A Step-by-Step Guide to Pandas Pivot Tables

The Ultimate Guide for Python and Pandas

Let's get started! Source: Nik Piepenbreier

You may be familiar with pivot tables in Excel to generate easy insights into your data. In this post, we'll explore how to create Python pivot tables using the *pivot table* function available in Pandas. The function itself is quite easy to use, but it's not the most intuitive. This post will give you a complete overview of how to use the function!

The function is quite similar to the group by function also available in Pandas, but offers significantly more customization, as we'll see later on in this post. If you

thought this post was about undoing your hard work to pivot your data, check out this tutorial on un-pivoting your data!

If you want to follow along with a video tutorial, check out my tutorial here:

Python Pivot Tables. Source: Nik Piepenbreier

How to Build a Pivot Table in Python

First off, let's quickly cover off what a pivot table actually is: it's a table of statistics that helps summarize the data of a larger table by "pivoting" that data. In Pandas, we can construct a pivot table using the following syntax, as described in the official Pandas documentation:

The function returns its own dataframe that can be accessed similar to any other dataframe you may come across. Let's take a moment to explore the different parameters available in the function:

Now that we have an understanding of the different parameters available in the function, let's load in our data set and begin exploring our data.

Let's start off by loading our data set! We've built a free downloadable data set that can be found at this link. We'll use Pandas to import the data into a dataframe called ***df***. We'll also print out the first five rows using the **.head()** function:

```
Out:
  Date       Region Type                  Units
0 2020-07-11 East   Children's Clothing  18
1 2020-09-23 North  Children's Clothing  14
2 2020-04-02 South  Women's Clothing     17
3 2020-02-28 East   Children's Clothing  26
4 2020-03-19 West   Women's Clothing     3
```

Based on the output of the first five rows shown above, we can see that we have five columns to work with:

Now that we have a bit more context around the data, let's explore creating our first pivot table in Pandas.

Creating a Pivot Table in Pandas

To get started with creating a pivot table in Pandas, let's build a very simple pivot table to start things off. We'll begin by aggregating the Sales values by the Region the sale took place in:

This returns the following output:

```
Sales   Region
East    408.182482
North   438.924051
South   432.956204
West    452.029412
```

This gave us a summary of the **Sales** field by **Region**.

Recall from our discussion around parameters earlier, that the default parameter for **aggfunc** is **mean**. Because of this, the **Sales** field in the resulting dataframe is the average of **Sales** per **Region**.

If we wanted to change the type of function used, we could use the **aggfunc** parameter. For example, if we wanted to return the **sum** of all **Sales** across a **region**, we could write:

This returns:

```
Sales     Region
East      167763
North     138700
South     59315
West      61476
```

Multi-Index Dataframes in Pandas

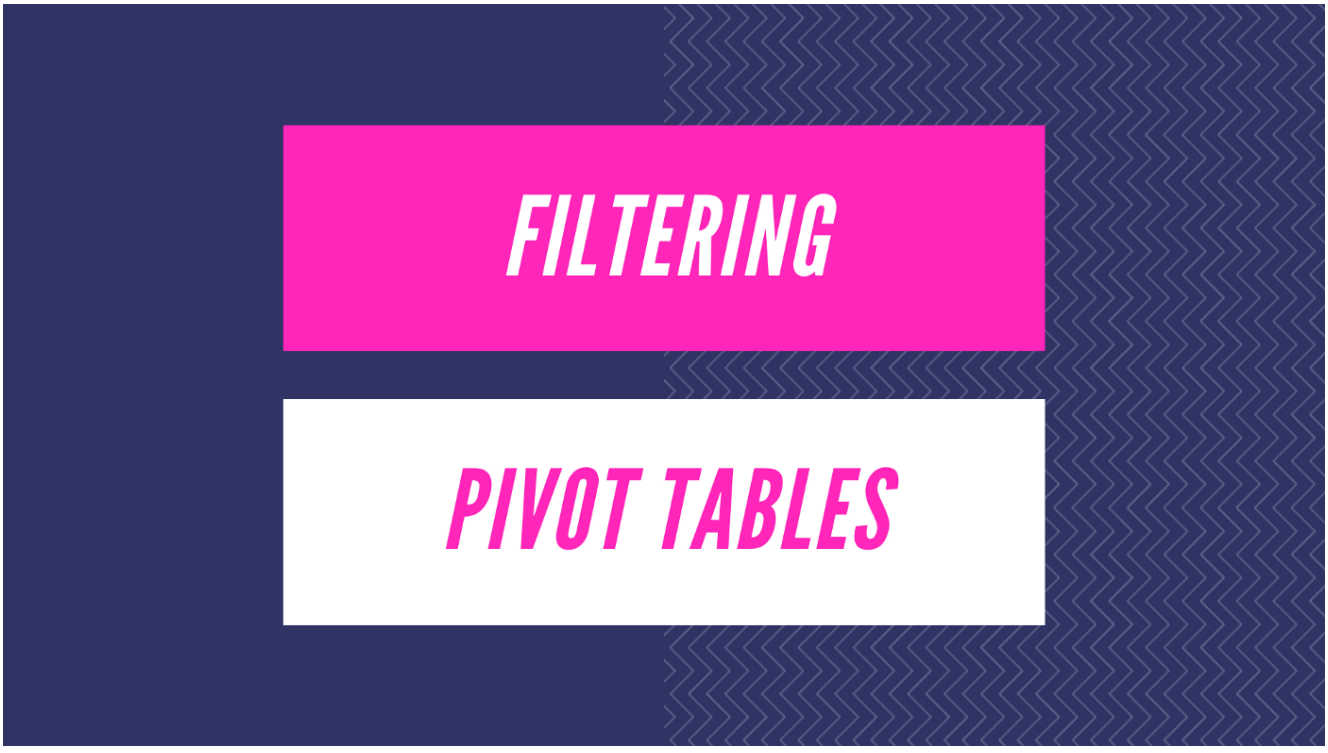
We can already notice a difference between the dataframe that this function put out, compared to the original dataframe (***df***) we put together. This is because the resulting dataframe from a pivot table function is a MultiIndex dataframes. You can learn more about these dataframes at this link.

As this dataframe is a very simple dataframe, we can simply reset the index to turn it into a normal dataframe:

Doing this resets the index and returns the following:

```
  Region   Sales
0 East     408.182482
1 North    438.924051
2 South    432.956204
3 West     452.029412
```

Filtering Python Pivot Tables



Let's filter our data! Source: Nik Piepenbreier

Now that we've created our first few pivot tables, let's explore how to filter the data. Let's create a dataframe that generates the **mean Sale** price by **Region**:

The values in this dataframe are:

```
Sales    Region
```

```
East     408.182482
North    438.924051
South    432.956204
West     452.029412
```

Now, say we wanted to filter the dataframe to only include **Regions** where the average sale price was over 450, we could write:

```
Sales    Region
West     452.029412
```

We can also apply multiple conditions, such as filtering to show only **sales** greater than 450 or less than 430. I know this is a strange example, but it's just illustrative:

Notice that we wrapped each condition in brackets and separated the conditions by a **pipe** (|) symbol. This returns the following:

```
Sales    Region
East     408.182482
West     452.029412
```

Creating a Multi-Index Pivot Table

Single index pivot tables are great for generating high-level overviews. However, we can also add additional indices to a pivot table to create further groupings. Say,

we wanted to calculate the sum per Type and per Region, we could write the following:

This will give us the following:

Region	Type	Sales
East	Children's Clothing	45849
	Men's Clothing	51685
	Women's Clothing	70229
North	Children's Clothing	37306
	Men's Clothing	39975
	Women's Clothing	61419
South	Children's Clothing	18570
	Men's Clothing	18542
	Women's Clothing	22203
West	Children's Clothing	20182
	Men's Clothing	19077
	Women's Clothing	22217

We could also apply multiple functions to our pivot table. Say we wanted to have the same pivot table that showed us the total sum but also the count of sales, we could write:

		sum	len
		Sales	Sales
Region	Type		
East	Children's Clothing	45849	113
	Men's Clothing	51685	122
	Women's Clothing	70229	176

```
North      Children's Clothing    37306    85
           Men's Clothing         39975    89
           Women's Clothing       61419    142
South      Children's Clothing    18570    45
           Men's Clothing         18542    39
           Women's Clothing       22203    53
West       Children's Clothing    20182    42
           Men's Clothing         19077    41
           Women's Clothing       22217    53
```

Adding Columns to a Pandas Pivot Table

Adding columns to a pivot table in Pandas can add another dimension to the tables. Based on the description we provided in our earlier section, the Columns parameter allows us to add a key to aggregate by. For example, if we wanted to see the number of units sold by Type and by Region, we could write:

Let’s visualize what this looks like:

```
Region                East    North   South    We
Type
Children's Clothing   2318.0  1763.0  1017.0   78
Men's Clothing        2420.0  0.0     725.0    82
Women's Clothing      3372.0  2596.0  1056.0   10
```

This allows us to see the data in a different, more easy-to-read format. It also makes plotting the data out a little

easier, as we'll explore in the next section.

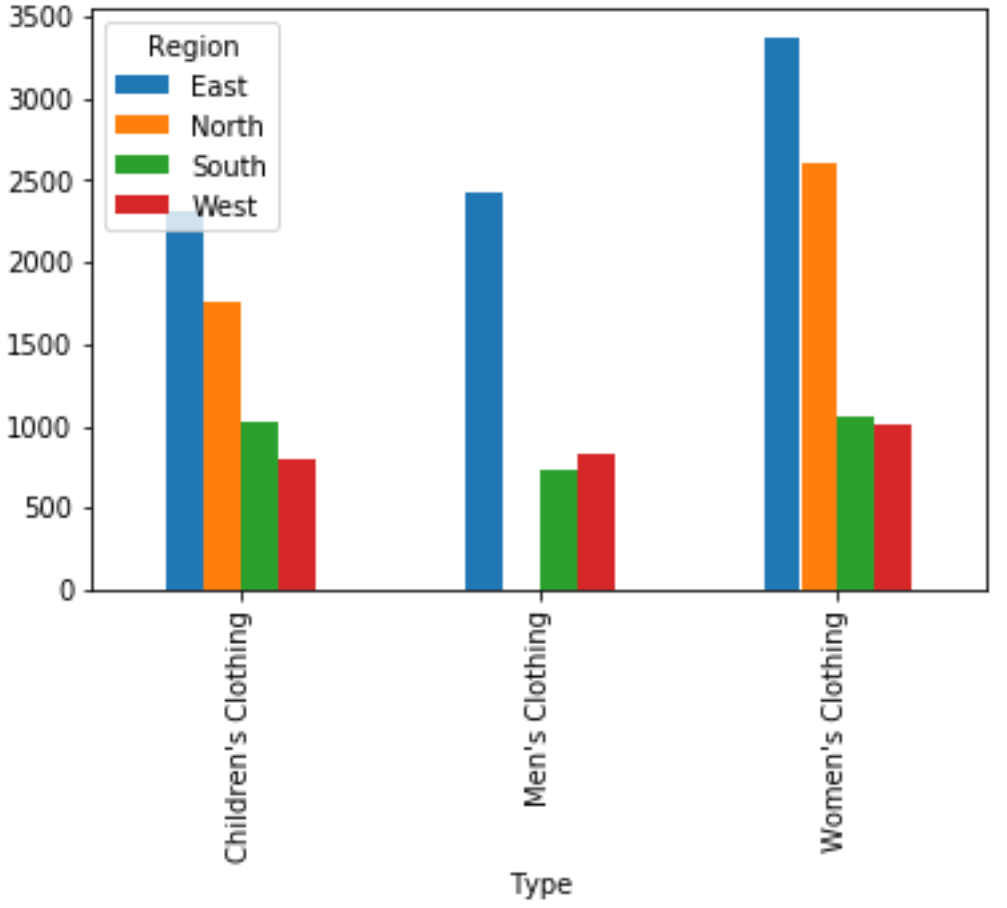
Columns might be one of the more confusing parts of the pivot table function, especially with how they relate to values. Columns are optional as we indicated above and provide the keys by which to separate the data. The pivot table aggregates the values in the values parameter.

Plotting Pandas Pivot Tables

Let's plot our data! Source: Nik Piepenbreier

While pivot tables make it easy to summarize data, plotting this data out makes it even easier to identify trends. Let's, for example, create bar plots for the pivot table we generated with columns. We can do this by writing the following:

This generates the following plot:



Graphing our pivot table. Source: Nik Piepenbreier

We can now visualize that the East region has the highest sales across Types, and that the South region has the lowest sales across Types.

If you want to find ways to make Matplotlib charts a little less ugly, check out my other tutorial here:

Visualizing COVID-19 Data Beautifully in Python (in 5 Minutes or Less!!)

Making Matplotlib a Little Less Painful!

Handling Missing Data in Python Pivot Tables

No data set is perfect! Let's see how we can handle missing data in Python pivot tables. To see which columns have missing data, we can run the info() function to explore the data set:

This returns the following output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1000 non-null   datetime64[ns]
 1   Region  1000 non-null   object
 2   Type    1000 non-null   object
 3   Units   911 non-null    float64
 4   Sales   1000 non-null   int64
dtypes: datetime64[ns](1), float64(1), int64(1),
memory usage: 39.2+ KB
```

We can see that Units is the only field with missing values. Let's see what happens when we generate this following pivot table:

This returns the following pivot table:

```
Region                  East    North   South   W
Type
Children's Clothing     35.0    35.0    35.0    3
Men's Clothing          35.0    NaN     34.0    3
Women's Clothing        35.0    35.0    35.0    3
```

The NaN displayed in the table, of course, doesn't look great. We can choose to fill that value with something else. For example, if we wanted to fill N/A for any missing values, we could write the following:

The output of this is:

```
Region                   East    North   South   W
Type
Children's Clothing      35.0    35      35.0    3
Men's Clothing           35.0    N/A     34.0    3
Women's Clothing         35.0    35      35.0    3
```

Much better!

Adding Totals for Rows and Columns to Pandas Pivot Tables

For our last section, let's explore how to add totals to both rows and columns in our Python pivot table. We do this with the margins and margins_name parameters. If we wanted to add this to the pivot table we created above, we would write the following:

This returns the following:

```
Region                   East    North   South   W
Type
Children's Clothing      35      35      35      3
Men's Clothing           35      N/A     34      3
```

```
Women's Clothing        35       35       35       3
Total                   35       35       35       3
```

The margins parameter requires a boolean (True/False) value to either add row/column totals or not. The margins_name parameter allows us to add labels to these values.

Conclusion: Python Pivot Tables — The Ultimate Guide

Thanks so much for reading! Source: Nik Piepenbreier

In this post, we explored how to easily generated a pivot table off of a given dataframe using Python and Pandas.

Pivot tables in Python allow you to easily generate insights into data sets, whether large or small. The multitude of parameters available in the pivot_table function allows for a lot of flexibility in how data is analyzed. In this post, we explored how to generate a pivot table, how to filter pivot tables in Python, how to add multiple indices and columns to pivot tables, how to plot pivot tables, how to deal with missing values, and how to add row and column totals.

Originally published at https://datagy.io on April 12, 2020.