03-05 Pivot Table

A pivot table is a way of summarizing data in a DataFrame for a particular purpose. **It makes heavy use of the aggregation function.**

A pivot table is itself a DataFrame, where the rows represent **one variable** that you're interested in, **the columns another**, and the cell's some aggregate value. A pivot table also tends to **includes marginal values** as well, which are the sums for each column and row. This allows you to be able to see the relationship between two variables at just a glance.

Creating Pivot Tables in Pandas

In [1]:

```
# Lets take a look at pivot tables in pandas
import pandas as pd
import numpy as np
```

In [2]:

```
# Here we have the Times Higher Education World University Ranking dataset, whic
h is one of the most
# influential university measures. Let's import the dataset and see what it look
s like
df = pd.read_csv('datasets/cwurData.csv')
df.head()
```

Out[2]:

	world_rank	institution	country	national_rank	quality_of_education	alumni_employmer
0	1	Harvard University	USA	1	7	
1	2	Massachusetts Institute of Technology	USA	2	9	1
2	3	Stanford University	USA	3	17	1
3	4	University of Cambridge	United Kingdom	1	10	2
4	5	California Institute of Technology	USA	4	2	2

Here we can see each institution's rank, country, quality of education, other metrics, and overall score. Let's say we want to create a new column called Rank_Level, where institutions with world ranking 1-100 are categorized as first tier and those with world ranking 101 - 200 are second tier, ranking 201 - 300 are third tier, after 301 is other top universities.

In [3]:

```
def create_category(ranking):
    # Since the rank is just an integer, I'll just do a bunch of if/elif stateme
nts
    if (ranking >= 1) & (ranking <= 100):
        return "First Tier Top Unversity"
    elif (ranking >= 101) & (ranking <= 200):
        return "Second Tier Top Unversity"
    elif (ranking >= 201) & (ranking <= 300):
        return "Third Tier Top Unversity"
    return "Other Top Unversity"

# Now we can apply this to a single column of data to create a new series
df['Rank_Level'] = df['world_rank'].apply(lambda x: create_category(x))
# And lets look at the result
df.head()
```

Out[3]:

	world_rank	institution	country	national_rank	quality_of_education	alumni_employmer
0	1	Harvard University	USA	1	7	
1	2	Massachusetts Institute of Technology	USA	2	9	1
2	3	Stanford University	USA	3	17	1
3	4	University of Cambridge	United Kingdom	1	10	2
4	5	California Institute of Technology	USA	4	2	2

In [4]:

```
# A pivot table allows us to pivot out one of these columns a new column headers and compare it against
# another column as row indices. Let's say we want to compare rank level versus country of the universities
# and we want to compare in terms of overall score

# To do this, we tell Pandas we want the values to be Score, and index to be the country and the columns to be
# the rank levels. Then we specify that the aggregation function, and here we'll use the NumPy mean to get the
# average rating for universities in that country

df.pivot_table(values='score', index='country', columns='Rank_Level', aggfunc=[np.mean]).head()
```

Out[4]:

	mean			
Rank_Level	**First Tier Top Unversity**	**Other Top Unversity**	**Second Tier Top Unversity**	**Third Tier Top Unversity**
country				
Argentina	NaN	44.672857	NaN	NaN
Australia	47.9425	44.645750	49.2425	47.285000
Austria	NaN	44.864286	NaN	47.066667
Belgium	51.8750	45.081000	49.0840	46.746667
Brazil	NaN	44.499706	49.5650	NaN

We can see a hierarchical dataframe where the index, or rows, are by country and the columns have two levels, the top level indicating that the mean value is being used and the second level being our ranks. In this example we only have one variable, the mean, that we are looking at, so we don't really need a heirarchical index.

We notice that there are some NaN values, for example, the first row, Argentia. The NaN values indicate that Argentia has only observations in the "Other Top Unversities" category

Multiple Hierarchical Indices

Now, pivot tables aren't limited to one function that you might want to apply. You can pass a named parameter, `aggfunc` , which is a **list** of the different functions to apply, and pandas will provide you with the result **using hierarchical column names**. Let's try that same query, but pass in the max() function too.

This example should make *Hierarchical Column Index* clearer.

In [5]:

```
df.pivot_table(values='score', index='country', columns='Rank_Level', aggfunc=[n
p.mean, np.max]).head()
```

Out[5]:

	mean				amax			
Rank_Level	First Tier Top Unversity	Other Top Unversity	Second Tier Top Unversity	Third Tier Top Unversity	First Tier Top Unversity	Other Top Unversity	Second Tier Top Unversity	Third Tier Tc Unver
country								
Argentina	NaN	44.672857	NaN	NaN	NaN	45.66	NaN	N
Australia	47.9425	44.645750	49.2425	47.285000	51.61	45.97	50.40	47
Austria	NaN	44.864286	NaN	47.066667	NaN	46.29	NaN	47
Belgium	51.8750	45.081000	49.0840	46.746667	52.03	46.21	49.73	47
Brazil	NaN	44.499706	49.5650	NaN	NaN	46.08	49.82	N

Marginal Values

In [6]:

```
df.pivot_table(values='score', index='country', columns='Rank_Level', aggfunc=[n
p.mean, np.max],
               margins=True).head()
```

Out[6]:

	mean					amax		
Rank_Level	First Tier Top Unversity	Other Top Unversity	Second Tier Top Unversity	Third Tier Top Unversity	All	First Tier Top Unversity	Other Top Unversity	Secon Tier T Unver
country								
Argentina	NaN	44.672857	NaN	NaN	44.672857	NaN	45.66	
Australia	47.9425	44.645750	49.2425	47.285000	45.825517	51.61	45.97	5
Austria	NaN	44.864286	NaN	47.066667	45.139583	NaN	46.29	
Belgium	51.8750	45.081000	49.0840	46.746667	47.011000	52.03	46.21	4
Brazil	NaN	44.499706	49.5650	NaN	44.781111	NaN	46.08	4

Pivot Tables as MultiLevel DataFrames

A pivot table is just a multi-level dataframe, and we can access series or cells in the dataframe in a similar way as we do so for a regular dataframe. In the below example, we can see the columns are hierarchical. The top level column indices have two categories: mean and max, and the lower level column indices have four categories, which are the four rank levels.

In [7]:

```
# Let's create a new dataframe from our previous example
new_df=df.pivot_table(values='score', index='country', columns='Rank_Level', agg
func=[np.mean, np.max],
               margins=True)
# Now let's look at the index
print(new_df.index)
# And let's look at the columns
print(new_df.columns)
```

```
Index(['Argentina', 'Australia', 'Austria', 'Belgium', 'Brazil', 'Bu
lgaria',
       'Canada', 'Chile', 'China', 'Colombia', 'Croatia', 'Cyprus',
       'Czech Republic', 'Denmark', 'Egypt', 'Estonia', 'Finland',
'France',
       'Germany', 'Greece', 'Hong Kong', 'Hungary', 'Iceland', 'Indi
a', 'Iran',
       'Ireland', 'Israel', 'Italy', 'Japan', 'Lebanon', 'Lithuani
a',
       'Malaysia', 'Mexico', 'Netherlands', 'New Zealand', 'Norway',
'Poland',
       'Portugal', 'Puerto Rico', 'Romania', 'Russia', 'Saudi Arabi
a',
       'Serbia', 'Singapore', 'Slovak Republic', 'Slovenia', 'South
Africa',
       'South Korea', 'Spain', 'Sweden', 'Switzerland', 'Taiwan', 'T
hailand',
       'Turkey', 'USA', 'Uganda', 'United Arab Emirates', 'United Ki
ngdom',
       'Uruguay', 'All'],
      dtype='object', name='country')
MultiIndex([('mean',  'First Tier Top Unversity'),
            ('mean',       'Other Top Unversity'),
            ('mean', 'Second Tier Top Unversity'),
            ('mean',  'Third Tier Top Unversity'),
            ('mean',                       'All'),
            ('amax',  'First Tier Top Unversity'),
            ('amax',       'Other Top Unversity'),
            ('amax', 'Second Tier Top Unversity'),
            ('amax',  'Third Tier Top Unversity'),
            ('amax',                       'All')],
           names=[None, 'Rank_Level'])
```

How would we query this if we want to get the average scores of First Tier Top Unversity levels in each country? We would just need to **make two dataframe projections**, the first for the mean, then the second for the top tier

In [8]:

```
new_df['mean']['First Tier Top Unversity'].head()
```

Out[8]:

```
country
Argentina        NaN
Australia    47.9425
Austria          NaN
Belgium      51.8750
Brazil           NaN
Name: First Tier Top Unversity, dtype: float64
```

In [9]:

```
# We can see that the output is a series object which we can confirm by printing
the type. Remember that when
# you project a single column of values out of a DataFrame you get a series.
type(new_df['mean']['First Tier Top Unversity'])
```

Out[9]:

```
pandas.core.series.Series
```

In [10]:

```
# What if we want to find the country that has the maximum average score on Firs
t Tier Top University level?
# We can use the idxmax() function.
print(type(new_df['mean']['First Tier Top Unversity'].idxmax()))
new_df['mean']['First Tier Top Unversity'].idxmax()
```

```
<class 'str'>
```

Out[10]:

```
'United Kingdom'
```

Now, the `idxmax()` function isn't special for pivot tables, it's a built in function to the Series object. We don't have time to go over all pandas functions and attributes, and I want to encourage you to explore the API to learn more deeply what is available to you.

Stacking & Unstacking

To achieve a different shape of your pivot table, you can do so with the `stack()` and `unstack()` functions.

- **Stacking** is pivoting the **lowermost** column **index** to become the **innermost row index**.
- **Unstacking** is the opposite of stacking, pivoting the **innermost** row index to become the **lowermost** column index.

In [11]:

```
# If you want to achieve a different shape of your pivot table, you can do so wi
th the stack and unstack
# functions. Stacking is pivoting the lowermost column index to become the inner
most row index. Unstacking is
# the inverse of stacking, pivoting the innermost row index to become the lowerm
ost column index. An example
# will help make this clear

# Let's look at our pivot table first to refresh what it looks like
new_df.head()
```

Out[11]:

	mean					amax		
Rank_Level	**First Tier Top Unversity**	**Other Top Unversity**	**Second Tier Top Unversity**	**Third Tier Top Unversity**	**All**	**First Tier Top Unversity**	**Other Top Unversity**	**Seco Tier T Unve**
country								
Argentina	NaN	44.672857	NaN	NaN	44.672857	NaN	45.66	
Australia	47.9425	44.645750	49.2425	47.285000	45.825517	51.61	45.97	5
Austria	NaN	44.864286	NaN	47.066667	45.139583	NaN	46.29	
Belgium	51.8750	45.081000	49.0840	46.746667	47.011000	52.03	46.21	4
Brazil	NaN	44.499706	49.5650	NaN	44.781111	NaN	46.08	4

In [12]:

```
# Now let's try stacking, this should move the lowermost column, so the tiers of
the university rankings, to
# the inner most row
new_df=new_df.stack()
new_df.head()
```

Out[12]:

		mean	amax
country	**Rank_Level**		
Argentina	**Other Top Unversity**	44.672857	45.66
	All	44.672857	45.66
Australia	**First Tier Top Unversity**	47.942500	51.61
	Other Top Unversity	44.645750	45.97
	Second Tier Top Unversity	49.242500	50.40

In [13]:

```
# In the original pivot table, rank levels are the lowermost column, after stack
ing, rank levels become the
# innermost index, appearing to the right after country

# Now let's try unstacking
new_df.unstack().head()
```

Out[13]:

	mean					amax		
Rank_Level	**First Tier Top Unversity**	**Other Top Unversity**	**Second Tier Top Unversity**	**Third Tier Top Unversity**	**All**	**First Tier Top Unversity**	**Other Top Unversity**	**Secoı Tier T Unveı**
country								
Argentina	NaN	44.672857	NaN	NaN	44.672857	NaN	45.66	
Australia	47.9425	44.645750	49.2425	47.285000	45.825517	51.61	45.97	5
Austria	NaN	44.864286	NaN	47.066667	45.139583	NaN	46.29	
Belgium	51.8750	45.081000	49.0840	46.746667	47.011000	52.03	46.21	4
Brazil	NaN	44.499706	49.5650	NaN	44.781111	NaN	46.08	4

In [14]:

```
# That seems to restore our dataframe to its original shape. What do you think w
ould happen if we unstacked twice in a row?
new_df.unstack().unstack().head()
```

Out[14]:

```
      Rank_Level                 country
mean  First Tier Top Unversity  Argentina         NaN
                                Australia     47.9425
                                Austria           NaN
                                Belgium       51.8750
                                Brazil            NaN
dtype: float64
```

In [15]:

```
# We actually end up unstacking all the way to just a single column, so a series
object is returned. This
# column is just a "value", the meaning of which is denoted by the heirarachical
index of operation, rank, and
# country.
```

Summary

So that's pivot tables. This has been a pretty short description, but they're incredibly useful when dealing with numeric data, especially if you're trying to summarize the data in some form. You'll regularly be creating new pivot tables on slices of data, whether you're exploring the data yourself or preparing data for others to report on. And of course, you can pass any function you want to the aggregate function, including those that you define yourself.

In [ ]:

```
```