

1 Importing and Exporting Data

1.1 9.1 Importing Data using pandas

Pandas is an increasingly important component of the Python scientific stack, and a complete discussion of its main features is included in Chapter 17.

All of the data readers in pandas load data into a pandas DataFrame (see Section 17.1.2), and so these examples all make use of the `values` property to extract a NumPy array.

In practice, the DataFrame is much more useful since it includes useful information such as column names read from the data source. In addition to the three formats presented here, pandas can also read json, SQL, html tables or from the clipboard, which is particularly useful for interactive work since virtually any source that can be copied to the clipboard can be imported.

1.2 9.1.1 CSV and other formatted text files

Comma-separated value (CSV) files can be read using `read_csv`. When the CSV file contains mixed data, the default behavior will read the file into an array with an object data type, and so further processing is usually required to extract the individual series.

```
>>> from pandas import read_csv
>>> csv_data = read_csv('FTSE_1984_2012.csv')
>>> csv_data = csv_data.values
>>> csv_data[:4]
array([[2012-02-15, 5899.9, 5923.8, 5880.6, 5892.2, 801550000L, 5892.2],
       [2012-02-14, 5905.7, 5920.6, 5877.2, 5899.9, 832567200L, 5899.9],
       [2012-02-13, 5852.4, 5920.1, 5852.4, 5905.7, 643543000L, 5905.7],
       [2012-02-10, 5895.5, 5895.5, 5839.9, 5852.4, 948790200L, 5852.4]], dtype=object)
>>> open = csv_data[:,1]
```

1.3 Reading Data

To create a DataFrame out of common Python data structures, we can pass a dictionary of lists to the DataFrame constructor.

Using the columns parameter allows us to tell the constructor how we'd like the columns ordered. By default, the DataFrame constructor will order the columns alphabetically (though this isn't the case when reading from a file - more on that next).

```
In [17]:
data = {'year': [2010, 2011, 2012, 2011, 2012, 2010, 2011, 2012],
        'team': ['Bears', 'Bears', 'Bears', 'Packers', 'Packers', 'Lions', 'Lions', 'Lions'],
        'wins': [11, 8, 10, 15, 11, 6, 10, 4],
        'losses': [5, 8, 6, 1, 5, 10, 6, 12]}
football = pd.DataFrame(data, columns=['year', 'team', 'wins', 'losses'])
print football
```

	year	team	wins	losses
0	2010	Bears	11	5
1	2011	Bears	8	8
2	2012	Bears	10	6
3	2011	Packers	15	1
4	2012	Packers	11	5
5	2010	Lions	6	10
6	2011	Lions	10	6
7	2012	Lions	4	12

Much more often, you'll have a dataset you want to read into a DataFrame. Let's go through several common ways of doing so.

1.4 CSV

Reading a CSV is as simple as calling the read_csv function. By default, the read_csv function expects the column separator to be a comma, but you can change that using the sep parameter.

```
%cd ~/Dropbox/tutorials/pandas/
/Users/greda/Dropbox/tutorials/pandas

# Source: baseball-reference.com/players/r/riverma01.shtml
!head -n 5 mariano-rivera.csv
```

```

Year, Age, Tm, Lg, W, L, W-L%, ERA, G, GS, GF, CG, SHO, SV, IP, H, R, ER, HR, BB, IBB, SO, HBP, BK, WP, BF, ERA+, WHIP
1995, 25, NYY, AL, 5, 3, .625, 5.51, 19, 10, 2, 0, 0, 0, 67.0, 71, 43, 41, 11, 30, 0, 51, 2, 1, 0, 301, 84, 1.507, 9.5
1996, 26, NYY, AL, 8, 3, .727, 2.09, 61, 0, 14, 0, 0, 5, 107.2, 73, 25, 25, 1, 34, 3, 130, 2, 0, 1, 425, 240, 0.994, 6.1
1997, 27, NYY, AL, 6, 4, .600, 1.88, 66, 0, 56, 0, 0, 43, 71.2, 65, 17, 15, 5, 20, 6, 68, 0, 0, 2, 301, 239, 1.186, 8.2
1998, 28, NYY, AL, 3, 0, 1.000, 1.91, 54, 0, 49, 0, 0, 36, 61.1, 48, 13, 13, 3, 17, 1, 36, 1, 0, 0, 246, 233, 1.060, 7.0

```

```

In [20]:
from_csv = pd.read_csv('mariano-rivera.csv')
from_csv.head()

```

```

Year Age Tm Lg W L W-L% ERA G GS GF CG SHO SV IP H R ER HR BB
IBB SO HBP BK WP BF ERA+ WHIP H/9 HR/9 BB/9 SO/9 SO/BB Awards
0 1995 25 NYY AL 5 3 0.625 5.51 19 10 2 0 0 0 67.0 71 43 41 11 30 0 51 2 1 0 301 84 1.507 9.5
1.5 4.0 6.9 1.70 NaN
1 1996 26 NYY AL 8 3 0.727 2.09 61 0 14 0 0 5 107.2 73 25 25 1 34 3 130 2 0 1 425 240 0.994
6.1 0.1 2.8 10.9 3.82 CYA-3MVP-12
2 1997 27 NYY AL 6 4 0.600 1.88 66 0 56 0 0 43 71.2 65 17 15 5 20 6 68 0 0 2 301 239 1.186 8.2
0.6 2.5 8.5 3.40 ASMVP-25
3 1998 28 NYY AL 3 0 1.000 1.91 54 0 49 0 0 36 61.1 48 13 13 3 17 1 36 1 0 0 246 233 1.060 7.0
0.4 2.5 5.3 2.12 NaN
4 1999 29 NYY AL 4 3 0.571 1.83 66 0 63 0 0 45 69.0 43 15 14 2 18 3 52 3 1 2 268 257 0.884 5.6
0.3 2.3 6.8 2.89 ASCYA-3MVP-14

```

Our file had headers, which the function inferred upon reading in the file. Had we wanted to be more explicit, we could have passed `header=None` to the function along with a list of column names to use:

```

# Source: pro-football-reference.com/players/M/MannPe00/touchdowns/passing/2012/
!head -n 5 peyton-passing-TDs-2012.csv
1,1,2012-09-09,DEN,,PIT,W 31-19,3,71,Demaryius Thomas,Trail 7-13,Lead 14-13*
2,1,2012-09-09,DEN,,PIT,W 31-19,4,1,Jacob Tamme,Trail 14-19,Lead 22-19*
3,2,2012-09-17,DEN,@,ATL,L 21-27,2,17,Demaryius Thomas,Trail 0-20,Trail 7-20
4,3,2012-09-23,DEN,,HOU,L 25-31,4,38,Brandon Stokley,Trail 11-31,Trail 18-31

```

```

5,3,2012-09-23,DEN,,HOU,L 25-31,4,6,Joel Dreessen,Trail 18-31,Trail 25-31

In [22]:
cols = ['num', 'game', 'date', 'team', 'home_away', 'opponent',
        'result', 'quarter', 'distance', 'receiver', 'score_before',
        'score_after']
no_headers = pd.read_csv('peyton-passing-TDs-2012.csv', sep=',', header=None,
                        names=cols)

no_headers.head()
Out[22]:
num game date team home_away opponent result quarter distance
receiver score_before score_after
0 1 1 2012-09-09 DEN NaN PIT W 31-19 3 71
Demaryius Thomas Trail 7-13 Lead 14-13*
1 2 1 2012-09-09 DEN NaN PIT W 31-19 4 1 Jacob Tamm
Trail 14-19 Lead 22-19*
2 3 2 2012-09-17 DEN @ ATL L 21-27 2 17
Demaryius Thomas Trail 0-20 Trail 7-20
3 4 3 2012-09-23 DEN NaN HOU L 25-31 4 38
Brandon Stokley Trail 11-31 Trail 18-31
4 5 3 2012-09-23 DEN NaN HOU L 25-31 4 6
Joel Dreessen Trail 18-31 Trail 25-31

```

pandas various reader functions have many parameters allowing you to do things like skipping lines of the file, parsing dates, or specifying how to handle NA/NULL datapoints.

There's also a set of writer functions for writing to a variety of formats (CSVs, HTML tables, JSON). They function exactly as you'd expect and are typically called `to_format`:

```
my_dataframe.to_csv('path_to_file.csv')
```

Take a look at the IO documentation to familiarize yourself with file reading/writing functionality.

1.5 Excel

Know who hates VBA? Me. I bet you do, too. Thankfully, pandas allows you to read and write Excel files, so you can easily read from Excel, write your code in Python, and then write back out to Excel - no need for VBA.

Reading Excel files requires the `xlrd` library. You can install it via pip (`pip install xlrd`).

Let's first write a DataFrame to Excel.

```
# this is the DataFrame we created from a dictionary earlier
print football.head()
```

```
   year team wins losses
0  2010  Bears   11     5
1  2011  Bears    8     8
2  2012  Bears   10     6
3  2011 Packers   15     1
4  2012 Packers   11     5
```

```
# since our index on the football DataFrame is meaningless, let's not write it
football.to_excel('football.xlsx', index=False)

!ls -l *.xlsx
-rw-r--r--  1 greda  staff  5618 Oct 26 00:44 football.xlsx

# delete the DataFrame
del football

# read from Excel
football = pd.read_excel('football.xlsx', 'sheet1')
print football
```

```
   year team wins losses
0  2010  Bears   11     5
1  2011  Bears    8     8
2  2012  Bears   10     6
3  2011 Packers   15     1
4  2012 Packers   11     5
5  2010  Lions    6    10
6  2011  Lions   10     6
7  2012  Lions    4    12
```

1.6 Databases

pandas also has some support for reading/writing DataFrames directly from/to a database [docs]. You'll typically just need to pass a connection object to the `read_frame` or `write_frame` functions within the `pandas.io` module.

Note that `write_frame` executes as a series of INSERT INTO statements and thus trades speed for simplicity. If you're writing a large DataFrame to a database, it might be quicker to write the DataFrame to CSV and load that directly using the database's file import arguments.

```
from pandas.io import sql
import sqlite3

conn = sqlite3.connect('/Users/greda/Dropbox/gregreda.com/_code/towed')
query = "SELECT * FROM towed WHERE make = 'FORD';"

results = sql.read_frame(query, con=conn)
print results.head()
```

	tow_date	make	style	model	color	plate	state	towed_address \
0	01/19/2013	FORD	LL		RED	N786361	IL	400 E. Lower Wacker
1	01/19/2013	FORD	4D		GRN	L307211	IL	701 N. Sacramento
2	01/19/2013	FORD	4D		GRY	P576738	IL	701 N. Sacramento
3	01/19/2013	FORD	LL		BLK	N155890	IL	10300 S. Doty
4	01/19/2013	FORD	LL		TAN	H953638	IL	10300 S. Doty

	phone	inventory
0	(312) 744-7550	877040
1	(773) 265-7605	6738005
2	(773) 265-7605	6738001
3	(773) 568-8495	2699210
4	(773) 568-8495	2699209