Lab 8

Pandas III: Grouping and Presenting Data

Lab Objective: *Learn about Pivot tables, groupby, etc.*

Introduction

Pandas originated as a wrapper for numpy that was developed for purposes of data analysis. Data seldom comes in a format that is perfectly ready to use. We always need to be able to interpret what our data is telling us. Some data may be of little or no use, while other aspects of our data may be vital. *Pivoting* is an extremely useful way to sort through data and be able to present results clearly and compactly. Two central ways to accomplish this are by using `groupby` and by using *Pivot Tables*.

Groupby

In Lab 7 we introduced `pydatasets`, and mentioned how, in their raw format, plotting would be nonsensical. Many datasets are simply composed of tables of individuals (represented as rows), with a list of classifiers associated with each one (columns).

For example, consider the `msleep` dataset. In order to view the columns present in this dataset, we make use of the function `head()`. This will show us the first five rows, and all of the columns.

```
>>> import pandas as pd
>>> from pydataset import data
>>> msleep = data("msleep")
>>> msleep.head()
                        name       genus   vore         order  conservation
1                   Cheetah    Acinonyx  carni     Carnivora            lc
2                 Owl monkey       Aotus   omni      Primates           NaN
3            Mountain beaver  Aplodontia  herbi      Rodentia            nt
4  Greater short-tailed shrew     Blarina   omni  Soricomorpha            lc
5                       Cow         Bos  herbi  Artiodactyla  domesticated

   sleep_total  sleep_rem  sleep_cycle  awake  brainwt   bodywt
1         12.1        NaN          NaN   11.9      NaN   50.000
2         17.0        1.8          NaN    7.0  0.01550    0.480
```

```
3         14.4         2.4         NaN     9.6      NaN     1.350
4         14.9         2.3    0.133333     9.1  0.00029     0.019
5          4.0         0.7    0.666667    20.0  0.42300   600.000
```

Each row consists of a single type of mammal and its corresponding identifiers, including genus and order, as well as sleep measurements such as total amount of sleep (in hours) and REM sleep, in hours. When we try to plot this data using plt.plot, the individual data points do not demonstrate overall trends.

```
>>> msleep.plot(y="sleep_total", title="Mammalian Sleep Data", legend=False)
>>> plt.xlabel("Animal Index")
>>> plt.ylabel("Sleep in Hours")
>>> plt.show()
```

The above code results in Figure 8.1.

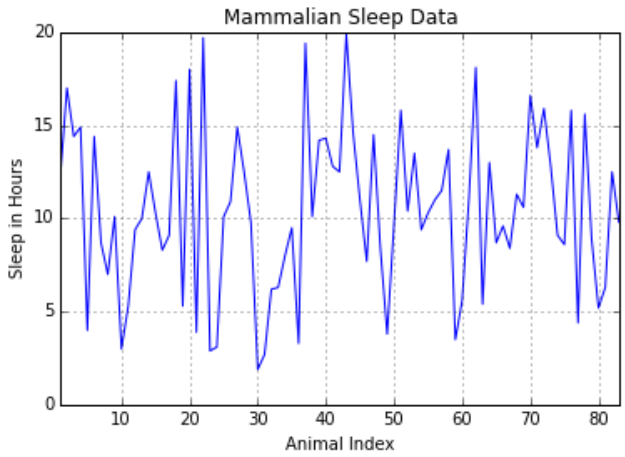

Figure 8.1: Source: Proceedings of the National Academy of Sciences, 104 (3):1051-1056, 2007. Updates from V. M. Savage and G. B. West, with additional variables supplemented by Wikipedia.

This set of connected data points is not particularly revealing, as it plots the first numerical column, `sleep_total`, as a function of the animal index, which is seemingly random.

The `DataFrame` contains information that will help us to make better sense of this data. Using `groupby()`, we can select the parts we want. As the name implies, `groupby()` takes a `DataFrame` and creates different groupings. For this dataset, let's consider the sleep differences between herbivores, omnivores, insectivores, and carnivores. To do so, we simply call the `groupby` method on the `vore` column to obtain a `groupby` object organized by diet classification.

```
>>> vore = msleep.groupby("vore")
```

You can also group the data by multiple columns, for example, both the `vore` and `order` classifications. To group and view this data, simply use:

```
>>> vorder = msleep.groupby(["vore", "order"])

# View groups within vorder
>>> vorder.describe()
```

This listing is much too long to view here, but you should be able to see that it first sorts all rows into the appropriate `vore` category, and then into the appropriate `order` category. It gives the count of rows in each section, along with the mean, standard deviation, and other potentially useful statistics.

Pandas `groupby` objects are not lists of new `DataFrames` associated with groupings. They are better thought of as a dictionary or generator-like object which can be *used* to produce the necessary groups. However, the `get_group()` method will do this, as follows:

```
# Get carnivore group
>>> Carni = vore.get_group("carni")
# Get herbivore group
>>> Herbi = vore.get_group("herbi")
```

The `groupby` object includes many useful methods that can help us make visual comparisons between groups. The `mean()` method, for example, returns a new `DataFrame` consisting of the mean values attached to each group. Using this method on our `vore` object returns a nicely organized `DataFrame` of the average sleep data for each mammalian diet pattern. We can similarly create a `DataFrame` of the standard deviations for each group.

At this point, we have a nicely organized dataset that can easily be turned into a bar chart. Here, we use the `DataFrame.loc` method to access three specific columns in the bar chart (`sleep_total`, `sleep_rem`, and `sleep_cycle`).

```
>>> means = vore.mean()
>>> errors = vore.std()
>>> means.loc[:,["sleep_total", "sleep_rem", "sleep_cycle"]].plot(kind="bar", ↩
    yerr=errors, title="Mean Mammallian Sleep Data")
>>> plt.xlabel("Mammal diet classification (vore)")
>>> plt.ylabel("Hours")
>>> plt.show()
```

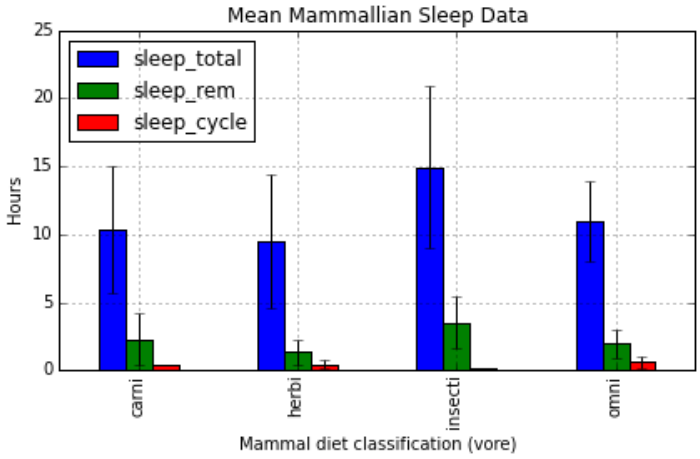

Figure 8.2: Source: Proceedings of the National Academy of Sciences, 104 (3):1051-1056, 2007. Updates from V. M. Savage and G. B. West, with additional variables supplemented by Wikipedia.

Problem 1. Examine the `diamonds` dataset found in the `pydataset` module. This dataset contains the identifiers and attributes of 53,940 individual round cut diamonds. Using the `groupby` method, create a visual highlighting and comparing different aspects of the data. This can be in the form of a single plot or comparative subplots. Use the plotting techniques from Lab 7.

Print a paragraph explaining what type of graph you used, why, and what we learn about the dataset from your plot. Don't forget to include titles, clear labels, and sourcing.

The following is an example of comparative subplots, which may *not* be used for your plot.

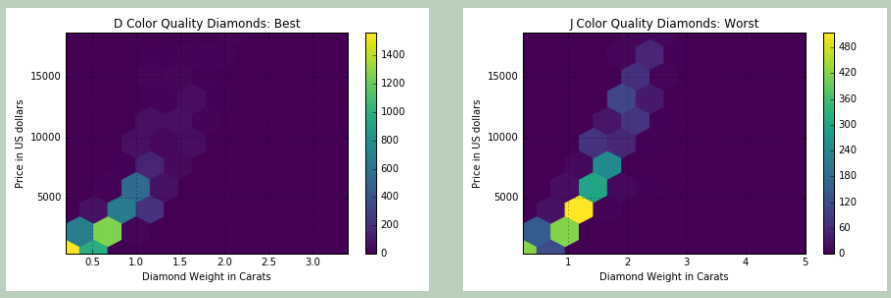

Figure 8.3: Source: Adopted from R Documentation

The following is an appropriate (if lengthy) description of our plots:

The above plots were created using `groupby` on the diamond colors and then using a hexbin comparing carats to price for the highest and lowest

quality diamonds, respectively. This hexbin is particularly revealing for each set of thousands of diamonds because it meaningfully displays concentration of datapoints. Matplotlib's new `viridis` colorplot, with a dark background, reveals bins that would have been invisible with a white background. By comparing these plots, we note that the greatest number of J quality diamonds in the dataset are about 1.25 carats and $4000 dollars in price, whereas the highest concentration of D quality diamonds are smaller and therefore cheaper. We may attribute this to D quality diamonds being rarer, but the colorbar on the side reveals that D diamond numbers are, in fact, far higher than those of the J color. Instead it is simply more likely that D quality diamonds of larger sizes are rarer than those of smaller sizes. Both hexbins reveal a linearity between diamond weight and diamond price, with D diamonds showing more variability and J diamonds displaying a strict linearity.

`Groupby` is a very useful method to order data. However, it is not the perfect tool for every situation. As we saw when sorting data by multiple columns, we can end up with a useful grouping, but too much information to display. If we want to see information in table format, we can make use Pivot Tables.

Pivot Tables

With a given pandas `DataFrame`, we can visualize data easily with the method `pivot_table()`. The Titanic dataset (`titanic.csv`) is especially apt for this. It includes many columns, but we will use only `Survived`, `Pclass`, `Sex`, `Age`, `Fare`, and `Embarked` here. You will need to load in this dataset for upcoming problems, using principles taught in Lab 6. Note that many of the ages are missing values. For our purposes, simply fill these missing values with the average age. Then drop any rows that are missing data. Once we have a usable dataset, we can make Pivot Tables to view trends in the data.

```
>>> titanic.pivot_table('Survived', index='Sex', columns='Pclass')
Pclass         1         2         3
Sex
female  0.962406  0.893204  0.473684
male    0.350993  0.145570  0.169540
```

This simple command makes a table with rows `Sex` and columns `Pclass`, and averages the result of the column `Survived`, thereby giving the percentage of survivors in each grouping. Note that this is similar to `groupby`: it sorts all entries into their gender and class, and the "function" it performs is getting the percentage of survivors.

This is interesting, and we can clearly see how much more likely females were to survive than males. But how does age factor into survival rates? Were male children really that likely to die as compared to females in general?

We can investigate these results by *multi-indexing*. We can pivot based on more than just two variables, by adding in another index. Note that in the original

dataset, the column `Age` has a floating point value for the age of each passenger. If we were to just add 'age' as an argument for index, then the table would create a new row for *each* age present. This wouldn't be a very useful or simple table to visualize, so we desire to partition the ages into 3 categories. We use the function `cut()` to do this.

```
>>> # Partition each of the passengers into 3 categories based on their age
>>> age = pd.cut(titanic['Age'], [0,12,18,80])
```

Now with this other partition of the column age, we can add this dimension to our pivot table, passing it along with `Sex` in a list to the parameter `index`.

```
>>> # Add a third dimension, age, to our pivot table
>>> titanic.pivot_table('Survived', index =['Sex', age], columns='Pclass')
Pclass                  1         2         3
Sex    Age
female (0, 12]   0.000000  1.000000  0.466667
       (12, 18]  1.000000  0.875000  0.607143
       (18, 80]  0.966667  0.878049  0.436170
male   (0, 12]   1.000000  1.000000  0.342857
       (12, 18]  0.500000  0.000000  0.081081
       (18, 80]  0.328671  0.087591  0.159420
```

What do we notice? First of all, male children (ages 0 to 12) in the 1st and 2nd class were very likely to survive, whereas those in 3rd class were much less likely to. However, look at the female children in first class. It says that zero percent survived. This might seem a little odd, but if we looked at our data set again to see how many passengers fell into this category of female, 1st class, and age 0 to 12, we would find only one passenger. Therefore, the statistic that 0% of female children in first class lived is misleading. We can see the number of people in each category by simply specifying the parameter aggfunc='count'.

```
>>> titanic.pivot_table('Survived', index=['Sex',age], columns='Pclass', aggfunc=↩
    'count')
Pclass             1    2    3
Sex    Age
female (0, 12]     1   13   30
       (12, 18]   12    8   28
       (18, 80]  120   82   94
male   (0, 12]     4   11   35
       (12, 18]    4   10   37
       (18, 80]  143  137  276
```

The parameter aggfunc defaults to 'mean', which is why we have seen the mean survival rate for each of the different categories. By specifying aggfunc to be 'count', we get how many passengers of each category are present in the table. In this case, specifying the value is 'survived' is redundant. We get the same table if we put in `Fare` in place of `Survived`.

This table brings up another point about partitioning datasets. This Titanic dataset includes data for only about 1000 passengers. This is not that large of a sample size, relatively speaking, and significant sample sizes aren't guaranteed for each possible partitioning of the the data columns. It is a good practice to ask questions about the numbers you see in pivot tables before making any conclusions.

Now, for fun, let's add a fourth dimension to our table–the cost of the passenger's ticket. Let's add this dimension to our columns of the pivot table. We now use the function `qcut()` to partition the fare column's data into two different categories.

```
>>> # Partition fare column into 2 categories based on the values present in fare↩
    column
>>> fare = pd.qcut(titanic['fare'], 2)
>>> # Add the fare as a dimension of columns
>>> titanic.pivot_table('Survived', index=['Sex',age], columns=[fare, 'Pclass'])
Fare                    [0, 15.75]                    (15.75, 512.329]
Pclass              1         2         3           1           2           3
Sex       Age
female   (0, 12]    NaN  1.000000  0.600000    0.000000    1.000000    0.400000
        (12, 18]    NaN  0.750000  0.666667    1.000000    1.000000    0.250000
        (18, 80]    NaN  0.875000  0.434783    0.966667    0.880000    0.440000
male     (0, 12]    NaN  1.000000  0.636364    1.000000    1.000000    0.208333
        (12, 18]    NaN  0.000000  0.115385    0.500000    0.000000    0.000000
        (18, 80]    0.2  0.113636  0.153846    0.333333    0.040816    0.206897
```

We now see something else about our dataset–we get NaNs in some entries. The dataset has some invalid entries or incomplete data for the column fare. Let's investigate further.

By inspecting our dataset, we can see that there might not be any people in the categories given in the pivot table. We can fill in the NaN values in the pivot table with the argument `fill_value`. We show this below.

```
>>> # Specify fill_value = 0.0
>>> titanic.pivot_table('survived', index=['sex',age], columns=[fare, 'class'], ↩
    fill_value=0.0)
Fare                    [0, 15.75]                    (15.75, 512.329]
Pclass              1         2         3           1           2           3
Sex       Age
female   (0, 12]    0.0  1.000000  0.600000    0.000000    1.000000    0.400000
        (12, 18]    0.0  0.750000  0.666667    1.000000    1.000000    0.250000
        (18, 80]    0.0  0.875000  0.434783    0.966667    0.880000    0.440000
male     (0, 12]    0.0  1.000000  0.636364    1.000000    1.000000    0.208333
        (12, 18]    0.0  0.000000  0.115385    0.500000    0.000000    0.000000
        (18, 80]    0.2  0.113636  0.153846    0.333333    0.040816    0.206897
```

It should be noted though that with datasets where NaN's occur frequently, some preprocessing needs to be done so as to get the most accurate statistics and insights about your dataset. You should've worked on cleaning datasets in previous lab, so we don't go further into detail about the topic here.

Problem 2. Suppose that someone claims that the city from which a passenger embarked had a strong influence on the passenger's survival rate. Investigate this claim.

1. Using a `groupby()` call, see what the survival rates of the passengers were based on where they embarked from (`embark_town`).
2. Next, create a pivot table to further look at survival rates based on both place of embarkment and gender.

3. What do these tables suggest to you about the significance of where people embarked in influencing their survival rate? Examine the context of the problem, and explain what you think this really means. Find 2 more pivot tables that investigate the claim further, looking at other criterion (e.g., class, age, etc.). Include explanations.

As you have learned in the pandas labs, pandas is a powerful tool for data processing, and has a plethora of built-in functions that greatly simplify data analysis. Ler's now put the skills you have acquired to practical use.

Problem 3. Search through the `pydatasets` and find one which you would like to present. Make sure there is enough data to produce a proper presentation. Process the data appropriately, and create your own presentation, as though you were presenting to a superior in the appropriate line of work. Your presentation can be in any format you like, but as a bare minimum, it must include the following:

- An appropriate title.
- 3 charts or visuals with captions.
- 3 tables with captions.
- 2 paragraps of textual explanations.
- Appropriate data sourcing.