

Practical Business Python (<http://pbpython.com/>)

Taking care of business, one python script at a time

Mon 29 December 2014

Pandas Pivot Table Explained (<http://pbpython.com/pandas-pivot-table-explained.html>)

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

Introduction

Most people likely have experience with pivot tables in Excel. Pandas provides a similar function called (appropriately enough) `pivot_table`. While it is exceedingly useful, I frequently find myself struggling to remember how to use the syntax to format the output for my needs. This article will focus on explaining the pandas `pivot_table` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.tools.pivot.pivot_table.html) function and how to use it for your data analysis.

If you are not familiar with the concept, wikipedia (http://en.wikipedia.org/wiki/Pivot_table) explains it in high level terms. BTW, did you know that Microsoft trademarked PivotTable? Neither did I. Needless to say, I'll be talking about a pivot table not PivotTable (<http://archive.oreilly.com/pub/a/windows/archive/whatisapivottable.html>)!

As an added bonus, I've created a simple cheat sheet that summarizes the `pivot_table`. You can find it at the end of this post and I hope it serves as a useful reference. Let me know if it is helpful.

The Data

One of the challenges with using the panda's `pivot_table` is making sure you understand your data and what questions you are trying to answer with the pivot table. It is a seemingly simple function but can produce very powerful analysis very quickly.

In this scenario, I'm going to be tracking a sales pipeline (also called funnel). The basic problem is that some sales cycles are very long (think "enterprise software", capital equipment, etc.) and management wants to understand it in more detail throughout the year.

Typical questions include:

- How much revenue is in the pipeline?
- What products are in the pipeline?
- Who has what products at what stage?
- How likely are we to close deals by year end?

Many companies will have CRM tools or other software that sales uses to track the process. While they may have useful tools for analyzing the data, inevitably someone will export the data to Excel and use a PivotTable to summarize the data.

Using a panda's pivot table can be a good alternative because it is:

- Quicker (once it is set up)
- Self documenting (look at the code and you know what it does)
- Easy to use to generate a report or email
- More flexible because you can define custome aggregation functions

Read in the data

Let's set up our environment first.

If you want to follow along, you can download the Excel file (<http://pbpython.com/extras/sales-funnel.xlsx>).

```
import pandas as pd
import numpy as np
```

Version Warning

The pivot_table API has changed over time so please make sure you have a recent version of pandas (> 0.15) installed for this example to work. This example also uses the category data type which requires a recent version as well.

Read in our sales funnel data into our DataFrame

```
df = pd.read_excel("../in/sales-funnel.xlsx")
df.head()
```

	Account	Name	Rep	Manager	Product	Quantity	Price	Status
0	714466	Trantow-Barrows	Craig Booker	Debra Henley	CPU	1	30000	presented

	Account	Name	Rep	Manager	Product	Quantity	Price	Status
1	714466	Trantow-Barrows	Craig Booker	Debra Henley	Software	1	10000	presented
2	714466	Trantow-Barrows	Craig Booker	Debra Henley	Maintenance	2	5000	pending
3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won

For convenience sake, let's define the status column as a `category` and set the order we want to view.

This isn't strictly required but helps us keep the order we want as we work through analyzing the data.

```
df["Status"] = df["Status"].astype("category")
df["Status"].cat.set_categories(["won", "pending", "presented", "declined"], inplace=True)
```

Pivot the data

As we build up the pivot table, I think it's easiest to take it one step at a time. Add items and check each step to verify you are getting the results you expect. Don't be afraid to play with the order and the variables to see what presentation makes the most sense for your needs.

The simplest pivot table must have a dataframe and an `index`. In this case, let's use the `Name` as our index.

```
pd.pivot_table(df, index=["Name"])
```

	Account	Price	Quantity
Name			
Barton LLC	740150	35000	1.000000
Fritsch, Russel and Anderson	737550	35000	1.000000
Herman LLC	141962	65000	2.000000
Jerde-Hilpert	412290	5000	2.000000
Kassulke, Ondricka and Metz	307599	7000	3.000000
Keeling LLC	688981	100000	5.000000
Kiehn-Spinka	146832	65000	2.000000
Koepp Ltd	729833	35000	2.000000

	Account	Price	Quantity
Name			
Kulas Inc	218895	25000	1.500000
Purdy-Kunde	163416	30000	1.000000
Stokes LLC	239344	7500	1.000000
Trantow-Barrows	714466	15000	1.333333

You can have multiple indexes as well. In fact, most of the `pivot_table` args can take multiple values via a list.

```
pd.pivot_table(df, index=["Name", "Rep", "Manager"])
```

			Account	Price	Quantity
Name	Rep	Manager			
Barton LLC	John Smith	Debra Henley	740150	35000	1.000000
Fritsch, Russel and Anderson	Craig Booker	Debra Henley	737550	35000	1.000000
Herman LLC	Cedric Moss	Fred Anderson	141962	65000	2.000000
Jerde-Hilpert	John Smith	Debra Henley	412290	5000	2.000000
Kassulke, Ondricka and Metz	Wendy Yule	Fred Anderson	307599	7000	3.000000
Keeling LLC	Wendy Yule	Fred Anderson	688981	100000	5.000000
Kiehn-Spinka	Daniel Hilton	Debra Henley	146832	65000	2.000000
Koepp Ltd	Wendy Yule	Fred Anderson	729833	35000	2.000000
Kulas Inc	Daniel Hilton	Debra Henley	218895	25000	1.500000
Purdy-Kunde	Cedric Moss	Fred Anderson	163416	30000	1.000000
Stokes LLC	Cedric Moss	Fred Anderson	239344	7500	1.000000
Trantow-Barrows	Craig Booker	Debra Henley	714466	15000	1.333333

This is interesting but not particularly useful. What we probably want to do is look at this by Manager and Rep. It's easy enough to do by changing the `index`.

```
pd.pivot_table(df, index=["Manager", "Rep"])
```

		Account	Price	Quantity
Manager	Rep			
Debra Henley	Craig Booker	720237.0	20000.000000	1.250000

		Account	Price	Quantity
Manager	Rep			
	Daniel Hilton	194874.0	38333.333333	1.666667
	John Smith	576220.0	20000.000000	1.500000
Fred Anderson	Cedric Moss	196016.5	27500.000000	1.250000
	Wendy Yule	614061.5	44250.000000	3.000000

You can see that the pivot table is smart enough to start aggregating the data and summarizing it by grouping the reps with their managers. Now we start to get a glimpse of what a pivot table can do for us.

For this purpose, the Account and Quantity columns aren't really useful. Let's remove it by explicitly defining the columns we care about using the `values` field.

```
pd.pivot_table(df, index=["Manager", "Rep"], values=["Price"])
```

		Price
Manager	Rep	
Debra Henley	Craig Booker	20000
	Daniel Hilton	38333
	John Smith	20000
Fred Anderson	Cedric Moss	27500
	Wendy Yule	44250

The price column automatically averages the data but we can do a count or a sum. Adding them is simple using `aggfunc` and `np.sum`.

```
pd.pivot_table(df, index=["Manager", "Rep"], values=["Price"], aggfunc=np.sum)
```

		Price
Manager	Rep	
Debra Henley	Craig Booker	80000
	Daniel Hilton	115000
	John Smith	40000
Fred Anderson	Cedric Moss	110000
	Wendy Yule	177000

`aggfunc` can take a list of functions. Let's try a mean using the numpy `mean` function and `len` to

get a count.

```
pd.pivot_table(df, index=["Manager", "Rep"], values=["Price"], aggfunc=[np.mean, len])
```

		mean	len
		Price	Price
Manager	Rep		
Debra Henley	Craig Booker	20000	4
	Daniel Hilton	38333	3
	John Smith	20000	2
Fred Anderson	Cedric Moss	27500	4
	Wendy Yule	44250	4

If we want to see sales broken down by the products, the `columns` variable allows us to define one or more columns.

Columns vs. Values

I think one of the confusing points with the `pivot_table` is the use of `columns` and `values`. Remember, `columns` are optional - they provide an additional way to segment the actual values you care about. The aggregation functions are applied to the `values` you list.

```
pd.pivot_table(df, index=["Manager", "Rep"], values=["Price"],
               columns=["Product"], aggfunc=[np.sum])
```

		sum			
		Price			
	Product	CPU	Maintenance	Monitor	Software
Manager	Rep				
Debra Henley	Craig Booker	65000	5000	NaN	10000
	Daniel Hilton	105000	NaN	NaN	10000
	John Smith	35000	5000	NaN	NaN
Fred Anderson	Cedric Moss	95000	5000	NaN	10000
	Wendy Yule	165000	7000	5000	NaN

The NaN's are a bit distracting. If we want to remove them, we could use `fill_value` to set them

to 0.

```
pd.pivot_table(df, index=["Manager", "Rep"], values=["Price"],
               columns=["Product"], aggfunc=[np.sum], fill_value=0)
```

		sum			
		Price			
	Product	CPU	Maintenance	Monitor	Software
Manager	Rep				
Debra Henley	Craig Booker	65000	5000	0	10000
	Daniel Hilton	105000	0	0	10000
	John Smith	35000	5000	0	0
Fred Anderson	Cedric Moss	95000	5000	0	10000
	Wendy Yule	165000	7000	5000	0

I think it would be useful to add the quantity as well. Add Quantity to the `values` list.

```
pd.pivot_table(df, index=["Manager", "Rep"], values=["Price", "Quantity"],
               columns=["Product"], aggfunc=[np.sum], fill_value=0)
```

		sum							
		Price				Quantity			
	Product	CPU	Maintenance	Monitor	Software	CPU	Maintenance	Monitor	Software
Manager	Rep								
Debra Henley	Craig Booker	65000	5000	0	10000	2	2	0	1
	Daniel Hilton	105000	0	0	10000	4	0	0	1
	John Smith	35000	5000	0	0	1	2	0	0
Fred Anderson	Cedric Moss	95000	5000	0	10000	3	1	0	1
	Wendy Yule	165000	7000	5000	0	7	3	2	0

What's interesting is that you can move items to the index to get a different visual representation.

Remove Product from the `columns` and add to the `index`.

```
pd.pivot_table(df, index=["Manager", "Rep", "Product"],
               values=["Price", "Quantity"], aggfunc=[np.sum], fill_value=0)
```

			sum	
			Price	Quantity
Manager	Rep	Product		
Debra Henley	Craig Booker	CPU	65000	2
		Maintenance	5000	2
		Software	10000	1
	Daniel Hilton	CPU	105000	4
		Software	10000	1
	John Smith	CPU	35000	1
		Maintenance	5000	2
Fred Anderson	Cedric Moss	CPU	95000	3
		Maintenance	5000	1
		Software	10000	1
	Wendy Yule	CPU	165000	7
		Maintenance	7000	3
		Monitor	5000	2

For this data set, this representation makes more sense. Now, what if I want to see some totals?

`margins=True` does that for us.

```
pd.pivot_table(df, index=["Manager", "Rep", "Product"],
               values=["Price", "Quantity"],
               aggfunc=[np.sum, np.mean], fill_value=0, margins=True)
```

			sum		mean	
			Price	Quantity	Price	Quantity
Manager	Rep	Product				
Debra Henley	Craig Booker	CPU	65000	2	32500.000000	1.000000
		Maintenance	5000	2	5000.000000	2.000000
		Software	10000	1	10000.000000	1.000000
	Daniel Hilton	CPU	105000	4	52500.000000	2.000000
		Software	10000	1	10000.000000	1.000000

			sum		mean	
			Price	Quantity	Price	Quantity
Manager	Rep	Product				
	John Smith	CPU	35000	1	35000.000000	1.000000
		Maintenance	5000	2	5000.000000	2.000000
Fred Anderson	Cedric Moss	CPU	95000	3	47500.000000	1.500000
		Maintenance	5000	1	5000.000000	1.000000
		Software	10000	1	10000.000000	1.000000
	Wendy Yule	CPU	165000	7	82500.000000	3.500000
		Maintenance	7000	3	7000.000000	3.000000
		Monitor	5000	2	5000.000000	2.000000
All			522000	30	30705.882353	1.764706

Let's move the analysis up a level and look at our pipeline at the manager level. Notice how the status is ordered based on our earlier category definition.

```
pd.pivot_table(df, index=["Manager", "Status"], values=["Price"],
               aggfunc=[np.sum], fill_value=0, margins=True)
```

		sum
		Price
Manager	Status	
Debra Henley	declined	70000
	pending	50000
	presented	50000
	won	65000
Fred Anderson	declined	65000
	pending	5000
	presented	45000
	won	172000
All		522000

A really handy feature is the ability to pass a dictionary to the `aggfunc` so you can perform different functions on each of the values you select. This has a side-effect of making the labels a little cleaner.

```
pd.pivot_table(df, index=["Manager", "Status"], columns=["Product"], values=["Quantity",
                                "Price"], aggfunc={"Quantity": len, "Price": np.sum}, fill_value=0)
```

		Price				Quantity			
		CPU	Maintenance	Monitor	Software	CPU	Maintenance	Monitor	Softw
Manager	Status								
Debra Henley	declined	70000	0	0	0	2	0	0	0
	pending	40000	10000	0	0	1	2	0	0
	presented	30000	0	0	20000	1	0	0	2
	won	65000	0	0	0	1	0	0	0
Fred Anderson	declined	65000	0	0	0	1	0	0	0
	pending	0	5000	0	0	0	1	0	0
	presented	30000	0	5000	10000	1	0	1	1
	won	165000	7000	0	0	2	1	0	0

You can provide a list of aggfunctions to apply to each value too:

```
table = pd.pivot_table(df, index=["Manager", "Status"], columns=["Product"], values=["Quantity",
                                "Price"], aggfunc={"Quantity": len, "Price": [np.sum, np.mean]}, fill_value=0)
table
```

		Price							
		mean				sum			
	Product	CPU	Maintenance	Monitor	Software	CPU	Maintenance	Monitor	Soft
Manager	Status								
Debra Henley	declined	35000	0	0	0	70000	0	0	0
	pending	40000	5000	0	0	40000	10000	0	0
	presented	30000	0	0	10000	30000	0	0	20000
	won	65000	0	0	0	65000	0	0	0
Fred Anderson	declined	65000	0	0	0	65000	0	0	0
	pending	0	5000	0	0	0	5000	0	0
	presented	30000	0	5000	10000	30000	0	5000	10000
	won	82500	7000	0	0	165000	7000	0	0

It can look daunting to try to pull this all together at one time but as soon as you start playing with the data and slowly add the items, you can get a feel for how it works. My general rule of thumb is that once

you use multiple `groupby` you should evaluate whether a pivot table is a useful approach.

Advanced Pivot Table Filtering

Once you have generated your data, it is in a `DataFrame` so you can filter on it using your standard `DataFrame` functions.

If you want to look at just one manager:

```
table.query('Manager == ["Debra Henley"]')
```

		Price							
		mean				sum			
	Product	CPU	Maintenance	Monitor	Software	CPU	Maintenance	Monitor	Softw
Manager	Status								
Debra Henley	declined	35000	0	0	0	70000	0	0	0
	pending	40000	5000	0	0	40000	10000	0	0
	presented	30000	0	0	10000	30000	0	0	2000
	won	65000	0	0	0	65000	0	0	0

We can look at all of our pending and won deals.

```
table.query('Status == ["pending", "won"]')
```

		Price							
		mean				sum			
	Product	CPU	Maintenance	Monitor	Software	CPU	Maintenance	Monitor	Softw
Manager	Status								
Debra Henley	pending	40000	5000	0	0	40000	10000	0	0
	won	65000	0	0	0	65000	0	0	0
Fred Anderson	pending	0	5000	0	0	0	5000	0	0
	won	82500	7000	0	0	165000	7000	0	0

This a powerful feature of the `pivot_table` so do not forget that you have the full power of pandas once you get your data into the `pivot_table` format you need.

The full notebook (<http://nbviewer.ipython.org/url/pbpython.com/extras/Pandas-Pivot-Table-Explained.ipynb>) is available if you would like to save it as a reference.

Cheat Sheet

In order to try to summarize all of this, I have created a cheat sheet that I hope will help you remember how to use the pandas `pivot_table`. Take a look and let me know what you think.

pandas pivot_table explained

	Account	Name	Rep	Manager	Product	Quantity	Price	Status
0	714466	Trantow-Barrows	Craig Booker	Debra Henley	CPU	1	30000	presented
1	714466	Trantow-Barrows	Craig Booker	Debra Henley	Software	1	10000	presented
2	714466	Trantow-Barrows	Craig Booker	Debra Henley	Maintenance	2	5000	pending
3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won

```
pd.pivot_table(df,
index=["Manager", "Status"],
columns=["Product"],
aggfunc=[np.sum],
values=["Price"],
fill_value=0,
margins=True,
dropna=True)
```

Can also use a dictionary:
`aggfunc={"Quantity":len, "Price":[np.sum,np.mean]}`



		sum				
		Price				
	Product	CPU	Maintenance	Monitor	Software	All
Manager	Status					
Debra Henley	declined	70000	0	0	0	70000
	pending	40000	10000	0	0	50000
	presented	30000	0	0	20000	50000
	won	65000	0	0	0	65000
Fred Anderson	declined	65000	0	0	0	65000
	pending	0	5000	0	0	5000
	presented	30000	0	5000	10000	45000
	won	165000	7000	0	0	172000
All		465000	22000	5000	30000	522000

pbpython.com

Thanks and good luck with creating your own pivot tables.

← Combining Data From Multiple Excel Files (<http://pbpython.com/excel-file-combine.html>)

Web Scraping - It's Your Civic Duty → (<http://pbpython.com/web-scraping-mn-budget.html>)

Tags  pandas (<http://pbpython.com/tag/pandas.html>)  excel (<http://pbpython.com/tag/excel.html>)

Tweet



Sha

Vote

97

Share


223



15 points

Comments

64 Comments pbpython.com

 Login ▾ Recommend 24 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH



OR SIGN UP WITH DISQUS (?)

Name

**Marco Massenzio** • 3 years ago

Excellent post - I really liked the gradual approach, it made it clear how the various arguments fit together.

Thanks for sharing!

4 ^ | ▾ • Reply • Share >

**Dogeun Kim** • 24 days ago

It's awesome code!!! I have a question.

I just added "margins=True" in your code, it is not working.
Error occurs "KeyError: 'Level None not found'"

Could it be working?

```
table = pd.pivot_table(df, index=["Manager", "Status"], columns=["Product"], values=
["Quantity", "Price"],
aggfunc={"Quantity": len, "Price": [np.sum, np.mean]}, fill_value=0, margins=True)
table
```

1 ^ | ▾ • Reply • Share >

**Xuanyao Wu** → **Dogeun Kim** • 22 days ago

I also met this issue

^ | ▾ • Reply • Share >

**Chris Moffitt** Mod → **Xuanyao Wu** • 22 days ago

Hmm. What version of pandas are you using?

^ | ▾ • Reply • Share >

**Dogeun Kim** → **Chris Moffitt** • 21 days ago

I'm using 0.20.1

^ | ▾ • Reply • Share >

**Chris Moffitt** Mod → **Dogeun Kim** • 20 days ago

Have you tried downloading the pivot table notebook from github and try to run it on your system? I just tried to run the notebook on my system and it works fine.

^ | ▾ • Reply • Share >

**Dogeun Kim** → **Chris Moffitt** • 20 days ago

Would you let me know where pivot table notebook site is. Thanks.



Social

Github (<https://github.com/chris1610/pbpython>)

Twitter (<https://twitter.com/chris1610>)

BitBucket (<http://bitbucket.org/chris1610>)

Reddit (<http://www.reddit.com/user/chris1610/>)

LinkedIn (<http://www.linkedin.com/in/cmoffitt>)

Categories

articles (<http://pbpython.com/category/articles.html>)

news (<http://pbpython.com/category/news.html>)

Popular

Pandas Pivot Table Explained (<http://pbpython.com/pandas-pivot-table-explained.html>)

Common Excel Tasks Demonstrated in Pandas (<http://pbpython.com/excel-pandas-comp.html>)

Overview of Python Visualization Tools (<http://pbpython.com/visualization-tools-1.html>)

Web Scraping - It's Your Civic Duty (<http://pbpython.com/web-scraping-mn-budget.html>)

Simple Graphing with IPython and Pandas (<http://pbpython.com/simple-graphing-pandas.html>)

Tags

plotting (<http://pbpython.com/tag/plotting.html>) plotly (<http://pbpython.com/tag/plotly.html>) powerpoint (<http://pbpython.com/tag/powerpoint.html>)

excel (<http://pbpython.com/tag/excel.html>) process (<http://pbpython.com/tag/process.html>) python (<http://pbpython.com/tag/python.html>)

ipython (<http://pbpython.com/tag/ipython.html>) google (<http://pbpython.com/tag/google.html>) sets (<http://pbpython.com/tag/sets.html>)

xlwings (<http://pbpython.com/tag/xlwings.html>) s3 (<http://pbpython.com/tag/s3.html>) stdlib (<http://pbpython.com/tag/stdlib.html>)

pandas (<http://pbpython.com/tag/pandas.html>) mlxtend (<http://pbpython.com/tag/mlxtend.html>) analyze-this (<http://pbpython.com/tag/analyze-this.html>)

pygal (<http://pbpython.com/tag/pygal.html>) matplotlib (<http://pbpython.com/tag/matplotlib.html>) bokeh (<http://pbpython.com/tag/matplotlib.html>)

(<http://pbpython.com/tag/bokeh.html>)  [ggplot](http://pbpython.com/tag/ggplot.html) (<http://pbpython.com/tag/ggplot.html>)  [beautifulsoup](http://pbpython.com/tag/beautifulsoup.html) (<http://pbpython.com/tag/beautifulsoup.html>)  [gui](http://pbpython.com/tag/gui.html) (<http://pbpython.com/tag/gui.html>)  [seaborn](http://pbpython.com/tag/seaborn.html) (<http://pbpython.com/tag/seaborn.html>)  [pelican](http://pbpython.com/tag/pelican.html) (<http://pbpython.com/tag/pelican.html>)  [oauth2](http://pbpython.com/tag/oauth2.html) (<http://pbpython.com/tag/oauth2.html>)  [cases](http://pbpython.com/tag/cases.html) (<http://pbpython.com/tag/cases.html>)  [vcs](http://pbpython.com/tag/vcs.html) (<http://pbpython.com/tag/vcs.html>)  [word](http://pbpython.com/tag/word.html) (<http://pbpython.com/tag/word.html>)  [matplotlib](http://pbpython.com/tag/matplotlib.html) (<http://pbpython.com/tag/matplotlib.html>)  [jinja](http://pbpython.com/tag/jinja.html) (<http://pbpython.com/tag/jinja.html>)  [barnum](http://pbpython.com/tag/barnum.html) (<http://pbpython.com/tag/barnum.html>)  [scikit-learn](http://pbpython.com/tag/scikit-learn.html) (<http://pbpython.com/tag/scikit-learn.html>)  [notebooks](http://pbpython.com/tag/notebooks.html) (<http://pbpython.com/tag/notebooks.html>)  [xlsxwriter](http://pbpython.com/tag/xlsxwriter.html) (<http://pbpython.com/tag/xlsxwriter.html>)  [csv](http://pbpython.com/tag/csv.html) (<http://pbpython.com/tag/csv.html>)  [github](http://pbpython.com/tag/github.html) (<http://pbpython.com/tag/github.html>)  [pdf](http://pbpython.com/tag/pdf.html) (<http://pbpython.com/tag/pdf.html>)

Feeds

 [Atom Feed \(http://pbpython.com/feeds/all.atom.xml\)](http://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 (<http://Amazon.com>) and affiliated sites.