

16th November 2015 Python for Data Analysis Part 19: Frequency Tables

Discovering relationships between variables is the fundamental goal of data analysis. Frequency tables are a basic tool you can use to explore data and get an idea of the relationships between variables. A frequency table is just a data table that shows the counts of one or more categorical variables.

To explore frequency tables, we'll revisit the [Titanic training set](https://www.kaggle.com/c/titanic/data) [https://www.kaggle.com/c/titanic/data] from Kaggle that we studied in [lesson 14](http://hamelg.blogspot.com/2015/08/introduction-to-r-part-13-initial-data.html) [http://hamelg.blogspot.com/2015/08/introduction-to-r-part-13-initial-data.html]. We will perform a couple of the same preprocessing steps we did in lesson 14:

```
In [1]: import numpy as np
import pandas as pd
import os
```

```
In [2]: os.chdir('C:\\Users\\Greg\\Desktop\\Kaggle\\titanic') # Set working director
y

titanic_train = pd.read_csv("titanic_train.csv") # Read the data

char_cabin = titanic_train["Cabin"].astype(str) # Convert cabin to str

new_Cabin = np.array([cabin[0] for cabin in char_cabin]) # Take first letter

titanic_train["Cabin"] = pd.Categorical(new_Cabin) # Save the new cabin var
```

One-Way Tables

Create frequency tables (also known as crosstabs) in pandas using the `pd.crosstab()` function. The function takes one or more array-like objects as indexes or columns and then constructs a new DataFrame of variable counts based on the supplied arrays. Let's make a one-way table of the survived variable:

```
In [3]: my_tab = pd.crosstab(index=titanic_train["Survived"], # Make a crosstab
                           columns="count") # Name the count column

my_tab
```

```
Out[3]:
```

col_0	count
Survived	
0	549
1	340

```
In [4]: type(my_tab) # Confirm that the crosstab is a DataFrame
```

```
Out[4]: pandas.core.frame.DataFrame
```

Let's make a couple more crosstabs to explore other variables:

```
In [5]: pd.crosstab(index=titanic_train["Pclass"], # Make a crosstab
                  columns="count") # Name the count column
```

Out[5]:

col_0	count
Pclass	
1	214
2	184
3	491

```
In [6]: pd.crosstab(index=titanic_train["Sex"], # Make a crosstab
                  columns="count") # Name the count column
```

Out[6]:

col_0	count
Sex	
female	312
male	577

```
In [7]: cabin_tab = pd.crosstab(index=titanic_train["Cabin"], # Make a crosstab
                               columns="count") # Name the count column
mn
cabin_tab
```

Out[7]:

col_0	count
Cabin	
A	15
B	45
C	59
D	33
E	32
F	13
G	4
n	688

Even these simple one-way tables give us some useful insight: we immediately get a sense of distribution of records across the categories. For instance, we see that males outnumbered females by a significant margin and that there were more third class passengers than first and second class passengers combined.

If you pass a variable with many unique values to `table()`, such a numeric variable, it will still produce a table of counts for each unique value, but the counts may not be particularly meaningful.

Since the `crosstab` function produces DataFrames, the DataFrame operations we've learned work on crosstabs:

```
In [8]: print (cabin_tab.sum(), "\n") # Sum the counts

print (cabin_tab.shape, "\n") # Check number of rows and cols
```

```
cabin_tab.iloc[1:7] # Slice rows 1-6
```

```
col_0
count      889
dtype: int64

(8, 1)
```

Out[8]:

col_0	count
Cabin	
B	45
C	59
D	33
E	32
F	13
G	4

One of the most useful aspects of frequency tables is that they allow you to extract the proportion of the data that belongs to each category. With a one-way table, you can do this by dividing each table value by the total number of records in the table:

In [9]: `cabin_tab/cabin_tab.sum()`

Out[9]:

col_0	count
Cabin	
A	0.016873
B	0.050619
C	0.066367
D	0.037120
E	0.035996
F	0.014623
G	0.004499
n	0.773903

Two-Way Tables

Two-way frequency tables, also called contingency tables, are tables of counts with two dimensions where each dimension is a different variable. Two-way tables can give you insight into the relationship between two variables. To create a two way table, pass two variables to the `pd.crosstab()` function instead of one:

```
In [10]: # Table of survival vs. sex
survived_sex = pd.crosstab(index=titanic_train["Survived"],
                           columns=titanic_train["Sex"])

survived_sex.index= ["died", "survived"]

survived_sex
```

```
Out[10]:
```

Sex	female	male
died	81	468
survived	231	109

```
In [11]: # Table of survival vs passenger class
survived_class = pd.crosstab(index=titanic_train["Survived"],
                             columns=titanic_train["Pclass"])

survived_class.columns = ["class1", "class2", "class3"]
survived_class.index= ["died", "survived"]

survived_class
```

```
Out[11]:
```

	class1	class2	class3
died	80	97	372
survived	134	87	119

You can get the marginal counts (totals for each row and column) by including the argument `margins=True`:

```
In [12]: # Table of survival vs passenger class
survived_class = pd.crosstab(index=titanic_train["Survived"],
                             columns=titanic_train["Pclass"],
                             margins=True) # Include row and column totals

survived_class.columns = ["class1", "class2", "class3", "rowtotal"]
survived_class.index= ["died", "survived", "coltotal"]

survived_class
```

```
Out[12]:
```

	class1	class2	class3	rowtotal
died	80	97	372	549
survived	134	87	119	340
coltotal	214	184	491	889

To get the total proportion of counts in each cell, divide the table by the grand total:

```
In [13]: survived_class/survived_class.ix["coltotal", "rowtotal"]
```

```
Out[13]:
```

	class1	class2	class3	rowtotal
died	0.089989	0.109111	0.418448	0.617548
survived	0.150731	0.097863	0.133858	0.382452

	class1	class2	class3	rowtotal
coltotal	0.240720	0.206974	0.552306	1.000000

To get the proportion of counts along each column (in this case, the survival rate within each passenger class) divide by the column totals:

In [14]: `survived_class/survived_class.ix["coltotal"]`

Out[14]:

	class1	class2	class3	rowtotal
died	0.373832	0.527174	0.757637	0.617548
survived	0.626168	0.472826	0.242363	0.382452
coltotal	1.000000	1.000000	1.000000	1.000000

To get the proportion of counts along each row divide by the row totals. The division operator functions on a row-by-row basis when used on DataFrames by default. In this case we want to divide each column by the rowtotals column. To get division to work on a column by column basis, use `df.div()` with the axis set to 0 (or "index"):

In [15]: `survived_class.div(survived_class["rowtotal"], axis=0)`

Out[15]:

	class1	class2	class3	rowtotal
died	0.145719	0.176685	0.677596	1
survived	0.394118	0.255882	0.350000	1
coltotal	0.240720	0.206974	0.552306	1

Alternatively, you can transpose the table with `df.T` to swap rows and columns and perform row by row division as normal:

In [16]: `survived_class.T/survived_class["rowtotal"]`

Out[16]:

	died	survived	coltotal
class1	0.145719	0.394118	0.240720
class2	0.176685	0.255882	0.206974
class3	0.677596	0.350000	0.552306
rowtotal	1.000000	1.000000	1.000000

Higher Dimensional Tables

The `crosstab()` function lets you create tables out of more than two categories. Higher dimensional tables can be a little confusing to look at, but they can also yield finer-grained insight into interactions between multiple variables. Let's create a 3-way table inspecting survival, sex and passenger class:

In [17]: `surv_sex_class = pd.crosstab(index=titanic_train["Survived"], columns=[titanic_train["Pclass"],`

```
titanic_train["Sex"]],
margins=True) # Include row and column totals

surv_sex_class
```

Out[17]:

Pclass	1		2		3		All
Sex	female	male	female	male	female	male	
Survived							
0	3	77	6	91	72	300	549
1	89	45	70	17	72	47	340
All	92	122	76	108	144	347	889

Notice that by passing a second variable to the columns argument, the resulting table has columns categorized by both Pclass and Sex. The outermost index (Pclass) returns sections of the table instead of individual columns:

In [18]:

```
surv_sex_class[2] # Get the subtable under Pclass 2
```

Out[18]:

Sex	female	male
Survived		
0	6	91
1	70	17
All	76	108

The secondary column index, Sex, can't be used as a top level index, but it can be used within a given Pclass:

In [19]:

```
surv_sex_class[2]["female"] # Get female column within Pclass 2
```

Out[19]:

```
Survived
0      6
1     70
All    76
Name: female, dtype: int64
```

Due to the convenient hierarchical structure of the table, we still use one division to get the proportion of survival across each column:

In [20]:

```
surv_sex_class/surv_sex_class.ix["All"] # Divide by column totals
```

Out[20]:

Pclass	1		2		3		All
Sex	female	male	female	male	female	male	
Survived							
0	0.032609	0.631148	0.078947	0.842593	0.5	0.864553	0.617548
1	0.967391	0.368852	0.921053	0.157407	0.5	0.135447	0.382452
All	1.000000	1.000000	1.000000	1.000000	1.0	1.000000	1.000000

Here we see something quite interesting: over 90% of women in first class and second class survived, but only 50% of women in third class survived. Men in first class also survived at a greater rate than men in lower classes. Passenger class seems to have a significant impact on survival, so it would likely be useful to include as a feature in a predictive model.


Wrap Up

Frequency tables are a simple yet effective tool for exploring relationships between variables that take on few unique values. Tables do, however, require you to inspect numerical values and proportions closely and it is not always easy to quickly convey insights drawn from tables to others. Creating plots is a way to visually investigate data, which takes advantage of our innate ability to process and detect patterns in images.

Next Time: Python for Data Analysis Part 20: Plotting with Pandas
[<http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-19.html>]

Posted 16th November 2015 by [hamelg](#)

Labels: [contingency tables](#), [crosstabs](#), [frequency tables](#), [pandas](#), [Python for data analysis](#), [tables](#), [tables of counts](#)

 0 Add a comment

Enter your comment...

Comment as: Paulo Ribeiro (▼)

Sign out

Publish

Preview

☐ Notify me