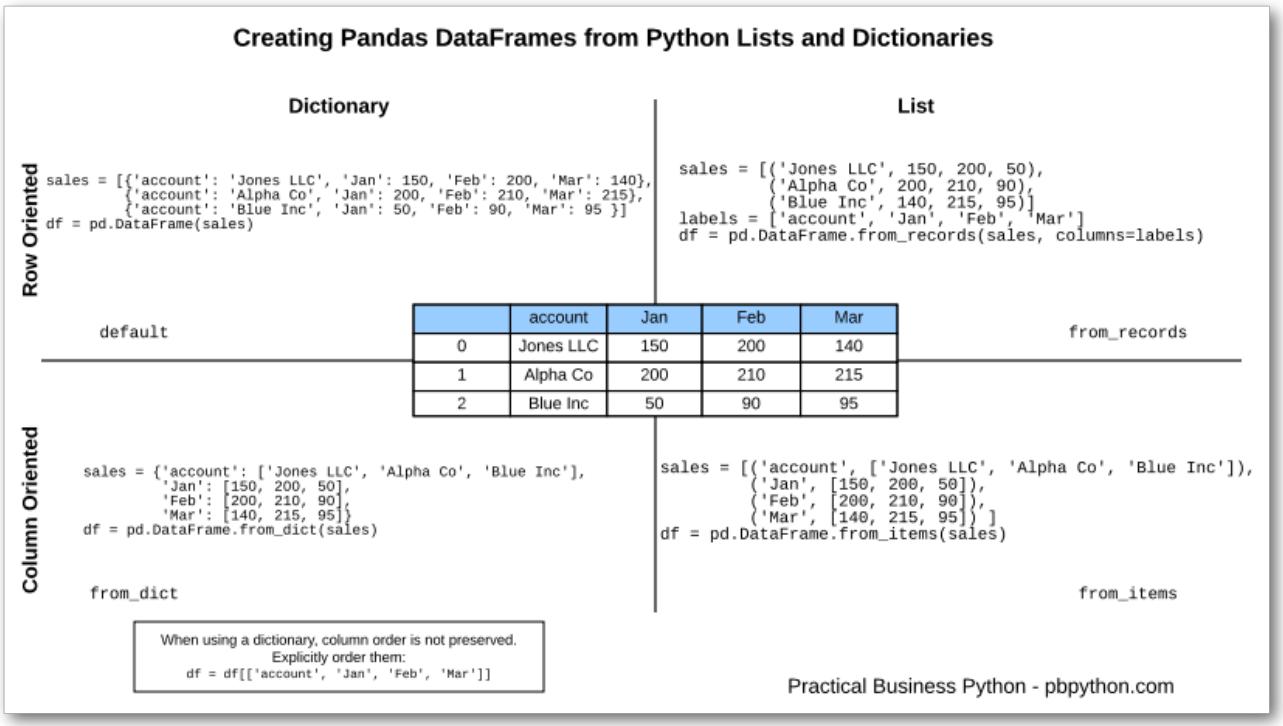
# Practical Business Python (https://pbpython.com/)

Taking care of business, one python script at a time

Tue 06 September 2016

# Creating Pandas DataFrames from Lists and Dictionaries (https://pbpython.com/pandas-list-dict.html)

Posted by Chris Moffitt (https://pbpython.com/author/chris-moffitt.html) in articles (https://pbpython.com/category/articles.html)




## Introduction

Whenever I am doing analysis with pandas my first goal is to get data into a panda's DataFrame using one of the many available options (http://pandas.pydata.org/pandas-docs/stable/io.html). For the vast majority of instances, I use `read_excel` , `read_csv` , or `read_sql` .

However, there are instances when I just have a few lines of data or some calculations that I want to include in my analysis. In these cases it is helpful to know how to create DataFrames from standard python lists or dictionaries. The basic process is not difficult but because there are several different options it is helpful to understand how each works. I can never remember whether I should use `from_dict`, `from_records`, `from_items` or the default `DataFrame` constructor. Normally, through some trial and error, I figure it out. Since it is still confusing to me, I thought I would walk through several examples below to clarify the different approaches. At the end of the article, I briefly show how this can be useful when generating Excel reports.

# DataFrames from Python Structures

There are multiple methods you can use to take a standard python datastructure and create a panda's DataFrame. For the purposes of these examples, I'm going to create a DataFrame with 3 months of sales information for 3 fictitious companies.

| | account | Jan | Feb | Mar |
|---|---|---|---|---|
| 0 | Jones LLC | 150 | 200 | 140 |
| 1 | Alpha Co | 200 | 210 | 215 |
| 2 | Blue Inc | 50 | 90 | 95 |

## Dictionaries

Before showing the examples below, I am assuming the following imports have been executed:

```
import pandas as pd
from collections import OrderedDict
from datetime import date
```

The "default" manner to create a DataFrame from python is to use a list of dictionaries. In this case each dictionary key is used for the column headings. A default index will be created automatically:

```
sales = [{'account': 'Jones LLC', 'Jan': 150, 'Feb': 200, 'Mar': 140},
         {'account': 'Alpha Co',  'Jan': 200, 'Feb': 210, 'Mar': 215},
         {'account': 'Blue Inc',  'Jan': 50,  'Feb': 90,  'Mar': 95 }]
df = pd.DataFrame(sales)
```

| | Feb | Jan | Mar | account |
|---|---|---|---|---|
| 0 | 200 | 150 | 140 | Jones LLC |
| 1 | 210 | 200 | 215 | Alpha Co |
| 2 | 90 | 50 | 95 | Blue Inc |

As you can see, this approach is very "row oriented". If you would like to create a DataFrame in a "column oriented" manner, you would use `from_dict`

```
sales = {'account': ['Jones LLC', 'Alpha Co', 'Blue Inc'],
         'Jan': [150, 200, 50],
         'Feb': [200, 210, 90],
         'Mar': [140, 215, 95]}
df = pd.DataFrame.from_dict(sales)
```

Using this approach, you get the same results as above. The key point to consider is which method is easier to understand in your unique situation. Sometimes it is easier to get your data in a row oriented approach and others in a column oriented. Knowing the options will help make your code simpler and easier to understand for your particular need.

Most of you will notice that the order of the columns looks wrong. The issue is that the standard python dictionary does not preserve the order of its keys. If you want to control column order then there are two options.

First, you can manually re-order the columns:

```
df = df[['account', 'Jan', 'Feb', 'Mar']]
```

Alternatively you could create your dictionary using python's `OrderedDict` .

```
sales = OrderedDict([ ('account', ['Jones LLC', 'Alpha Co', 'Blue Inc']),
          ('Jan', [150, 200, 50]),
          ('Feb',  [200, 210, 90]),
          ('Mar', [140, 215, 95]) ] )
df = pd.DataFrame.from_dict(sales)
```

Both of these approaches will give you the results in the order you would likely expect.

| | account | Jan | Feb | Mar |
|---|---|---|---|---|
| 0 | Jones LLC | 150 | 200 | 140 |
| 1 | Alpha Co | 200 | 210 | 215 |
| 2 | Blue Inc | 50 | 90 | 95 |

For reasons I outline below, I tend to specifically re-order my columns vs. using an OrderedDict but it is always good to understand the options.

## Lists

The other option for creating your DataFrames from python is to include the data in a list structure.

The first approach is to use a row oriented approach using pandas `from_records` . This approach is similar to the dictionary approach but you need to explicitly call out the column labels.

```
sales = [('Jones LLC', 150, 200, 50),
         ('Alpha Co', 200, 210, 90),
         ('Blue Inc', 140, 215, 95)]
labels = ['account', 'Jan', 'Feb', 'Mar']
df = pd.DataFrame.from_records(sales, columns=labels)
```

The second method is the `from_items` which is column oriented and actually looks similar to the `OrderedDict` example above.

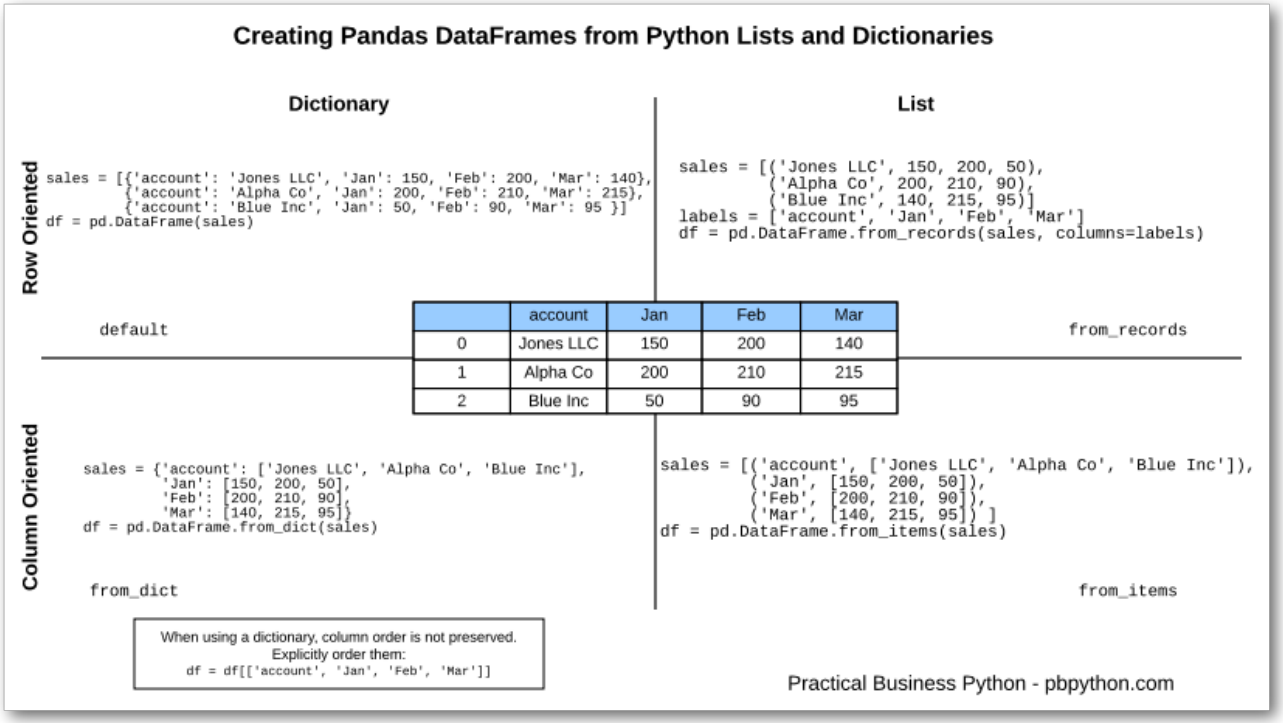
```
sales = [('account', ['Jones LLC', 'Alpha Co', 'Blue Inc']),
         ('Jan', [150, 200, 50]),
         ('Feb', [200, 210, 90]),
         ('Mar', [140, 215, 95]),
         ]
df = pd.DataFrame.from_items(sales)
```

Both of these examples will generate the following DataFrame:

| | account | Jan | Feb | Mar |
|---|---|---|---|---|
| 0 | Jones LLC | 150 | 200 | 140 |
| 1 | Alpha Co | 200 | 210 | 215 |
| 2 | Blue Inc | 50 | 90 | 95 |

# Keeping the Options Straight

In order to keep the various options clear in my mind, I put together this simple graphic to show the dictionary vs. list options as well as row vs. column oriented approaches. It's a 2X2 grid so I hope all the consultants are impressed!




For the sake of simplicity, I am not showing the `OrderedDict` approach because the `from_items` approach is probably a more likely real world solution.

If this is a little hard to read, you can also get the PDF version (https://pbpython.com/extras/DataFrame-From-Python.pdf).

# Simple Example

This may seem like a lot of explaining for a simple concept. However, I frequently use these approaches to build small DataFrames that I combine with my more complicated analysis.

For one example, let's say we want to save our DataFrame and include a footer so we know when it was created and who it was created by. This is much easier to do if we populate a DataFrame and write it to Excel than if we try to write individual cells to Excel.

Take our existing DataFrame:

```
sales = [('account', ['Jones LLC', 'Alpha Co', 'Blue Inc']),
         ('Jan', [150, 200, 50]),
         ('Feb', [200, 210, 90]),
         ('Mar', [140, 215, 95]),
         ]
df = pd.DataFrame.from_items(sales)
```

Now build a footer (in a column oriented manner):

```
from datetime import date

create_date = "{:%m-%d-%Y}".format(date.today())
created_by = "CM"
footer = [('Created by', [created_by]), ('Created on', [create_date]), ('Version
df_footer = pd.DataFrame.from_items(footer)
```

| | Created by | Created on | Version |
|---|---|---|---|
| 0 | CM | 09-05-2016 | 1.1 |

Combine into a single Excel sheet:

```
writer = pd.ExcelWriter('simple-report.xlsx', engine='xlsxwriter')
df.to_excel(writer, index=False)
df_footer.to_excel(writer, startrow=6, index=False)
writer.save()
```

| | A | B | C | D |
|---|---|---|---|---|
| 1 | account | Jan | Feb | Mar |
| 2 | Jones LLC | 150 | 200 | 140 |
| 3 | Alpha Co | 200 | 210 | 215 |
| 4 | Blue Inc | 50 | 90 | 95 |
| 5 | | | | |
| 6 | | | | |
| 7 | Created by | Created on | Version | |
| 8 | CM | 09-05-2016 | 1.1 | |
| 9 | | | | |

The secret sauce here is to use `startrow` to write the footer DataFrame below the sales DataFrame. There is also a corresponding `startcol` so you can control the column layout as well. This allows for a lot of flexibility with the basic `to_excel` function.

# Summary

Most pandas users quickly get familiar with ingesting spreadsheets, CSVs and SQL data. However, there are times when you will have data in a basic list or dictionary and want to populate a DataFrame. Pandas offers several options but it may not always be immediately clear on when to use which ones.

There is no one approach that is “best”, it really depends on your needs. I tend to like the list based methods because I normally care about the ordering and the lists make sure I preserve the order. The most important thing is to know the options are available so you can be smart about using the simplest one for your specific case.

On the surface, these samples may seem simplistic but I do find that it is pretty common that I use these methods to generate quick snippets of information that can augment or clarify the more complex analysis. The nice thing about data in a DataFrame is that it is very easy to convert into other formats such as Excel, CSV, HTML, LaTeX, etc. This flexibility is really handy for ad-hoc report generation.

# Updates

- 19-Nov-2018: As of pandas 0.23, `DataFrame.from_items()` has been deprecated. You can use `DataFrame.from_dict(dict(items))` instead. If you want to preserve order, you can use `DataFrame.from_dict(OrderedDict(items))`

← Introduction to Data Visualization with Altair (https://pbpython.com/altair-intro.html)

Building a Financial Model with Pandas → (https://pbpython.com/amortization-model.html)

Tags pandas (https://pbpython.com/tag/pandas.html) excel (https://pbpython.com/tag/excel.html)

# Comments

I have a doubt.What if the value of a key in dictionary is a tuple and i want to print only 1 value of that tuple.
For Example:-
The dataframe that is produced have whole tuple as a paralled value.
B
0 x (a,b,c)
1 y (d,e,f)

The above is the solution to my dictionary:
dict={
B:{
"x":[a,b,c],
"y":[d,e,f]
}
}

In the above example i want to print only a and d under the Column

Reply • Share ›

**David Howell** • a year ago

Consultants will all love those quadrants, great job ;)
It's also a really useful reference, thanks.

Reply • Share ›

**km** • a year ago

This is great, very helpful

Reply • Share ›

**MP** • a year ago

"Since it is still confusing to me, I though I would walk" -> "Since it is still confusing to me, I thought I would walk"

Reply • Share ›

**Chris Moffitt** Mod ➚ MP • a year ago

Ah. Good catch. Thanks for pointing it out. I'll fix it shortly.

1 Reply • Share ›

**Jose Gonzalez** • 2 years ago

The first diagram is perfect. Really, really helpful.

Reply • Share ›

**Mark Ginsburg** • 2 years ago

I think it's interesting in the "List" quadrant, you are actually passing in a List of Tuples. Just making a list doesn't seem to work.

Reply • Share ›

**Chris Moffitt** Mod ➚ Mark Ginsburg • 2 years ago

Yes. That's true. I have not looked at this in depth enough to see if there are other options without using the list of tuples.

Reply • Share ›

**Jordi Warmenhoven** • 3 years ago

You can actually use the standard DataFrame constructor to create the DataFrame from the dict in the lower left quadrant of the graphic. It will take the dict keys as the columns just as pd.DataFrame.from_dict(). However, if you have a dict of lists which is row oriented (dict keys are row indexes), then the .from_dict() method allows you to use the parameter orient='index' to create your DataFrame correctly.
http://nbviewer.jupyter.org...

Reply • Share ›

**Chris Moffitt** Mod → Jordi Warmenhoven • 3 years ago

Ah. Good point. It is amazing how complex this "simple" concept can get. Thanks for commenting.

Reply • Share ›

**ATUL VARSHNEY** → Chris Moffitt • 2 years ago

Hey. I want to insert my data frame in list ya dictionary in python. if u can write the code. then please share.

Reply • Share ›

**Abelardo** • 3 years ago

Thanks! There is another "idiom":
pd.DataFrame({'account': {0: 'Jones LLC', 1: 'Alpha Co', 2: 'Blue Inc'},
'Jan': {0: 150, 1:200, 2:50},
'Feb':{0:200, 1:210, 2:90},
'Mar':{0:140, 1:215, 2:95}})

Reply • Share ›

**Chris Moffitt** Mod → Abelardo • 3 years ago

Interesting. I did not realize that. Thank for letting us know.

Reply • Share ›

**shantanuo** • 3 years ago

I have read for the first time such a nice explanation of pandas DataFrames. For the sake of simplicity, I prefer to re-order the columns instead of using OrderedDict.

Reply • Share ›

**Вячеслав Буторов** • 3 years ago

Thanks for making things clearer)

Reply • Share ›

**Ricardo Colasanti** • 3 years ago

Thank you. The very thing I was trying to do. Very well explained.

Reply • Share ›

**Chris Moffitt** Mod → Ricardo Colasanti • 3 years ago

Great! I'm glad to hear it was helpful.

(https://pbpython.com/tag/s3.html) beautifulsoup (https://pbpython.com/tag/beautifulsoup.html) plotly (https://pbpython.com/tag/plotly.html) word (https://pbpython.com/tag/word.html) process (https://pbpython.com/tag/process.html) scikit (https://pbpython.com/tag/scikit.html) altair (https://pbpython.com/tag/altair.html) notebooks (https://pbpython.com/tag/notebooks.html) scikit-learn (https://pbpython.com/tag/scikit-learn.html) stdlib (https://pbpython.com/tag/stdlib.html) csv (https://pbpython.com/tag/csv.html) analyze-this (https://pbpython.com/tag/analyze-this.html) notebook (https://pbpython.com/tag/notebook.html) github (https://pbpython.com/tag/github.html) jinja (https://pbpython.com/tag/jinja.html) python (https://pbpython.com/tag/python.html) numpy (https://pbpython.com/tag/numpy.html) oauth2 (https://pbpython.com/tag/oauth2.html) excel (https://pbpython.com/tag/excel.html) ipython (https://pbpython.com/tag/ipython.html) google (https://pbpython.com/tag/google.html) mlxtend (https://pbpython.com/tag/mlxtend.html) vcs (https://pbpython.com/tag/vcs.html) xlwings (https://pbpython.com/tag/xlwings.html) pdf (https://pbpython.com/tag/pdf.html) pygal (https://pbpython.com/tag/pygal.html) matplotlib (https://pbpython.com/tag/matplotlib.html) pandas (https://pbpython.com/tag/pandas.html) vega (https://pbpython.com/tag/vega.html) seaborn (https://pbpython.com/tag/seaborn.html) outlook (https://pbpython.com/tag/outlook.html) barnum (https://pbpython.com/tag/barnum.html) cases (https://pbpython.com/tag/cases.html) powerpoint (https://pbpython.com/tag/powerpoint.html) pelican (https://pbpython.com/tag/pelican.html) matplot (https://pbpython.com/tag/matplot.html) plotting (https://pbpython.com/tag/plotting.html) gui (https://pbpython.com/tag/gui.html) bokeh (https://pbpython.com/tag/bokeh.html)

## Feeds

Atom Feed (https://pbpython.com/feeds/all.atom.xml)

## Disclosure

We are a participant in the Amazon Services LLC Associates Program, an affiliate advertising program designed to provide a means for us to earn fees by linking to Amazon.com and affiliated sites.

Custom Search


(https://www.python.org/psf-landing/)