Explore Happiness Data Using Python Pivot Tables

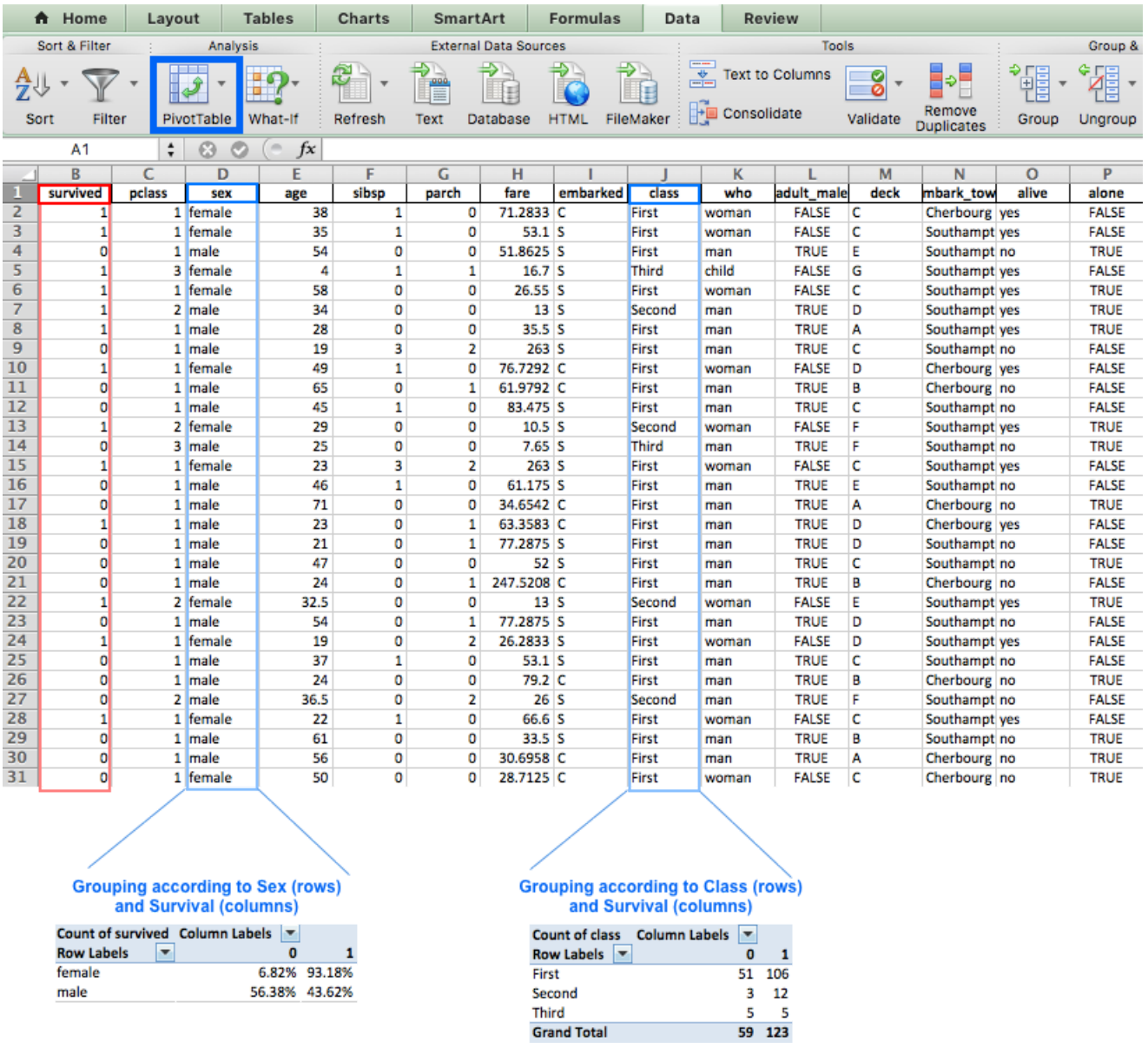
Michal Weizman | 25 SEP 2017 in python

One of the biggest challenges when facing a new data set is knowing where to start and what to focus on. Being able to quickly summarize hundreds of rows and columns can save you a lot of time and frustration. A simple tool you can use to achieve this is a pivot table, which helps you slice, filter, and group data at the speed of inquiry and represent the information in a visually appealing way.

Pivot table, what is it good for?

You may already be familiar with the concept of pivot tables from Excel, where they were introduced in 1994 by the trademarked name PivotTable. This tool enabled users to automatically sort, count, total, or average the data stored in one table. In the image below we used the PivotTable functionality to quickly summarize the Titanic data set. The larger

table below displays the first ~30 rows of the data set, and the smaller tables are the PivotTables we created.



	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	mbark_tow	alive	alone
2	1	1	female	38	1	0	71.2833	C	First	woman	FALSE	C	Cherbourg	yes	FALSE
3	1	1	female	35	1	0	53.1	S	First	woman	FALSE	C	Southampt	yes	FALSE
4	0	1	male	54	0	0	51.8625	S	First	man	TRUE	E	Southampt	no	TRUE
5	1	3	female	4	1	1	16.7	S	Third	child	FALSE	G	Southampt	yes	FALSE
6	1	1	female	58	0	0	26.55	S	First	woman	FALSE	C	Southampt	yes	TRUE
7	1	2	male	34	0	0	13	S	Second	man	TRUE	D	Southampt	yes	TRUE
8	1	1	male	28	0	0	35.5	S	First	man	TRUE	A	Southampt	yes	TRUE
9	0	1	male	19	3	2	263	S	First	man	TRUE	C	Southampt	no	FALSE
10	1	1	female	49	1	0	76.7292	C	First	woman	FALSE	D	Cherbourg	yes	FALSE
11	0	1	male	65	0	1	61.9792	C	First	man	TRUE	B	Cherbourg	no	FALSE
12	0	1	male	45	1	0	83.475	S	First	man	TRUE	C	Southampt	no	FALSE
13	1	2	female	29	0	0	10.5	S	Second	woman	FALSE	F	Southampt	yes	TRUE
14	0	3	male	25	0	0	7.65	S	Third	man	TRUE	F	Southampt	no	TRUE
15	1	1	female	23	3	2	263	S	First	woman	FALSE	C	Southampt	yes	FALSE
16	0	1	male	46	1	0	61.175	S	First	man	TRUE	E	Southampt	no	FALSE
17	0	1	male	71	0	0	34.6542	C	First	man	TRUE	A	Cherbourg	no	TRUE
18	1	1	male	23	0	1	63.3583	C	First	man	TRUE	D	Cherbourg	yes	FALSE
19	0	1	male	21	0	1	77.2875	S	First	man	TRUE	D	Southampt	no	FALSE
20	0	1	male	47	0	0	52	S	First	man	TRUE	C	Southampt	no	TRUE
21	0	1	male	24	0	1	247.5208	C	First	man	TRUE	B	Cherbourg	no	FALSE
22	1	2	female	32.5	0	0	13	S	Second	woman	FALSE	E	Southampt	yes	TRUE
23	0	1	male	54	0	1	77.2875	S	First	man	TRUE	D	Southampt	no	FALSE
24	1	1	female	19	0	2	26.2833	S	First	woman	FALSE	D	Southampt	yes	FALSE
25	0	1	male	37	1	0	53.1	S	First	man	TRUE	C	Southampt	no	FALSE
26	0	1	male	24	0	0	79.2	C	First	man	TRUE	B	Cherbourg	no	TRUE
27	0	2	male	36.5	0	2	26	S	Second	man	TRUE	F	Southampt	no	FALSE
28	1	1	female	22	1	0	66.6	S	First	woman	FALSE	C	Southampt	yes	FALSE
29	0	1	male	61	0	0	33.5	S	First	man	TRUE	B	Southampt	no	TRUE
30	0	1	male	56	0	0	30.6958	C	First	man	TRUE	A	Cherbourg	no	TRUE
31	0	1	female	50	0	0	28.7125	C	First	woman	FALSE	C	Cherbourg	no	TRUE

The pivot table on the left grouped the data according to the `Sex` and `Survived` column. As a result, this table displays the percentage of each gender among the different survival status (`0`: Didn't survive, `1`: Survived). This allows us to quickly see that women had better chances of survival than men. The table on the right also uses the `Survived` column, but this time the data is grouped by `Class`.

Introducing our data set: World Happiness Report

We used Excel for the above examples, but this post will demonstrate the advantages of the built-in pandas function pivot_table built in function in Pandas. We'll use the World Happiness Report, which is a survey about the state of global happiness. The report ranks

more than 150 countries by their happiness levels, and has been published almost every year since 2012. We'll use data collected in the years 2015, 2016, and 2017, which is available for download if you'd like to follow along. We're running python 3.6 and pandas 0.19.

Some interesting questions we might like to answer are:

- Which are the happiest and least happy countries and regions in the world?
- Is happiness affected by region?
- Did the happiness score change significantly over the past three years?

Let's import our data and take a quick first look:

```
import pandas as pd
import numpy as np
# reading the data
data = pd.read_csv('data.csv', index_col=0)
# sort the df by ascending years and descending happiness scores
data.sort_values(['Year', "Happiness Score"], ascending=[True, False], inplace=
#diplay first 10 rows
data.head(10)
```

	Country	Region	Happiness Rank	Happiness Score	Ecor
141	Switzerland	Western Europe	1.0	7.587	1.396
60	Iceland	Western Europe	2.0	7.561	1.302
38	Denmark	Western Europe	3.0	7.527	1.325
108	Norway	Western Europe	4.0	7.522	1.459
25	Canada	North America	5.0	7.427	1.326
46	Finland	Western Europe	6.0	7.406	1.290
102	Netherlands	Western Europe	7.0	7.378	1.329
140	Sweden	Western Europe	8.0	7.364	1.331
103	New Zealand	Australia and New Zealand	9.0	7.286	1.250
6	Australia	Australia and New Zealand	10.0	7.284	1.333

Each country's `Happiness Score` is calculated by summing the seven other variables in the table. Each of these variables reveals a population-weighted average score on a scale running from 0 to 10, that is tracked over time and compared against other countries.

These variables are:

- `Economy`: real GDP per capita
- `Family`: social support
- `Health`: healthy life expectancy
- `Freedom`: freedom to make life choices
- `Trust`: perceptions of corruption
- `Generosity`: perceptions of generosity
- `Dystopia`: each country is compared against a hypothetical nation that represents the lowest national averages for each key variable and is, along with residual error, used as a regression benchmark

Each country's `Happiness Score` determines its `Happiness Rank` - which is its relative position among other countries in that specific year. For example, the first row indicates that Switzerland was ranked the happiest country in 2015 with a happiness score of 7.587. Switzerland was ranked first just before Iceland, which scored 7.561. Denmark was ranked third in 2015, and so on. It's interesting to note that Western Europe took seven of the top eight rankings in 2015.

We'll concentrate on the final `Happiness Score` to demonstrate the technical aspects of pivot table.

```
# getting an overview of our data
print("Our data has {0} rows and {1} columns".format(data.shape[0], data.shape[
# checking for missing values
print("Are there missing values? {}".format(data.isnull().any().any()))
data.describe()
```

```
Our data has 495 rows and 12 columns
Are there missing values? True
```

	Happiness Rank	Happiness Score	Economy (GDP per Capita)	Family	He
count	470.000000	470.000000	470.000000	470.000000	47
mean	78.829787	5.370728	0.927830	0.990347	0.5
std	45.281408	1.136998	0.415584	0.318707	0.2
min	1.000000	2.693000	0.000000	0.000000	0.0
25%	40.000000	4.509000	0.605292	0.793000	0.4
50%	79.000000	5.282500	0.995439	1.025665	0.6
75%	118.000000	6.233750	1.252443	1.228745	0.7
max	158.000000	7.587000	1.870766	1.610574	1.0

The describe() method reveals that `Happiness Rank` ranges from 1 to 158, which means that the largest number of surveyed countries for a given year was 158. It's worth noting that `Happiness Rank` was originally of type `int`. The fact it's displayed as a float here implies we have `NaN` values in this column (we can also determine this by the `count` row which only amounts to 470 as opposed to the 495 rows in our data set).

The `Year` column doesn't have any missing values. Firstly, because it's displayed in the data set as `int`, but also - the count for `Year` amounts to 495 which is the number of rows in our data set. By comparing the `count` value for `Year` to the other columns, it seems we can expect 25 missing values in each column (495 in `Year` VS. 470 in all other columns).

Categorizing the data by `Year` and `Region`

The fun thing about pandas `pivot_table` is you can get another point of view on your data with only one line of code. Most of the `pivot_table` parameters use default values, so the only mandatory parameters you must add are `data` and `index`. Though it isn't mandatory, we'll also use the `value` parameter in the next example.

- `data` is self explanatory - it's the DataFrame you'd like to use

- `index` is the column, grouper, array (or list of the previous) you'd like to group your data by. It will be displayed in the index column (or columns, if you're passing in a list)
- `values` (optional) is the column you'd like to aggregate. If you do not specify this then the function will aggregate all numeric columns.

Let's first look at the output, and then explain how the table was produced:

```
pd.pivot_table(data, index= 'Year', values= "Happiness Score")
```

	Happiness Score
Year	
2015	5.375734
2016	5.382185
2017	5.354019

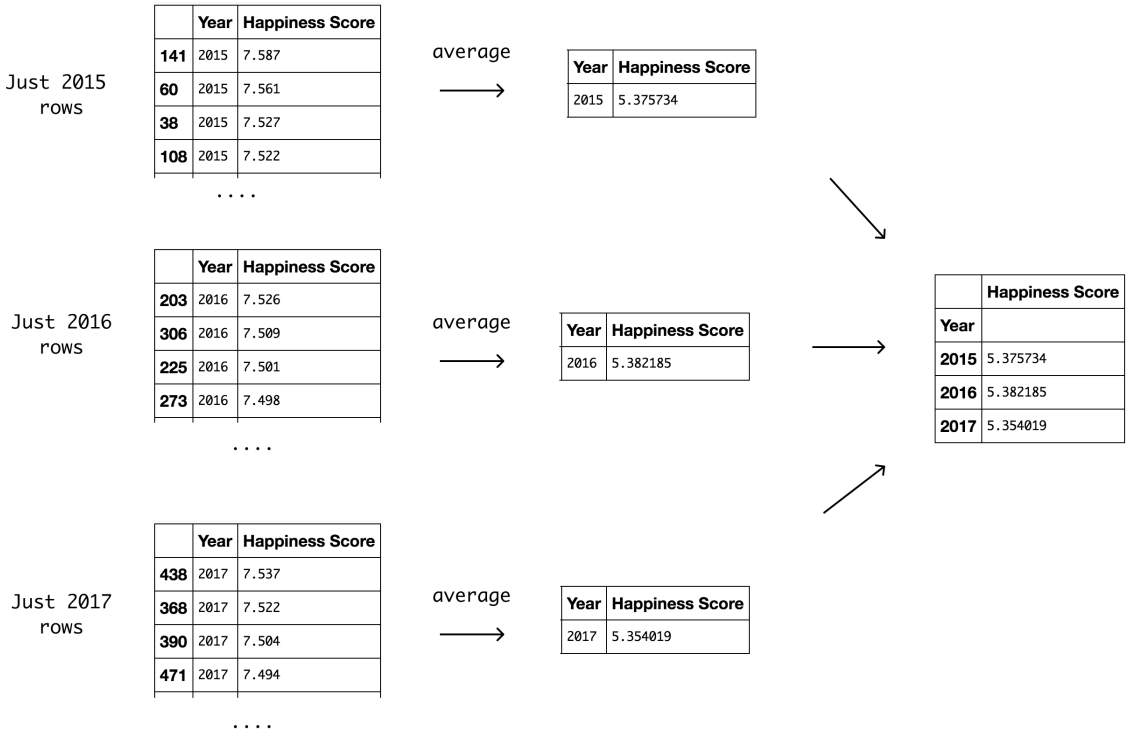
By passing `Year` as the `index` parameter, we chose to group our data by `Year`. The output is a pivot table that displays the three different values for `Year` as `index`, and the `Happiness Score` as `values`. It's worth noting that the aggregation default value is mean (or average), so the values displayed in the `Happiness Score` column are the yearly average for all countries. The table shows the average for all countries was highest in 2016, and is currently the lowest in the past three years.

Here's a detailed diagram of how this pivot table was created:

```
data[['Country', 'Year', 'Happiness Score']]
```

	Country	Year	Happiness Score
141	Switzerland	2015	7.587
60	Iceland	2015	7.561
38	Denmark	2015	7.527
108	Norway	2015	7.522
25	Canada	2015	7.427
46	Finland	2015	7.406
102	Netherlands	2015	7.378
140	Sweden	2015	7.364
103	New Zealand	2015	7.286

```
pd.pivot_table(data, index= 'Year', values= "Happiness Score")
```



Next, let's use the `Region` column as `index`:

```
pd.pivot_table(data, index = 'Region', values="Happiness Score")
```

	Happiness Score
Region	
Australia and New Zealand	7.302500
Central and Eastern Europe	5.371184
Eastern Asia	5.632333
Latin America and Caribbean	6.069074
Middle East and Northern Africa	5.387879
North America	7.227167
Southeastern Asia	5.364077
Southern Asia	4.590857
Sub-Saharan Africa	4.150957
Western Europe	6.693000

The numbers displayed in the `Happiness Score` column in the pivot table above are the mean, just as before - but this time it's each region's mean for all years documented (2015, 2016, 2017). This display makes it easier to see `Australia and New Zealand` have the highest average score, while `North America` is ranked close behind. It's interesting that despite the initial impression we got from reading the data, which showed `Western Europe` in most of the top places, `Western Europe` is actually ranked third when calculating the average for the past three years. The lowest ranked region is `Sub-Saharan Africa`, and close behind is `Southern Asia`.

Creating a multi-index pivot table

You may have used `groupby()` to achieve some of the pivot table functionality (we've previously demonstrated how to use groupby() to analyze your data). However, the `pivot_table()` built-in function offers straightforward parameter names and default values that can help simplify complex procedures like multi-indexing.

In order to group the data by more than one column, all we have to do is pass in a list of column names. Let's categorize the data by `Region` and `Year`.

```
pd.pivot_table(data, index = ['Region', 'Year'], values="Happiness Score")
```

		Happiness Score
Region	**Year**	
Australia and New Zealand	**2015**	7.285000
	2016	7.323500
	2017	7.299000
Central and Eastern Europe	**2015**	5.332931
	2016	5.370690
	2017	5.409931
Eastern Asia	**2015**	5.626167
	2016	5.624167
	2017	5.646667
Latin America and Caribbean	**2015**	6.144682
	2016	6.101750
	2017	5.957818
Middle East and Northern Africa	**2015**	5.406900
	2016	5.386053
	2017	5.369684
North America	**2015**	7.273000
	2016	7.254000
	2017	7.154500
Southeastern Asia	**2015**	5.317444
	2016	5.338889
	2017	5.444875
Southern Asia	**2015**	4.580857
	2016	4.563286
	2017	4.628429
Sub-Saharan Africa	**2015**	4.202800
	2016	4.136421
	2017	4.111949
Western Europe	**2015**	6.689619
	2016	6.685667
	2017	6.703714

These examples also reveal where pivot table got its name from: it allows you to rotate or pivot the summary table, and this rotation gives us a different perspective of the data. A perspective that can very well help you quickly gain valuable insights.

This is one way to look at the data, but we can use the `columns` parameter to get a better display:

- `columns` is the column, grouper, array, or list of the previous you'd like to group your data by. Using it will spread the different values horizontally.

Using `Year` as the `Columns` argument will display the different values for `year`, and will make for a much better display, like so:

```
pd.pivot_table(data, index= 'Region', columns='Year', values="Happiness Score")
```

Year	2015	2016	2017
Region			
Australia and New Zealand	7.285000	7.323500	7.299000
Central and Eastern Europe	5.332931	5.370690	5.409931
Eastern Asia	5.626167	5.624167	5.646667
Latin America and Caribbean	6.144682	6.101750	5.957818
Middle East and Northern Africa	5.406900	5.386053	5.369684
North America	7.273000	7.254000	7.154500
Southeastern Asia	5.317444	5.338889	5.444875
Southern Asia	4.580857	4.563286	4.628429
Sub-Saharan Africa	4.202800	4.136421	4.111949
Western Europe	6.689619	6.685667	6.703714

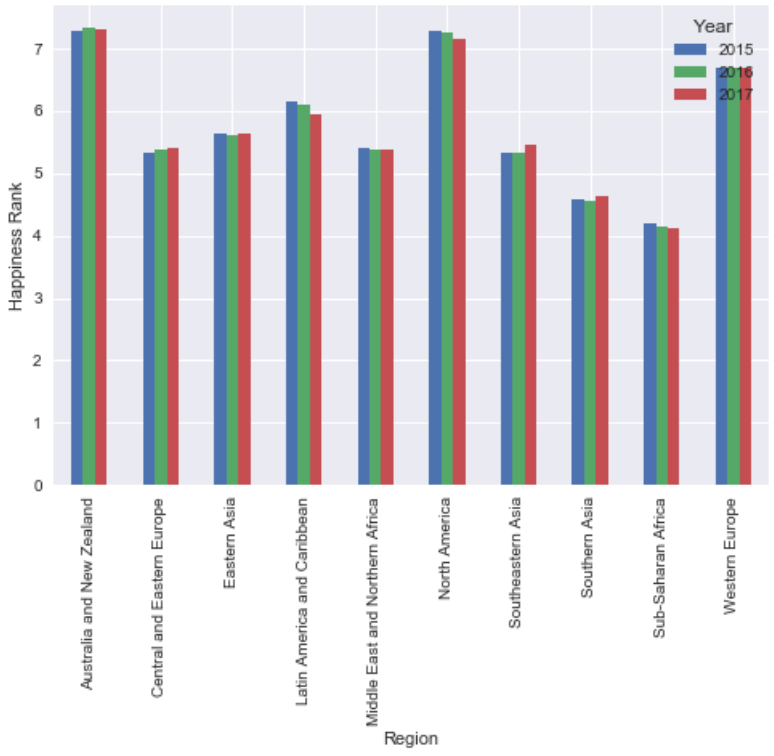
Visualizing the pivot table using `plot()`

If you want to look at the visual representation of the previous pivot table we created, all you need to do is add `plot()` at the end of the `pivot_table` function call (you'll also need to import the relevant plotting libraries).

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
# use Seaborn styles
sns.set()

pd.pivot_table(data, index= 'Region', columns= 'Year', values= "Happiness Score
plt.ylabel("Happiness Rank")
```

```
<matplotlib.text.Text at 0x11b885630>
```



The visual representation helps reveal that the differences are minor. Having said that, this also shows that there's a permanent decrease in the Happiness rank of both of the regions located in America.

Manipulating the data using `aggfunc`

Up until now we've used the average to get insights about the data, but there are other important values to consider. Time to experiment with the `aggfunc` parameter:

- `aggfunc` (optional) accepts a function or list of functions you'd like to use on your group (default: `numpy.mean`). If a list of functions is passed, the resulting pivot table will have hierarchical columns whose top level are the function names.

Let's add the median, minimum, maximum, and the standard deviation for each region. This can help us evaluate how accurate the average is, and if it's really representative of the real picture.

```
pd.pivot_table(data, index= 'Region', values= "Happiness Score",
               aggfunc= [np.mean, np.median, min, max, np.std])
```

	mean	median	min
	Happiness Score	Happiness Score	Happiness Score
Region			
Australia and New Zealand	7.302500	7.2995	7.284
Central and Eastern Europe	5.371184	5.4010	4.096
Eastern Asia	5.632333	5.6545	4.874
Latin America and Caribbean	6.069074	6.1265	3.603
Middle East and Northern Africa	5.387879	5.3175	3.006
North America	7.227167	7.2175	6.993
Southeastern Asia	5.364077	5.2965	3.819
Southern Asia	4.590857	4.6080	3.360
Sub-Saharan Africa	4.150957	4.1390	2.693
Western Europe	6.693000	6.9070	4.857

Looks like some regions have extreme values that might affect our average more than we'd like them to. For example, `Middle East and Northern Africa` region have a high standard deviation, so we might want to remove extreme values. Let's see how many values we're calculating for each region. This might affect the representation we're seeing. For example, `Australia and new Zealand` have a very low standard deviation and are ranked happiest for all three years, but we can also assume they only account for two countries.

Applying a custom function to remove outliers

`pivot_table` allows you to pass your own custom aggregation functions as arguments. You can either use a lambda function, or create a function. Let's calculate the average number of countries in each region in a given year. We can do this easily using a lambda function, like so:

```
pd.pivot_table(data, index = 'Region', values="Happiness Score",
               aggfunc= [np.mean, min, max, np.std, lambda x: x.count()/3])
```

	mean	min	max
	Happiness Score	Happiness Score	Happiness Score
Region			
Australia and New Zealand	7.302500	7.284	7.334
Central and Eastern Europe	5.371184	4.096	6.609
Eastern Asia	5.632333	4.874	6.422
Latin America and Caribbean	6.069074	3.603	7.226
Middle East and Northern Africa	5.387879	3.006	7.278
North America	7.227167	6.993	7.427
Southeastern Asia	5.364077	3.819	6.798
Southern Asia	4.590857	3.360	5.269
Sub-Saharan Africa	4.150957	2.693	5.648
Western Europe	6.693000	4.857	7.587

Both highest ranking regions with the lowest standard deviation only account for only two countries. `Sub-Saharan Africa`, on the other hand, has the lowest `Happiness score`, but it accounts for 43 countries. An interesting next step would be to remove extreme values from the calculation to see if the ranking changes significantly. Let's create a function that only calculates the values that are between the 0.25th and 0.75th quantiles. We'll use this function as a way to calculate the average for each region and check if the ranking stays the same or not.

```
def remove_outliers(values):
    mid_quantiles = values.quantile([.25, .75])
    return np.mean(mid_quantiles)

pd.pivot_table(data, index = 'Region', values="Happiness Score",
               aggfunc= [np.mean, remove_outliers, lambda x: x.count()/3])
```

	mean	remove_outliers	<lambda>
	Happiness Score	Happiness Score	Happiness Scor
Region			
Australia and New Zealand	7.302500	7.299125	2.000000
Central and Eastern Europe	5.371184	5.449250	29.000000
Eastern Asia	5.632333	5.610125	6.000000
Latin America and Caribbean	6.069074	6.192750	22.666667
Middle East and Northern Africa	5.387879	5.508500	19.333333
North America	7.227167	7.244875	2.000000
Southeastern Asia	5.364077	5.470125	8.666667
Southern Asia	4.590857	4.707500	7.000000
Sub-Saharan Africa	4.150957	4.128000	39.000000
Western Europe	6.693000	6.846500	21.000000

Removing the outliers mostly affected the regions with a higher number of countries, which makes sense. We can see `Western Europe` (average of 21 countries surveyed per year) improved its ranking. Unfortunately, `Sub-Saharan Africa` (average of 39 countries surveyed per year) received an even lower ranking when we removed the outliers.

Categorizing using string manipulation

Up until now we've grouped our data according to the categories in the original table. However, we can search the strings in the categories to create our own groups. For example, it would be interesting to look at the results by continents. We can do this by looking for region names that contains `Asia`, `Europe`, etc. To do this, we can first assign our pivot table to a variable, and then add our filter:

```
table = pd.pivot_table(data, index = 'Region', values="Happiness Score",
               aggfunc= [np.mean, remove_outliers])

table[table.index.str.contains('Asia')]
```

	mean	remove_outliers
	Happiness Score	Happiness Score
Region		
Eastern Asia	5.632333	5.610125
Southeastern Asia	5.364077	5.470125
Southern Asia	4.590857	4.707500

Let's see the results for `Europe`:

```
table[table.index.str.contains('Europe')]
```

	mean	remove_outliers
	Happiness Score	Happiness Score
Region		
Central and Eastern Europe	5.371184	5.44925
Western Europe	6.693000	6.84650

The difference shows that the two European regions have a larger difference in happiness score. In most cases, removing outliers makes the score higher, but not in Eastern Asia.

If you'd like to extract specific values from more than one column, then it's better to use `df.query` because the previous method won't work for conditioning multi-indexes. For example, we can choose to view specific years, and specific regions in the Africa area.

```
table = pd.pivot_table(data, index = ['Region', 'Year'], values='Happiness Scor
                aggfunc= [np.mean, remove_outliers])

table.query('Year == [2015, 2017] and Region == ["Sub-Saharan Africa", "Middle
```

		mean	remove_outliers
		Happiness Score	Happiness Score
Region	**Year**		
Middle East and Northern Africa	**2015**	5.406900	5.515875
	2017	5.369684	5.425500
Sub-Saharan Africa	**2015**	4.202800	4.168375
	2017	4.111949	4.118000

In this example the differences are minor, but an interesting exercise would be to compare information from previous years since the survey has reports since 2012.

Handling missing data

We've covered the most powerful parameters of `pivot_table` thus far, so you can already get a lot out of it if you go experiment using this method on your own project. Having said that, it's useful to quickly go through the remaining parameters (which are all optional and have default values). The first thing to talk about is missing values.

- `dropna` is type boolean, and used to indicate you do not want to include columns whose entries are all `NaN` (default: True)
- `fill_value` is type scalar, and used to choose a value to replace missing values (default: None).

We don't have any columns where all entries are `NaN`, but it's worth knowing that if we did `pivot_table` would drop them by default according to `dropna` definition.

We have been letting `pivot_table` treat our `NaN`'s according to the default settings. The `fill_value` default value is `None` so this means we didn't replace missing values in our Data set. To demonstrate this we'll need to produce a pivot table with `NaN` values. We can split the `Happiness Score` of each region into three quantiles, and check how many countries fall into each of the three quantiles (hoping at least one of the quantiles will have missing values in it).

To do this, we'll use `qcut()`, which is a built-in pandas function that allows you to split your data into any number of quantiles you choose. For example, specifying `pd.qcut(data["Happiness Score"], 4)` will result in four quantiles:

- 0-25%
- 25%-50%
- 50%-75%
- 75%-100%

```
# splitting the happiness score into 3 quantiles
score = pd.qcut(data["Happiness Score"], 4)
pd.pivot_table(data, index= ['Region', score], values= "Happiness Score", aggfu
```

		Happiness Score
Region	**Happiness Score**	
Australia and New Zealand	**(2.692, 4.509]**	NaN
	(4.509, 5.283]	NaN
	(5.283, 6.234]	NaN
	(6.234, 7.587]	6.0
Central and Eastern Europe	**(2.692, 4.509]**	10.0
	(4.509, 5.283]	28.0
	(5.283, 6.234]	46.0
	(6.234, 7.587]	3.0
Eastern Asia	**(2.692, 4.509]**	NaN

Regions where there are no countries in a specific quantile show `NaN`. This isn't ideal because a count that equals NaN doesn't give us any useful information. It's less confusing to display `0`, so let's substitute `NaN` by zeros using `fill_value`:

```
# splitting the happiness score into 3 quantiles
score = pd.qcut(data["Happiness Score"], 3)
pd.pivot_table(data, index= ['Region', score], values= "Happiness Score", aggfu
               fill_value= 0)
```

		Happiness Score
Region	**Happiness Score**	
Australia and New Zealand	**(2.692, 4.79]**	0
	(4.79, 5.895]	0
	(5.895, 7.587]	6
Central and Eastern Europe	**(2.692, 4.79]**	15
	(4.79, 5.895]	58
	(5.895, 7.587]	14
Eastern Asia	**(2.692, 4.79]**	0
	(4.79, 5.895]	11
	(5.895, 7.587]	7
Latin America and Caribbean	**(2.692, 4.79]**	4
	(4.79, 5.895]	19
	(5.895, 7.587]	45
Middle East and Northern Africa	**(2.692, 4.79]**	18
	(4.79, 5.895]	20
	(5.895, 7.587]	20
North America	**(2.692, 4.79]**	0
	(4.79, 5.895]	0
	(5.895, 7.587]	6
Southeastern Asia	**(2.692, 4.79]**	6
	(4.79, 5.895]	12
	(5.895, 7.587]	8
Southern Asia	**(2.692, 4.79]**	13
	(4.79, 5.895]	8
	(5.895, 7.587]	0
Sub-Saharan Africa	**(2.692, 4.79]**	101
	(4.79, 5.895]	16
	(5.895, 7.587]	0
Western Europe	**(2.692, 4.79]**	0
	(4.79, 5.895]	12
	(5.895, 7.587]	51

Adding total rows/columns

The last two parameters are both optional and mostly useful to improve display:

- `margins` is type boolean and allows you to add an `all` row / columns, e.g. for subtotal / grand totals (Default False)
- `margins_name` which is type string and accepts the name of the row / column that will contain the totals when margins is True (default 'All')

Let's use these to add a total to our last table.

```
# splitting the happiness score into 3 quantiles
score = pd.qcut(data['Happiness Score'], 3)
# creating a pivot table and only displaying the first 9 values
pd.pivot_table(data, index= ['Region', score], values= "Happiness Score", aggfu
               margins = True, margins_name= 'Total count')
```

		Happiness Score
Region	**Happiness Score**	
Australia and New Zealand	**(2.692, 4.79]**	0.0
	(4.79, 5.895]	0.0
	(5.895, 7.587]	6.0
Central and Eastern Europe	**(2.692, 4.79]**	15.0
	(4.79, 5.895]	58.0
	(5.895, 7.587]	14.0
Eastern Asia	**(2.692, 4.79]**	0.0
	(4.79, 5.895]	11.0
	(5.895, 7.587]	7.0
Latin America and Caribbean	**(2.692, 4.79]**	4.0
	(4.79, 5.895]	19.0
	(5.895, 7.587]	45.0
Middle East and Northern Africa	**(2.692, 4.79]**	18.0
	(4.79, 5.895]	20.0
	(5.895, 7.587]	20.0
North America	**(2.692, 4.79]**	0.0
	(4.79, 5.895]	0.0
	(5.895, 7.587]	6.0
Southeastern Asia	**(2.692, 4.79]**	6.0
	(4.79, 5.895]	12.0
	(5.895, 7.587]	8.0
Southern Asia	**(2.692, 4.79]**	13.0
	(4.79, 5.895]	8.0
	(5.895, 7.587]	0.0
Sub-Saharan Africa	**(2.692, 4.79]**	101.0
	(4.79, 5.895]	16.0
	(5.895, 7.587]	0.0
Western Europe	**(2.692, 4.79]**	0.0
	(4.79, 5.895]	12.0
	(5.895, 7.587]	51.0
Total count		470.0

Let's summarize

If you're looking for a way to inspect your data from a different perspective then `pivot_table` is the answer. It's easy to use, it's useful for both numeric and categorical values, and it can get you results in one line of code.

If you enjoyed digging into this data and you're interested in investigating this further then we suggest adding survey results from previous years, and/or combining additional columns with countries information like: poverty, terror, unemployment, etc. Feel free to share your notebook in the comments below, and enjoy your learning!

Join 40,000+ Data Scientists: Get Data Science Tips, Tricks and Tutorials, Delivered Weekly.

Enter your email address Sign me up!

Michal Weizman

Student Success Engineer at Dataquest.io (Learn Data Science in your Browser).
Online learning and Data enthusiast.
Get in touch @hakabuk.

Share this post