入文件的数据。输出文件包含标题行和三个输入文件的所有的数据行。在代码讨论中，我提出为什么我们在第 13 行用'a' (append mode)而不是 'w' (write mode)。我也提出为什么我们要区别第一个输入文件和其它后面的文件。要学习和实验，你可以改变 'a' 到 'w'，然后再保存和运行脚本，然后看输出的变化。相似的，你想要去除 if-else 语句，并打印所有输入文件的所有行看输出的变化。重要的一点是，本例中的模式，'sales_*'，相对具体，意思是你不想在桌面有任何文件以'sales_'开头，除三个输入文件外。另一种情况同，

你更想用不太具体的模式，如'*.csv'来查找所有的 CSV 文件。这些情况，你不想创建你的输出文件在包含输入文件的目录中。你不想这么做的原因是在脚本中，你打开输出文件的同时打开了输入文件。所以，如果你的模式是'*.csv'，而你是输出文件是 CSV 文件，你的脚本处理输出文件就如处理输入文件一样，这会出现问题和错误。这是为什么要将输出文件放在不同目录的原因。

Pandas

Pandas 可以直接从多个文件拼接数据。基本的过程是读输入文件到 pandas 的 DataFrame，添加所有的 DataFrame 到 DataFrame 的列表，然后用 concat 函数将所有的 DataFrame 拼接到一个 DataFrame。Concat 函数有一个轴参数，可以用来指定你要垂直堆叠或水平堆叠 DataFrames。要用 pandas 将多个输入文件垂直堆叠到一个输出文件，在文本编辑器中输入如下代码并将文件保存为 pandas_concat_rows_from_multiple_files.py:

```
#!/usr/bin/env python3
import pandas as pd
import glob
import os
import sys
input_path = sys.argv[1]
output_file = sys.argv[2]
all_files = glob.glob(os.path.join(input_path,'sales_*'))
all_data_frames = []
for file in all_files:
data_frame = pd.read_csv(file, index_col=None)
all_data_frames.append(data_frame)
data_frame_concat = pd.concat(all_data_frames, axis=0, ignore_index=True)
data_frame_concat.to_csv(output_file, index = False)
```

本代码垂直堆叠 DataFrames。如果你要水平的拼接，则在 concat 函数中设置 axis=1。除了 DataFrame，pandas 也有 Series 数据容器。你用相同的代码来拼接 Series，只是你要拼接的对象是 Series 而不是 DataFrames。有时，并不是简单的水平或垂直拼接数据，你想基于数据集中关键列的值来合并数据集。Pandas 提供了 merge 函数，就像 SQL join 一样的操作。如果你熟悉 SQL joins，则你很容易理解 merge 的代码。pd.merge(DataFrame1, DataFrame2, on='key', how='inner')。NumPy，另一个 Python 模块，也提供了多个函数来水平或垂直的拼接数据。通常要 import NumPy as np。然后，要垂直拼接，你可以用 np.concatenate([array1, array2], axis=0)，np.vstack((array1, array2))，or np.r_[array1, array2]。相似的，你可以水平拼接数据，用 np.concatenate([array1,array2], axis=1)，np.hstack((array1, array2))，or np.c_[array1, array2]。

要运行脚本，你在命令行中输入以下并回车。

```
python pandas_concat_rows_from_multiple_files.py "C:\Users\Clinton\Desktop"\
output_files\pandas_output.csv
```

你打开输出文件 pandas_output.csv,检查结果。

按文件求数据集的和以及平均值

有时候你有多个文件，你想要计算每一个文件的统计量。本节的例子用我们创建的三个 CSV 文件，展示如何计算每个文件列的和以及平均值。

基础 Python

要用基础 python 计算多个文件的列的和以及平均值，在文本编辑器中输入以下代码并保存为 10csv_reader_sum_average_from_multiple_files:

```
1 #!/usr/bin/env python3
2 import csv
3 import glob
4 import os
5 import sys
6 input_path = sys.argv[1]
7 output_file = sys.argv[2]
8 output_header_list = ['file_name', 'total_sales', 'average_sales']
9 csv_out_file = open(output_file, 'a', newline='')
10 filewriter = csv.writer(csv_out_file)
11 filewriter.writerow(output_header_list)
12 for input_file in glob.glob(os.path.join(input_path,'sales_*')):
13 with open(input_file, 'r', newline='') as csv_in_file:
14 filereader = csv.reader(csv_in_file)
15 output_list = [ ]
16 output_list.append(os.path.basename(input_file))
17 header = next(filereader)
18 total_sales = 0.0
19 number_of_sales = 0.0
20 for row in filereader:
21 sale_amount = row[3]
22 total_sales += float(str(sale_amount).strip('$').replace(',',''))
23 number_of_sales += 1
24 average_sales = '{0:.2f}'.format(total_sales / number_of_sales)
25 output_list.append(total_sales)
26 output_list.append(average_sales)
27 filewriter.writerow(output_list)
28 csv_out_file.close()
```

第 8 行为输出文件创建列标题的列表。第 10 行创建 filewriter 对象，第 11 行写标题行到输出文件。第 15 行创建空的列表，贮存我们要写入到输出文件的每一行输出。因为我们想计算输入文件的和以及平均值，第 16 行添加输入文件的名到 output_list。第 17 行用 next 函数去除每个输入文件的标题行。第 18 行创建变量 total_sales 并设置值为 0。相似的，第 19 行创建变量 number_of_sales 并设置值为 0。第 20 行是 for 循环遍历每个输入文件的数据行。第 21 行用列表索

引提取 Sale Amount 列的值并赋值给 sale_amount。第 22 行用 str 函数确保 sale_amount 内的值是字串然后用 strip 和 replace 函数去除美元符号和逗号。然后用 float 函数转换值到浮点数，将值加到 total_sales。第 23 行，将 number_of_sales 加 1。第 24 行将 total_sales 的值除于 number_of_sales 的值以计算输入文件的平均销售量，将这个值格式化为二位小数转换为字串，赋值给 average_sales。第 25 加总销售量作为 output_list 的第二个值。第一个值是输入文件名。这个值加到第 17 行的列表。第 26 行加平均销售量作为 output_list 的第三个值。第 27 行写 output_list 值到到输出文件。脚本为每个输入文件执行代码，所以输出文件包含文件名列，总销量列，平均销量列，对应于每一个输入文件。要运行脚本，在命令行中输入以下并回车。

```
python 10csv_reader_sum_average_from_multiple_files.py \
"C:\Users\Clinton\Desktop" output_files\10output.csv
```

你可以找开输出文件 0output.csv,检查结果。

Pandas

Pandas 提供了 summary 统计函数，像 sum 和 mean，你可以用来计算行和列的统计量。下面的代码说明如何计算多个输入文件的特定列的两个统计量(sum 和 mean)并将结果写入输出文件 。

要用 pandas 计算两列统计量，输入如下代码到文件编辑器中并将文件保存为 pandas_sum_average_from_multiple_files.py:

```
#!/usr/bin/env python3
import pandas as pd
import glob
import os
import sys
input_path = sys.argv[1]
output_file = sys.argv[2]
all_files = glob.glob(os.path.join(input_path,'sales_*'))
all_data_frames = []
for input_file in all_files:
data_frame = pd.read_csv(input_file, index_col=None)
total_cost = pd.DataFrame([float(str(value).strip('$').replace(',','')) \
for value in data_frame.loc[:, 'Sale Amount']]).sum()
average_cost = pd.DataFrame([float(str(value).strip('$').replace(',','')) \
for value in data_frame.loc[:, 'Sale Amount']]).mean()
data = {'file_name': os.path.basename(input_file),
'total_sales': total_sales,
'average_sales': average_sales}
all_data_frames.append(pd.DataFrame(data, \
columns=['file_name', 'total_sales', 'average_sales']))
data_frames_concat = pd.concat(all_data_frames, axis=0, ignore_index=True)
data_frames_concat.to_csv(output_file, index = False)
```

我们用列表推导式将 Sale Amount 列的字串美元值转换为浮点数，然后用 DataFrame 函数转换对像到 DataFrame 以便我们用这两个函数来计算列的 sum 和 mean 值。因为输出文件的每一

行包含输入文件的文件名,Sale Amount 列的总和以及平均值,我们组合这三种数据到DataFrame,用 concat 函数拼接所有的 DataFrames 到一个 DataFrames。然后将 DataFrames 写入输出文件。

要运行脚本，在命令行中输入如下并回车:

```
python pandas_ sum_average_from_multiple_files.py "C:\Users\Clinton\Desktop"\
output_files\pandas_output.csv
```

你可以找开输出文件 pandas_output.csv,检查结果。

这一章我们讲了许多。我们计论了如何读取和解析 CSV 文件，导航 CSV 文件的行和列，处理多个 CSV 文件，计算多个 CSV 文件的统计量。如果你按照本章的例子，你写了 12 个 python 脚本。本章的最好的部分是这些例子是你导航和处理文件的基本部件。完成了本章的例子，你可以准备处理 Excel 文件了，这是下一章的主题。