3A Relationships between Variables

So far, we have seen different ways to summarize and visualize *individual* variables in a data set. But we have not really discussed how to summarize and visualize relationships between *multiple* variables. This chapter is all about how to understand relationships between the columns in a `DataFrame`. The methods will be different, depending on whether the variables are categorical or quantitative.

3.1 Relationships between Categorical Variables

In this section, we look at ways to summarize the relationship between two *categorical* variables. To do this, we will again use the Titanic data set.

In [1]:

```
%matplotlib inline
import pandas as pd

titanic_df = pd.read_csv("titanic.csv")
```

Suppose we want to understand the relationship between where a passenger embarked and what class they were in. We can completely summarize this relationship by counting the number of passengers in each class that embarked at each location. We can create a pivot table that summarizes this information.

In [2]:

```
embarked_pclass_counts = titanic_df.pivot_table(
    index="embarked", columns="pclass",
    values="name",  # We can pretty much count any column, as long as there are no NaNs.
    aggfunc="count" # The count function will count the number of non-null values.
)
embarked_pclass_counts
```

Out[2]:

pclass	1	2	3
embarked			
C	141	28	101
Q	3	7	113
S	177	242	495

A pivot table that stores counts is also called a **contigency table** or a **cross-tabulation**. This type of pivot table is common enough that there is a specific function in `pandas` to calculate one, allowing you to bypass `.pivot_table`:

In [3]:

```
counts = pd.crosstab(titanic_df.embarked, titanic_df.pclass)
counts
```

Out[3]:

pclass	1	2	3
embarked			
C	141	28	101
Q	3	7	113
S	177	242	495

Joint Distributions

It is common to normalize the counts in a table so that they add up to 1. These proportions represent the **joint distribution** of the two variables.

To calculate the joint distribution, we need to divide the table of counts above by the total count. To find the total count, we call `.sum()` twice; the first call gives us the sum of each column, and the second call adds those numbers together.

In [4]:

```
print(counts.sum().sum())
joint = counts / counts.sum().sum()
joint
```

```
1307
```

Out[4]:

pclass	1	2	3
embarked			
C	0.107881	0.021423	0.077276
Q	0.002295	0.005356	0.086458
S	0.135425	0.185157	0.378730

Note that this is yet another example of broadcasting. When we divided the `DataFrame` `counts` by the number 1307, the division was applied elementwise, producing another `DataFrame` .

Each cell in this `DataFrame` tells us a joint proportion. For example, the cell in the bottom right tells us the proportion of all passengers that embarked at Southampton and were in 3rd class. We notate this joint proportion as follows:
$$ P(\text{embarked at Southampton and in 3rd class}) = .379. $$

The joint distribution above could also have been obtained by specifying `normalize=True` when the contingency table was first created:

In [5]:

```
pd.crosstab(titanic_df.embarked, titanic_df.pclass,
            normalize=True)
```

Out[5]:

pclass embarked	1	2	3
C	0.107881	0.021423	0.077276
Q	0.002295	0.005356	0.086458
S	0.135425	0.185157	0.378730

The above joint distribution is not, strictly speaking, a contingency table. A contingency table is a table of all counts, while the above table is a table of proportions.

Marginal Distributions

The **marginal distribution** of a variable is simply the distribution of that variable, ignoring the other variables. To calculate the marginal distribution from a joint distribution of two variables, we sum the rows or the columns of the joint distribution.

For example, to calculate the marginal distribution of `embarked` , we have to sum the joint distribution over the columns---in other words, *roll-up* or *marginalize over* the `pclass` variable:

In [6]:

```
joint.sum(axis=1)
```

Out[6]:

```
embarked
C    0.206580
Q    0.094109
S    0.699311
dtype: float64
```

We can check this answer by calculating the distribution of `embarked` directly from the original data, ignoring `pclass` entirely.

In [7]:

```
embarked_counts = titanic_df.groupby("embarked")["name"].count()
embarked_counts / embarked_counts.sum()
```

Out[7]:

```
embarked
C    0.206580
Q    0.094109
S    0.699311
Name: name, dtype: float64
```

The numbers match!

Likewise, we calculate the marginal distribution of `pclass` by summing the joint distribution over the rows---in other words, by *rolling-up* or *marginalizing over* the `embarked` variable:

In [8]:

```
joint.sum(axis=0)
```

Out[8]:

```
pclass
1    0.245601
2    0.211936
3    0.542464
dtype: float64
```

So given the joint distribution of two categorical variables, there are two marginal distributions: one for each of the variables. These marginal distributions are obtained by summing the joint distribution table over the rows and over the columns.

The *marginal distribution* is so-named because these row and column totals would typically be included alongside the joint distribution, in the *margins* of the table. A contingency table with the marginal distributions included can be obtained by specifying `margins=True` in `pd.crosstab` :

In [9]:

```
pd.crosstab(titanic_df.embarked, titanic_df.pclass,
            normalize=True, margins=True)
```

Out[9]:

pclass embarked	1	2	3	All
C	0.107881	0.021423	0.077276	0.206580
Q	0.002295	0.005356	0.086458	0.094109
S	0.135425	0.185157	0.378730	0.699311
All	0.245601	0.211936	0.542464	1.000000

Conditional Distributions

The **conditional distribution** tells us about the distribution of one variable, *conditional on* the value of another. For example, we might want to know the proportion of 3rd class passengers that embarked at each location. In other words, what is the distribution of where a passenger embarked, *conditional on* being in 3rd class?

If we go back to the contingency table:

In [10]:

```
embarked_pclass_counts
```

Out[10]:

pclass embarked	1	2	3
C	141	28	101
Q	3	7	113
S	177	242	495

there were $101 + 113 + 495 = 709$ passengers in 3rd class, of whom

- $101 / 709 = .142$ were in 1st class,
- $113 / 709 = .159$ were in 2nd class, and
- $495 / 709 = .698$ were in 3rd class.

We can calculate these proportions in code by dividing the `pclass=3` column by its sum:

In [11]:

```
embarked_pclass_counts[3] / embarked_pclass_counts[3].sum()
```

Out[11]:

```
embarked
C    0.142454
Q    0.159379
S    0.698166
Name: 3, dtype: float64
```

Notice that these three proportions add up to 1, making this a proper distribution.

This conditional distribution helps us answer questions such as, "What proportion of 3rd class passengers embarked at Southampton?" We notate this conditional proportion as follows:
$$ P\big(\textrm{embarked at Southampton}\ \big|\ \textrm{in 3rd class}\big) = 0.698. $$

The pipe $\big|$ is read "given". So we are interested in the proportion of passengers who embarked at Southampton, *given* that they were in 3rd class.

We could have also calculated this conditional distribution from the joint distribution (i.e., proportions instead of counts):

In [12]:

```
joint[3] / joint[3].sum()
```

Out[12]:

```
embarked
C    0.142454
Q    0.159379
S    0.698166
Name: 3, dtype: float64
```

We have just calculated *one* of the conditional distributions of `embarked` : the distribution conditional on being in 3rd class. There are two more conditional distributions of `embarked` :

- the distribution conditional on being in 1st class
- the distribution conditional on being in 2nd class

It is common to report *all* of the conditional distributions of one variable given another variable.

Of course, it is straightforward to calculate these conditional distributions manually:

In [13]:

```
embarked_pclass_counts[1] / embarked_pclass_counts[1].sum()
```

Out[13]:

```
embarked
C    0.439252
Q    0.009346
S    0.551402
Name: 1, dtype: float64
```

In [14]:

```
embarked_pclass_counts[2] / embarked_pclass_counts[2].sum()
```

Out[14]:

```
embarked
C    0.101083
Q    0.025271
S    0.873646
Name: 2, dtype: float64
```

But there is a nifty trick for calculating all three conditional distributions at once. By summing the counts over `embarked` , we obtain the total number of people in each `pclass` :

In [15]:

```
pclass_counts = embarked_pclass_counts.sum(axis=0)
pclass_counts
```

Out[15]:

```
pclass
1    321
2    277
3    709
dtype: int64
```

This is exactly what we need to divide each column of `embarked_pclass_counts` by:

In [16]:

```
embarked_given_pclass = embarked_pclass_counts.divide(pclass_counts, axis=1)
embarked_given_pclass
```

Out[16]:

pclass	1	2	3
embarked			
C	0.439252	0.101083	0.142454
Q	0.009346	0.025271	0.159379
S	0.551402	0.873646	0.698166

(This is yet another example of *broadcasting*, since we are dividing a `DataFrame` by a `Series` .)

Compare each column with the numbers we obtained earlier. Notice also that each column sums to 1, a reminder that each column represents a separate distribution.

When comparing numbers across distributions, it is important to be careful. For example, the 87.4% and the 69.8% in the "Southampton" row represent percentages of different populations. Just because 87.4% is higher than 69.8% does not mean that more 2nd class passengers boarded at Southampton than 3rd class passengers. In fact, if we go back to the original contingency table, we see that more 3rd class passengers actually boarded at Southampton than 2nd class passengers!

There is also another set of conditional distributions for these two variables: the distribution of class, conditional on where they embarked. To calculate these conditional distributions, we instead divide `embarked_pclass_counts` by the sum of each row:

In [17]:

```
embarked_counts = embarked_pclass_counts.sum(axis=1)
pclass_given_embarked = embarked_pclass_counts.divide(embarked_counts, axis=0)
pclass_given_embarked
```

Out[17]:

pclass	1	2	3
embarked			
C	0.522222	0.103704	0.374074
Q	0.024390	0.056911	0.918699
S	0.193654	0.264770	0.541575

These conditional distributions answer questions like, "What proportion of Southampton passengers were in 3rd class?"

Notice that these proportions are *not* the same as the proportions for the other set of conditional distributions. That is because the two questions below are fundamentally different:

Question 1. What proportion of 3rd class passengers embarked at Southampton?
$$P\big(\textrm{embarked at Southampton}\ \big|\ \textrm{in 3rd class}\big) = \frac{\text{# passengers who embarked at Southampton and in 3rd class}}{\text{# passengers who in 3rd class}}$$

Question 2. What proportion of Southampton passengers were in 3rd class?
$$P\big(\textrm{in 3rd class}\ \big|\ \textrm{embarked at Southampton}\big) = \frac{\text{# passengers who embarked at Southampton and in 3rd class}}{\text{# passengers who embarked at Southampton}} \\ $$

In the first case, the reference population is all passengers who embarked at Southampton. In the second case, the reference population is all passengers who were in 3rd class. The numerators may be the same, but the denominators are different. In general, the conditional distributions of X given Y are *not* the same as the conditional distributions of Y given X.

If we rephrase the question slightly, we get yet another answer:

Question 3. What proportion of passengers embarked at Southampton *and* were in 3rd class?
$$P(\text{embarked at Southampton and in 3rd class}) = \frac{\text{# passengers who embarked at Southampton and in 3rd class}}{\text{# passengers (total)}}$$

The reference population here is all passengers. This is the proportion that one would get from the joint distribution.

It is important to pay attention to the wording of the question, to determine whether a joint distribution or a conditional distribution is called for---and, if the latter, which of the two conditional distributions is appropriate.

Visualization

How do we visualize the joint and conditional distributions of two categorical variables?

To visualize a joint distribution, we need to be able to represent three dimensions: two dimensions for the two categorical variables and a third dimension for the proportions. Although one option is a 3D graph, humans are not good at judging the sizes of 3D objects printed on a page. For this reason, **heat maps**, which use a color scale to represent the third dimension, are usually preferred.

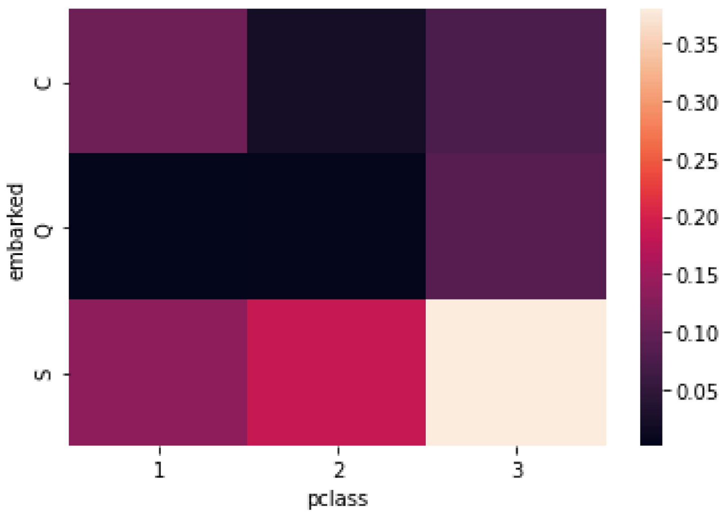
Unfortunately, heat maps are still not easy to create in `pandas`. We use the `seaborn` library to make a heat map:

In [18]:

```
import seaborn as sns

sns.heatmap(joint)
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f645db136d8>
```



A heat map encourages comparison across cells. So we see that 3rd class passengers who embarked at Southampton were by far the most common.

Although a heat map can also be used to visualize conditional distributions, it is not ideal because it does not tell us which variable we are conditioning on, and it is difficult to judge visually which dimension sums to 1. A stacked bar graph is better because it visually shows values summing to 1.

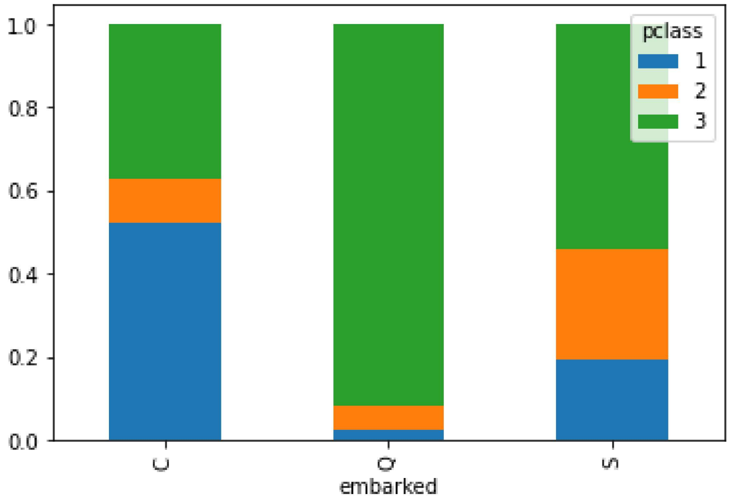
To make a stacked bar graph, we simply specify `stacked=True` in `.plot.bar()` , to get the bars to show up on top of one another, instead of side-by-side:

In [19]:

```
pclass_given_embarked.plot.bar(stacked=True)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f645b37b4a8>
```



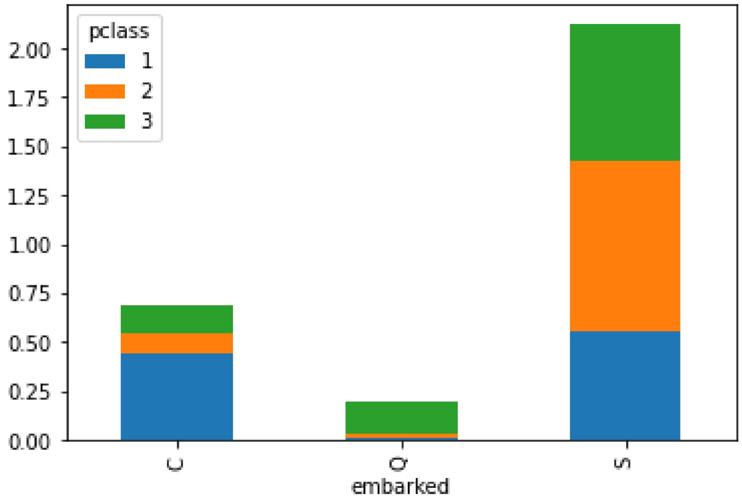
However, the same code does not work on the other set of conditional distributions:

In [20]:

```
embarked_given_pclass.plot.bar(stacked=True)
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f645b305908>
```



What went wrong? Recall that `.plot.bar()` automatically plots the (row) index of the `DataFrame` on the x-axis. To plot the distribution of `embarked` conditional on `pclass`, we need `pclass` to be on the x-axis, but

In [21]:

```
embarked_given_pclass
```

Out[21]:

pclass	1	2	3
embarked			
C	0.439252	0.101083	0.142454
Q	0.009346	0.025271	0.159379
S	0.551402	0.873646	0.698166

has `embarked` as the index. To make `pclass` the index, we can **transpose** this `DataFrame` so that the rows become columns and the columns become rows. The syntax for transposing a `DataFrame` is `.T`, which is inspired by the notation for transposing a matrix in linear algebra.

In [22]:

```
embarked_given_pclass.T
```

Out[22]:

embarked	C	Q	S
pclass			
1	0.439252	0.009346	0.551402
2	0.101083	0.025271	0.873646
3	0.142454	0.159379	0.698166

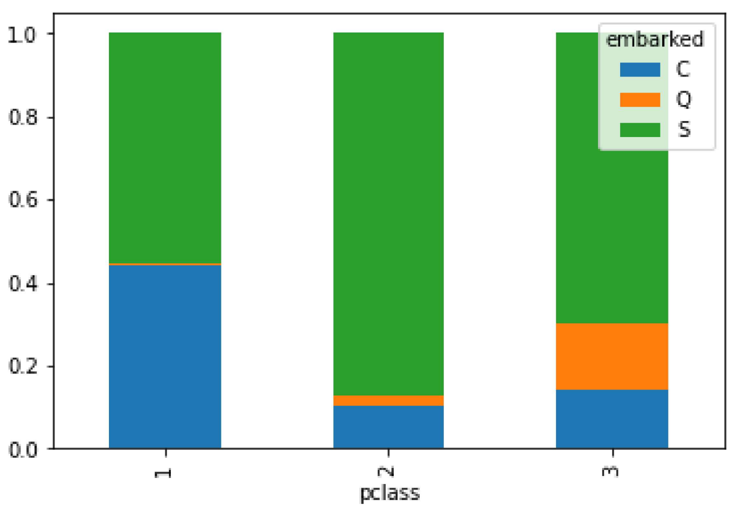
Now, we can make a stacked bar graph from this *transposed* `DataFrame`:

In [23]:

```
(embarked_given_pclass.T).plot.bar(stacked=True)
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f645b29e4e0>
```



Exercises

Exercises 1-4 deal with the Tips data set (`tips.csv`).

Exercise 1. Make a visualization that displays the relationship between the day of the week and party size.

In [24]:

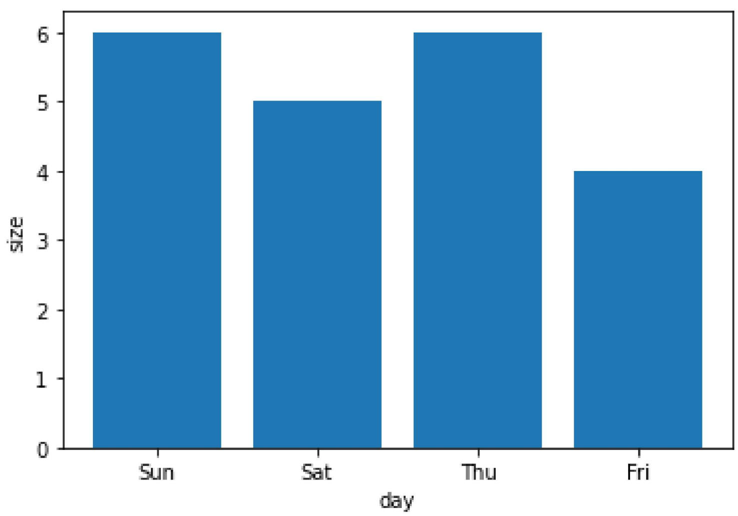
```
# ENTER YOUR CODE HERE
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

tips_df = pd.read_csv("tips.csv")
tips_df.head()
```

Out[24]:

	obs	totbill	tip	sex	smoker	day	time	size
0	1	16.99	1.01	F	No	Sun	Night	2
1	2	10.34	1.66	M	No	Sun	Night	3
2	3	21.01	3.50	M	No	Sun	Night	3
3	4	23.68	3.31	M	No	Sun	Night	2
4	5	24.59	3.61	F	No	Sun	Night	4

In [25]:

```
plt.bar(tips_df['day'], tips_df['size'])
plt.xlabel('day')
plt.ylabel('size')
plt.show()
```



Exercise 2. Calculate the marginal distribution of day of week in two different ways.

In [26]:

```
# ENTER YOUR CODE HERE

# Method 1: Using Functions of pandas
# Create a contingency table for tips
tip_counts1 = pd.crosstab(tips_df.day, tips_df.size)
# Create a pivot table for a marginal distribution of tips
pd.crosstab(tips_df.day, tips_df.size, normalize=True, margins=True)
```

Out[26]:

col_0	1952	All
day		
Fri	0.077869	0.077869
Sat	0.356557	0.356557
Sun	0.311475	0.311475
Thu	0.254098	0.254098
All	1.000000	1.000000

In [27]:

```
# Method 2: Using Calculations
# Create a pivot table that summarizes the information of tips
day_size_counts = tips_df.pivot_table(
 index="day", columns="size",
 values="obs", # We can pretty much count any column, as long as there are no NaNs.
 aggfunc="count" # The count function will count the number of non-null values.
)
# Calculate the marginal distribution of tips
tip_counts2 = tips_df.groupby("day")["size"].count()
tips2 = tip_counts2 / tip_counts2.sum()
tips2.head()
```

Out[27]:

```
day
Fri    0.077869
Sat    0.356557
Sun    0.311475
Thu    0.254098
Name: size, dtype: float64
```

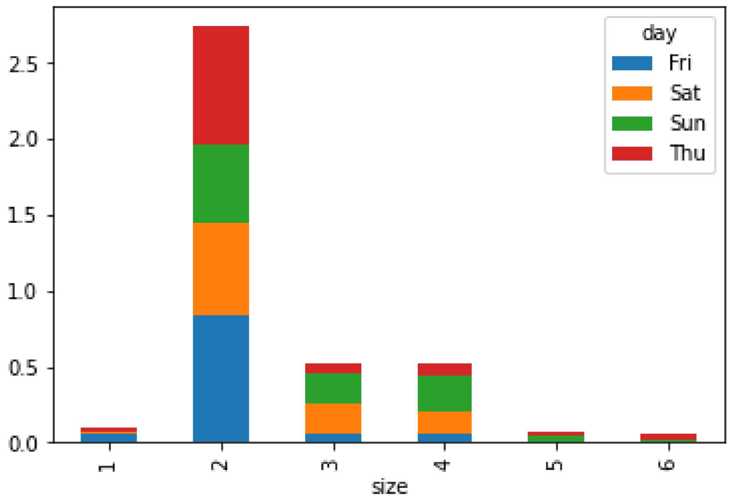
Exercise 3. Make a visualization that displays the conditional distribution of party size, given the day of the week.

In [28]:

```
# ENTER YOUR CODE HERE
day_counts = day_size_counts.sum(axis=1)
day_given_size = day_size_counts.divide(day_counts, axis=0)
(day_given_size.T).plot.bar(stacked=True)
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f645af11d30>
```



Exercise 4. What proportion of Saturday parties had 2 people? Is this the same as the proportion of 2-person parties that dined on Saturday?

In [29]:

```
# ENTER YOUR CODE HERE
print("Proportion of Saturday parties having 2 people: ", day_given_size[2].Sat)
day_given_size
```

```
Proportion of Saturday parties having 2 people:  0.6091954022988506
```

Out[29]:

size	1	2	3	4	5	6
day						
Fri	0.052632	0.842105	0.052632	0.052632	NaN	NaN
Sat	0.022989	0.609195	0.206897	0.149425	0.011494	NaN
Sun	NaN	0.513158	0.197368	0.236842	0.039474	0.013158
Thu	0.016129	0.774194	0.064516	0.080645	0.016129	0.048387

In [30]:

```
## Explaination:
# The proportion of Saturday parties having 2 people is around 61%.
# It would be the same proportion of a 2-person parties that dined on Saturday.
```