Playing with Tables

There are three types of table functions that are essential to organizing our data: group, pivot, and join. **Group** allows us to see information based on similar data. **Pivot** allows us to see a version of grouping twice, but in a very nice and readable format. **Join** allows us to merge to tables together into one bigger table based on similar data between the two.

Group

Let's say we are still working at that coffee shop and we have some data on how the drinks are made. Here it is (in a table named **drinks**):

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Latte	Large	0	True
Vanilla Latte	Small	3	True
Vanilla Latte	Large	5	True
Cold Brew	Small	0	False
Cold Brew	Large	0	False
Vanilla Cold Brew	Small	4	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Small	0	False
Iced Tea	Large	0	False
Espresso Macchiato	Small	0	True

Now your boss comes up to you and wants to know how many drinks on the menu have foam and how many don't. Luckily for you, you can use the group table function! Here's the syntax:

```
<Table Name>.group(<col name>)
```

This will give you a table with all the unique names in the column, along with the number of rows that have that unique name. Here's the output of the code.

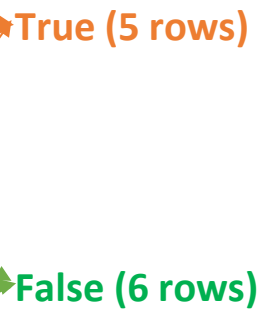
```
In  [1]: drinks.group('Foam')
```

Out [1]:

Foam	Count
False	6
True	5

Here's the visualization of what went on under the hood:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Latte	Large	0	True
Vanilla Latte	Small	3	True
Vanilla Latte	Large	5	True
Cold Brew	Small	0	False
Cold Brew	Large	0	False
Vanilla Cold Brew	Small	4	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Small	0	False
Iced Tea	Large	0	False
Espresso Macchiato	Small	0	True



So, Group will cluster all the rows into the unique values of the column you want to group by. Foam only has 2 unique values, True or False. Thus, every row will either be True or False. Then by default, python just takes the number of rows that correspond to each unique value and calls that the "Count". Then it outputs the table you saw earlier:

Foam	Count
False	6
True	5

Where the first column is the column we grouped by, "Foam". Then the rows correspond to the unique values (the clustered groups), which are True and False. Python also alphabetizes them, so that's why False comes before True. The second column is the "Count" mentioned earlier which displays the corresponding count.

Group also has an optional second argument, which takes in an **aggregate function**. An aggregate function is a function that python will use to aggregate the columns into one number instead of using the default count. It's easier to see this through an example:

```
In [2]: drinks.group('Size', np.mean)
```

Out [2]:

Size	Drink Mean	Pumps of Vanilla Mean	Foam Mean
Large		2.2	0.4
Small		1.16667	0.5

This may seem a little weird right now. Let's see what goes on in python's mind.

So what python does is it first it identifies the unique values of the column we specified. In this case, the column “Size” only has the values “Small” and “Large”. Here it is visualized:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Latte	Large	0	True
Vanilla Latte	Small	3	True
Vanilla Latte	Large	5	True
Cold Brew	Small	0	False
Cold Brew	Large	0	False
Vanilla Cold Brew	Small	4	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Small	0	False
Iced Tea	Large	0	False
Espresso Macchiato	Small	0	True

Then python groups them up:

Small:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Vanilla Latte	Small	3	True
Cold Brew	Small	0	False
Vanilla Cold Brew	Small	4	False
Iced Tea	Small	0	False
Espresso Macchiato	Small	0	True

Large:

Drink	Size	Pumps of Vanilla	Foam
Latte	Large	0	True
Vanilla Latte	Large	5	True
Cold Brew	Large	0	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Large	0	False

So now python has grouped the rows. By default, it would take the count, so the number of rows in each. In this case, “Small” would have a count of 6 and “Large” would have a count of 5. However, we put another function for python to use instead, `np.mean`. Python will use this function on each column of each group, and then output that in the new table it creates. Let’s see how it works for the “Small” group:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Vanilla Latte	Small	3	True
Cold Brew	Small	0	False
Vanilla Cold Brew	Small	4	False
Iced Tea	Small	0	False
Espresso Macchiato	Small	0	True

So python visits each column left to right and computes the function we gave it on each of those columns (except for the column we're grouping on). So first it takes the mean of the Drink column. In code it would look like this:

```
np.mean(drinks.column('Drink'))
```

However, we can't compute the mean of strings. Instead of erroring, python just doesn't do anything with it. Now python is done with the "Drink" column and moves onto "Size". That's what we're grouping by, so we skip it. So we go to the "Pumps of Vanila" column:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Vanilla Latte	Small	3	True
Cold Brew	Small	0	False
Vanilla Cold Brew	Small	4	False
Iced Tea	Small	0	False
Espresso Macchiato	Small	0	True

Equivalent python expression: `np.mean(drinks.column('Pumps of Vanilla'))`

Which ends up becoming **1.16667**.

Next we go onto the "Foam" column. You might think that we can't take the mean of Booleans. However, when taking the mean, True ends up being equivalent to 1 and False is 0. So we can compute the mean and we don't skip it. Here's what it looks like:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	1
Vanilla Latte	Small	3	1
Cold Brew	Small	0	0
Vanilla Cold Brew	Small	4	0
Iced Tea	Small	0	1
Espresso Macchiato	Small	0	1

Equivalent python expression: `np.mean(drinks.column('Foam'))`

Which ends up being **0.5**.

Here are our results:

- Mean of Drinks: Nothing
- Mean of Pumps of Vanilla: 1.1667
- Mean of Foam: 0.5

Now we have everything to create a table based on our group, “Small”. The first column is always the column we grouped by. And the value for our group is the value we grouped by, in this case “Small”. Then the following column titles are edited so instead of Drinks or Mean of Drinks, it’s Drink Means. So, it’s the column name concatenated with the name of the function. This results in this table:

Size	Drink Mean	Pumps of Vanilla Mean	Foam Mean
Small		1.16667	0.5

Now we do the same thing for the other groups. In our contrived case, the only other group is “Large” which creates this table:

Size	Drink Mean	Pumps of Vanilla Mean	Foam Mean
Large		2.2	0.4

Now we have all the groups and their corresponding tables, we just put them together. The order they go in is alphabetical based on the grouped column. So “Large” would come before “Small”. Thus we get this resulting table:

Size	Drink Mean	Pumps of Vanilla Mean	Foam Mean
Large		2.2	0.4
Small		1.16667	0.5

Grouping by Multiple Columns

We can abstract this notion of grouping by multiple columns. How I think about it is by grouping twice. We group by one column, and then we take those groups and group them by another column. Let’s see how it looks like in code:

```
In  [2]: drinks.group(['Size', 'Foam'], max)
```

Out [2]:

Size	Foam	Drink Max	Pumps of Vanilla Max
Large	False	Vanilla Cold Brew	6
Large	True	Vanilla Latte	5
Small	False	Vanilla Cold Brew	4
Small	True	Vanilla Latte	3

If you notice, we put the column names inside these brackets to signify we are grouping with multiple columns. We also used a different function that’s not the default or np.mean, max. Let’s see how python works through this visually:

First, we group by the first column we're given. We've done something similar to this before, but here are the two groups again:

Small:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Vanilla Latte	Small	3	True
Cold Brew	Small	0	False
Vanilla Cold Brew	Small	4	False
Iced Tea	Small	0	False
Espresso Macchiato	Small	0	True

Large:

Drink	Size	Pumps of Vanilla	Foam
Latte	Large	0	True
Vanilla Latte	Large	5	True
Cold Brew	Large	0	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Large	0	False

Ok great. Now python groups by the next column it's given, which in our case is "Foam". Let's see how this works for the Small group:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Vanilla Latte	Small	3	True
Cold Brew	Small	0	False
Vanilla Cold Brew	Small	4	False
Iced Tea	Small	0	False
Espresso Macchiato	Small	0	True

Above are the shaded groups, so we just separate them into groups again. Here they are separated:

Small and True

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Vanilla Latte	Small	3	True
Espresso Macchiato	Small	0	True

Small and False

Drink	Size	Pumps of Vanilla	Foam
Cold Brew	Small	0	False
Vanilla Cold Brew	Small	4	False
Iced Tea	Small	0	False

There is no more groups we have to group by, so now we compute the function we want on all of them. The default is the count, but we put in max, so we compute the max of every column in each group (excluding the groups we're grouping by):

Small and True

- Max of Drink: "Vanilla Latte"
- Max of Pumps of Vanilla: 3

Small and False

- Max of Drink: "Vanilla Cold Brew"
- Max of Pumps of Vanilla: 4

So now we can create tables based on the groups we just observed:

Small and True

Size	Foam	Drink Max	Pumps of Vanilla Max
Small	True	Vanilla Latte	3

Small and False

Size	Foam	Drink Max	Pumps of Vanilla Max
Small	False	Vanilla Cold Brew	4

Then we combine these groups together. Since they have the same "Size" (the first column we grouped by), we have to alphabetize by "Foam", the second group we group by and the most recent. Thus, the resulting table would be:

Size	Foam	Drink Max	Pumps of Vanilla Max
Small	False	Vanilla Cold Brew	4
Small	True	Vanilla Latte	3

Now we are done with the "Small" group. So we go onto the next group, which is large. If we repeat the process, we get this table:

Size	Foam	Drink Max	Pumps of Vanilla Max
Large	False	Vanilla Cold Brew	6
Large	True	Vanilla Latte	5

Then python would go onto the next group and repeat the process until there are no more groups. But the only two groups, Small and Large are done (and have already been grouped by "Foam", the second column to group by). So we just combine all the tables:

Size	Foam	Drink Max	Pumps of Vanilla Max
Large	False	Vanilla Cold Brew	6
Large	True	Vanilla Latte	5
Small	False	Vanilla Cold Brew	4
Small	True	Vanilla Latte	3

So we can see that we have every unique pair of Size and Foam in the table, along with the maximum "Drink" and maximum "Pumps of Vanilla"

Note: you're not expected to know how python determines what strings are greater than other strings. If you're interested, it does so by comparing ascii values. If you're still interested, try searching on Stack Overflow!

Pivot

Pivot is a way of grouping by two columns, like we just did before. However, it has a different format. Let's see it in code first before we dive into what happens:

```
In  [3]: drinks.pivot('Pumps of Vanilla', 'Drink')
```

Out [3]:

Drink	0	3	4	5	6
Cold Brew	2	0	0	0	0
Espresso Machiatto	1	0	0	0	0
Iced Tea	2	0	0	0	0
Latte	2	0	0	0	0
Vanilla Cold Brew	0	0	1	0	1
Vanilla Latte	0	1	0	1	0

Alright so this may be a huge jump. How is this like grouping by two columns? Well let's break it down. First let's look at the syntax. The first argument is the column `'Pumps of Vanilla'`. This takes all the **unique values** in the column "Pumps of Vanilla" and makes it their own column. So, the unique values are either 0, 3, 4, 5, or 6 pumps of vanilla, so they become the new column labels. The second argument is `'Drink'`. The unique values of the "Drink" column become the new row labels.

But what in the heck are the numbers?? Well just like group, the default aggregate function is count. So the numbers represent the number of drinks that correspond to the "Drink" and the "Pumps of Vanilla". This may be a little abstract, so let's visualize this on the next page.

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Latte	Large	0	True
Vanilla Latte	Small	3	True
Vanilla Latte	Large	5	True
Cold Brew	Small	0	False
Cold Brew	Large	0	False
Vanilla Cold Brew	Small	4	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Small	0	False
Iced Tea	Large	0	False
Espresso Macchiato	Small	0	True

So above, we have all the possible combinations of "Drink" and "Pumps of Vanilla". Just like grouping by two columns, which gives all the possible combinations of the two columns, pivot does the same. We can see that the different highlighted colors represent all the different combinations of "Drink" and "Pumps of Vanilla" in the table. Since we're using the default count aggregate function, then we just count the number of rows that correspond to each color.

Drink	Size	Pumps of Vanilla	Foam	
Latte	Small	0	True	**2**: Latte, 0
Latte	Large	0	True	
Vanilla Latte	Small	3	True	**1**: Vanilla Latte, 3
Vanilla Latte	Large	5	True	**1**: Vanilla Latte, 5
Cold Brew	Small	0	False	**2**: Cold Brew, 0
Cold Brew	Large	0	False	
Vanilla Cold Brew	Small	4	False	**1**: Vanilla Cold Brew, 4
Vanilla Cold Brew	Large	6	False	**1**: Vanilla Cold Brew, 6
Iced Tea	Small	0	False	**2**: Iced Tea, 0
Iced Tea	Large	0	False	
Espresso Macchiato	Small	0	True	**1**: Espresso Macchiato, 0

So now we have the total rows for each combination. Notice how the I set the combination as a coordinate, like **Latte, 0**. That's because that number goes into that coordinate in the pivot table! So after this step, it's just about filling in the pivot table.

Drink	0	3	4	5	6
Cold Brew	2	0	0	0	0
Espresso Machiatto	1	0	0	0	0
Iced Tea	2	0	0	0	0
Latte	2	0	0	0	0
Vanilla Cold Brew	0	0	1	0	1
Vanilla Latte	0	1	0	1	0

Latte, 0 has 2 rows

Vanilla Latte, 5 has 1 row

Pivot tables can also use aggregate functions like group. The syntax is as follows:

```
<Tbl Name>.pivot(<col for columns>,<col for rows>,<values>,<agg func>)
```

So, this kind of pivot table is a little different. It still turns the first column's unique values into its own separate columns. It also still turns the second column's unique values into its own separate rows. However, there are two extra arguments. Instead of using the default count aggregate function that counts the number of rows, we use a different function. We apply it to each unique combination. However, we need to specify which column to use, which is the `<values>` parameter. This values column doesn't have to be one of, and most likely shouldn't be either of the two columns already specified: `<col for columns>` and `<col for rows>`

Joins

So far, we've been able to manipulate one table. Joins allows us to combine two tables into one based on similar features. Let's take a look at two tables I have created here:

drinks:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Latte	Large	0	True
Vanilla Latte	Small	3	True
Vanilla Latte	Large	5	True
Cold Brew	Small	0	False
Cold Brew	Large	0	False
Vanilla Cold Brew	Small	4	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Small	0	False
Iced Tea	Large	0	False
Espresso Macchiato	Small	0	True

sugar:

Pumps	Grams of Sugar
0	0
1	5
2	10
3	15
4	20
5	25
6	30

It would be nice if we could get how much sugar is in each drink all in one table so it's reasy to read. Well we can do that with the join table method. Here's the syntax:

```
<tblA>.join(<colA>,<tblB>,<colB>)
```

I think the reference sheet explains this the best: "Generate a table with the columns of tblA and tblB, containing rows for all values of a column that appear in both tables."

Let's see how it works on the next page:

```
In [4]: drinks.join('Pumps of Vanilla', sugar, 'Pumps')
```

Out [4]:

Pumps of Vanilla	Drink	Size	Foam	Grams of Sugar
0	Latte	Small	True	0
0	Latte	Large	True	0
0	Cold Brew	Small	False	0
0	Cold Brew	Large	False	0
0	Iced Tea	Small	False	0
0	Iced Tea	Large	False	0
0	Espresso Macchiato	Small	True	0
3	Vanilla Latte	Small	True	15
4	Vanilla Cold Brew	Small	False	20
5	Vanilla Latte	Large	True	25
6	Vanilla Cold Brew	Large	False	30

Let's find out how this came to be. The way I think about it is I highlight the two columns we are comparing in the two tables:

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Latte	Large	0	True
Vanilla Latte	Small	3	True
Vanilla Latte	Large	5	True
Cold Brew	Small	0	False
Cold Brew	Large	0	False
Vanilla Cold Brew	Small	4	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Small	0	False
Iced Tea	Large	0	False
Espresso Macchiato	Small	0	True

Pumps	Grams of Sugar
0	0
1	5
2	10
3	15
4	20
5	25
6	30

Then, whenever we go down the highlighted column of tblA (in this case the drinks table) and we see if there exists the same value in tblB's highlighted column. That's illustrated on the next page.

Drink	Size	Pumps of Vanilla	Foam
Latte	Small	0	True
Latte	Large	0	True
Vanilla Latte	Small	3	True
Vanilla Latte	Large	5	True
Cold Brew	Small	0	False
Cold Brew	Large	0	False
Vanilla Cold Brew	Small	4	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Small	0	False
Iced Tea	Large	0	False
Espresso Macchiato	Small	0	True

Pumps	Grams of Sugar
0	0
1	5
2	10
3	15
4	20
5	25
6	30

Now that we've matched everything, we take the row that the arrow is pointing to in tblB and smush it onto tblA like so:

Drink	Size	Pumps of Vanilla	Foam	Pumps	Grams of Sugar
Latte	Small	0	True	0	0
Latte	Large	0	True	0	0
Vanilla Latte	Small	3	True	3	15
Vanilla Latte	Large	5	True	5	25
Cold Brew	Small	0	False	0	0
Cold Brew	Large	0	False	0	0
Vanilla Cold Brew	Small	4	False	4	20
Vanilla Cold Brew	Large	6	False	6	30
Iced Tea	Small	0	False	0	0
Iced Tea	Large	0	False	0	0
Espresso Macchiato	Small	0	True	0	0

To reiterate, the first row in tblA (drinks) had an arrow pointing to the first row of tblB (sugar). We took the whole row it was pointing to in tblB it was pointing to and added it to the row in tblA. Then we did this for all of the rows in tblA.

Since we smushed all the values from tblB to tblA, we have essentially the same columns (since we were comparing the two tables based on the same values of certain columns). So the join method is going to get rid of the highlighted column from tblB, in this case, "Pumps".

Drink	Size	Pumps of Vanilla	Foam	Grams of Sugar
Latte	Small	0	True	0
Latte	Large	0	True	0
Vanilla Latte	Small	3	True	15
Vanilla Latte	Large	5	True	25
Cold Brew	Small	0	False	0
Cold Brew	Large	0	False	0
Vanilla Cold Brew	Small	4	False	20
Vanilla Cold Brew	Large	6	False	30
Iced Tea	Small	0	False	0
Iced Tea	Large	0	False	0
Espresso Macchiato	Small	0	True	0

Then, python will make the compared column (the highlighted one) the first column and sort it (depending on what type the array is it could be numerically or alphabetically). That will give us the resulting table:

Pumps of Vanilla	Drink	Size	Foam	Grams of Sugar
0	Latte	Small	True	0
0	Latte	Large	True	0
0	Cold Brew	Small	False	0
0	Cold Brew	Large	False	0
0	Iced Tea	Small	False	0
0	Iced Tea	Large	False	0
0	Espresso Macchiato	Small	True	0
3	Vanilla Latte	Small	True	15
4	Vanilla Cold Brew	Small	False	20
5	Vanilla Latte	Large	True	25
6	Vanilla Cold Brew	Large	False	30

Conclusion

This was a lot to take in, and hopefully the visuals helped a little bit. Be sure to practice these as that's the best way to understand these methods. This walkthrough doesn't cover every case, for instance we never pivoted using an aggregate function. These methods are very powerful and can be used in very unique ways. Knowing each method's strength and weakness will help you solve many of the problems to come.