Data Analysis

Brandon Krakowsky

Penn Engineering

Loading Data – pandas Module

- The *pandas* module provides data analysis tools
- It can connect to and interact with a database
- It can also read and write Excel files
- It provides a useful *read_excel* method which reads an Excel file into a *DataFrame*

```
import pandas as pd
df = pd.read_excel('my_file.xlsx') #read the file into a DataFrame
```

- A *DataFrame* is a 2-dimensional labeled data structure
  - You can think of it like a spreadsheet or database table

For reference:

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_excel.html

Loading Data – pandas Module

- The *pandas* module also provides a useful *ExcelFile* class with a *parse* method that can read individual sheets in an Excel file

```
import pandas as pd
xls = pd.ExcelFile('my_file.xlsx')
df = xls.parse('my_sheet') #read the sheet into a DataFrame
```

- We'll use this for loading our data
  - Confirm you've downloaded the 'yelp.xlsx' file

For reference:

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.ExcelFile.parse.html

Our Data – Yelp Dataset

- Information about local businesses in 13 cities in PA and NV
- Courtesy Yelp Dataset Challenge (https://www.yelp.com/dataset_challenge)
- "yelp_data" tab data columns:
  - name: Name of business
  - category_0: 1st user-assigned business category
  - category_1: 2nd user-assigned business category
  - take-out: Flag (True/False) indicating if business provides take-out
  - review_count: Number of reviews
  - stars: Overall star rating
  - city_id: Identifier referencing city of business (match to *id* on "cities" tab)
  - state_id: Identifier referencing state of business (match to *id* on "states" tab)

name	category_0	category_1	take_out	review_count	stars	city_id	state_id
Elite Coach Limousine	Hotels & Travel	Airport Shuttles	FALSE	3	2.5	8	1
China Sea Chinese Restaurant	Restaurants	Chinese	TRUE	11	2.5	1	1
Discount Tire Center	Tires	Automotive	FALSE	24	4.5	1	1

Our Data – Yelp Dataset

- “cities” tab data columns:
  - id: Unique identifier of city
  - city: City name

id	city
1	Bellevue
7	Munhall
8	Pittsburgh
9	West Homestead
10	West Mifflin
11	Henderson
12	Las Vegas

- “states” tab data columns:
  - id: Unique identifier of state
  - state: State name

id	state
1	PA
2	NV

Loading Data – pandas Module

- Let's load and read the 'yelp.xlsx' file

```
import pandas as pd
xls = pd.ExcelFile('yelp.xlsx')
df = xls.parse('yelp_data') #read the "yelp_data" sheet into a DataFrame
```

- *df* is a DataFrame

```
type(df)
```

- Get a count of rows

```
len(df)
```

- Get the size (rows, columns)

```
df.shape
```

Inspecting Data - DataFrame

- Get a count of values in each column
  `df.count()`
- You can look at the column headers by accessing the *columns* attribute
  `df.columns`
- And the type of data stored in each column by accessing the *dtypes* attribute
  `df.dtypes`

For reference: http://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe

Inspecting Data - DataFrame

- Provides various summary statistics for the numerical values in *DataFrame*
  `df.describe()`
- Quickly examine the first 5 rows of data
  `df.head()`
- Or the first 100 rows of data
  `df.head(100)`
- Drop the duplicates (based on all columns) from df
  `df = df.drop_duplicates()`

For reference: http://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe

Querying Data

- Select just the business names using the “name” of the attribute in between square brackets []
```
df[”name"] #returns name for every record
```
- Query the location for the first 100 businesses
```
atts = ["name", "city_id", "state_id”] #store the list of attributes in a list
df[atts].head(100)
```
  - This only shows the id for each city and state
  - How can we get the actual values?

Joining Data

- We need to look up the values in the “cities” sheet and “states” sheet
  - Then combine them with the data in the “yelp_data” sheet
- We do this by *joining* the datasets using a common field (identifier) in each
  - Note: This process of joining tables is similar to what we do with tables in a relational database
- The *city_id* in “yelp_data” will *join* to the *id* in “cities”
- The *state_id* in “yelp_data” will *join* to the *id* in “states”

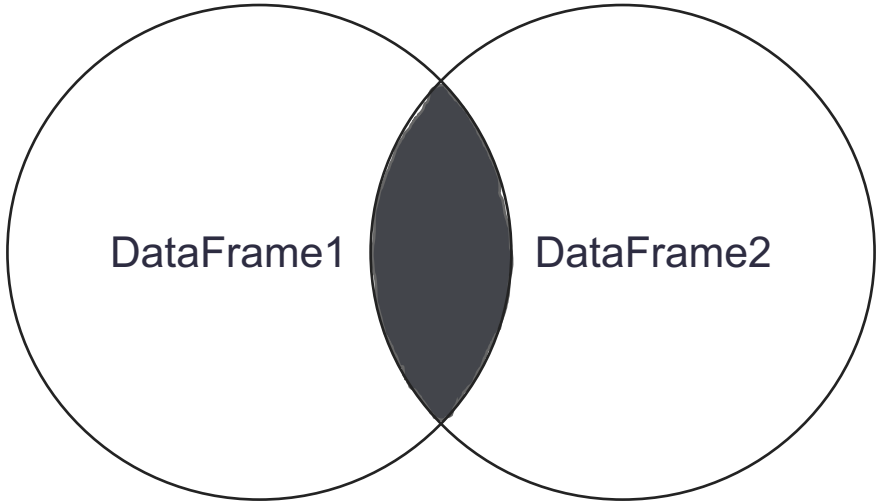
id	city
1	Bellevue
7	Munhall
8	Pittsburgh
9	West Homestead
10	West Mifflin
11	Henderson
12	Las Vegas

name	category_0	category_1	take_out	review_count	stars	city_id	state_id
Elite Coach Limousine	Hotels & Travel	Airport Shuttles	FALSE	3	2.5	8	1
China Sea Chinese Restaurant	Restaurants	Chinese	TRUE	11	2.5	1	1
Discount Tire Center	Tires	Automotive	FALSE	24	4.5	1	1

id	state
1	PA
2	NV

Joining Data

- The most common type of join is called an *inner join*
  - Combines *DataFrames* based on a *join* key (common identifier)
  - Returns a new *DataFrame* that contains only those rows where the value being *joined* exists in BOTH tables



Joining Data

- Import the “cities” sheet into it’s own *DataFrame* using the *parse* method
  ```
  df_cities = xls.parse('cities’)
  ```
- The pandas function for performing joins is called *merge*
  - Specify the *DataFrames* to join in the “left” and “right” arguments
  - Specify inner (the default option) for the “how” argument
  - Specify the join keys in the “left_on” and “right_on” arguments
    ```
    df = pd.merge(left=df, right=df_cities, how='inner',
    left_on='city_id', right_on='id’)
    ```
- What’s the new size (rows, columns) of df?
  ```
  df.shape
  ```
- Now we can see the cities in df
  ```
  df.head()
  ```

Joining Data - Exercise

- Import the “states” sheet into it’s own *DataFrame*
- Join (merge) with df
- Calculate the new size (rows, columns) of df
- Show the name, city, and state for the first 100 businesses

Joining Data - Exercise

- Import the “states” sheet into it’s own *DataFrame*
```
df_states = xls.parse(’states’)
```
- Join (merge) with df
```
df = pd.merge(left=df, right=df_states, how='inner',
left_on=’state_id', right_on='id’)
```
- Calculate the new size (rows, columns) of df
```
df.shape
```
- Show the name, city, and state for the first 100 businesses
```
atts = ["name", "city", "state”] #store the list of attributes in a
list
df[atts].head(100)
```

Querying Data - Slicing Rows

- You can get a *slice* of a DataFrame by using a colon (:)
- Format: [*start_index*:*end_index*]
  - *start_index* and *end_index* are both optional
  - *start_index* is the index of the first value (included in slice)
  - *end_index* is the index of the last value (not included in slice)

Querying Data - Slicing Rows

- Get the 2nd 100 businesses listed in the data

```
df[100:200] #returns rows 100 to 199
```

- Get the name of the last business listed in the data

```
last_index = len(df) - 1
last_business = df[last_index:] #returns the last row
last_business[“name”] #returns the “name” of the last row
```

  - Another way …

```
df[-1:][“name”] #returns the “name” of the last row
```

Querying Data - Conditions Using Boolean Indexing

- To filter a DataFrame using *Boolean Indexing*, you first create a *Series*, a one-dimensional array
  - Each element in the *Series* has a value of either True or False
  - *Boolean Indexing* compares each value in the *Series* to each record in the DataFrame

For reference: http://pandas.pydata.org/pandas-docs/stable/indexing.html

Querying Data - Conditions Using Boolean Indexing

- Select the businesses in Pittsburgh
```
pitts = df["city"] == "Pittsburgh” #creates a Series with True/False values
```
- The type is Series
```
type(pitts)
```
- You can see the True/False values
```
print(pitts)
```
- Filter the elements in df
```
df[pitts] #filters df based on the True/False values in the pitts Series
```

For reference: http://pandas.pydata.org/pandas-docs/stable/indexing.html

Querying Data - Conditions Using Boolean Indexing

- Does the "The Dragon Chinese Cuisine" offer take out?

```
rest = df["name"] == "The Dragon Chinese Cuisine" #creates the series
bus = df[rest]["take_out"] #filters the df and returns the "take_out" column
```

For reference: http://pandas.pydata.org/pandas-docs/stable/indexing.html

Querying Data - Conditions Using Boolean Indexing

- Select the bars

```
cat_0_bars = df["category_0"] == "Bars”
cat_1_bars = df["category_1"] == "Bars”
df[cat_0_bars | cat_1_bars] #returns rows where cat_0_bars is True |
(OR) cat_1_bars is True
```

- Select the bars in Carnegie

```
cat_0_bars = df["category_0"] == "Bars”
cat_1_bars = df["category_1"] == "Bars”
carnegie = df["city"] == "Carnegie”
df[(cat_0_bars | cat_1_bars) & carnegie] #returns rows where cat_0_bars
is True | (OR) cat_1_bars is True & (AND) carnegie is True
```

For reference: http://pandas.pydata.org/pandas-docs/stable/indexing.html

Querying Data - Conditions Using Boolean Indexing

- Select the bars and restaurants in Carnegie

```
cat_0 = df["category_0"].isin(["Bars", "Restaurants"]) #tests if
category_0 is in the provided list
cat_1 = df["category_1"].isin(["Bars", "Restaurants"]) #tests if
category_1 is in the provided list
carnegie = df["city"] == "Carnegie”
df[(cat_0 | cat_1) & carnegie]
#returns rows where cat_0 is True | (OR) cat_1 is True & (AND) carnegie
is True
```

For reference: http://pandas.pydata.org/pandas-docs/stable/indexing.html

Querying Data - Exercise

- How many total dive bars are there in Las Vegas?
  - Assume a Yelp user assigned “Dive Bars” in the *category_0* column or *category_1* column
- Recommend a random dive bar with at least a 4 star rating
  - Look at the total set of dive bars above and query for those that have a *star* rating of at least 4.0
  - Import the random module: import random
  - Get a random number using the randint method
  - Get a random dive bar from the set above using the random number

For reference: https://docs.python.org/3/library/random.html

Querying Data - Exercise

- How many total dive bars are there in Las Vegas?

```
#create 3 series from df
lv = df["city"] == "Las Vegas”
cat_0_bars = df["category_0"] == "Dive Bars”
cat_1_bars = df["category_1"] == "Dive Bars”

#filter df using 3 series
divebars_lv = df[lv & (cat_0_bars | cat_0_bars)]

#print length of dive bars in LV (divebars_lv)
print("There are", len(divebars_lv), "dive bar(s) in LV”)
```

Querying Data - Exercise

- Recommend a random dive bar with at least a 4 star rating

```
stars = divebars_lv["stars"] >= 4.0 #create new series from divebars_lv
divebars_4rating_lv = divebars_lv[stars] #filter divebars_lv

import random #import random module
#get random number between (but including) 0 and last index
rand_int = random.randint(0, len(divebars_4rating_lv) - 1)

#get random dive bar based on random number
rand_divebar = divebars_4rating_lv[rand_int:rand_int + 1]
#another way, is by using the iloc method to choose rows (and/or columns) by
position
#rand_divebar = divebars_4rating_lv.iloc[rand_int, :]

#show random dive bar with at least 4 star rating
rand_divebar
```

For reference: https://docs.python.org/3/library/random.html

Computations – *sum()*

- Calculate the total (sum) number of reviews for nail salons in Henderson

```
cat_0 = df["category_0"].str.contains("Nail Salon”) #tests if category_0 string value contains “Nail Salon”
cat_1 = df["category_1"].str.contains("Nail Salon”) #tests if category_1 string value contains “Nail Salon”
henderson = df["city"] == "Henderson”
df[(cat_0 | cat_1) & henderson]["review_count"].sum()  #returns the total “review_count” value for the rows where cat_0 is True | (OR) cat_1 is True & (AND) henderson is True
```

For reference: http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-stats

Computations – *mean()*

- Calculate the average (mean) star rating for auto repair shops in Pittsburg?

```
cat_0 = df["category_0"].str.contains("Auto Repair”)
cat_1 = df["category_1”].str.contains("Auto Repair”)
pitts = df["city"] == 'Pittsburgh’
df[(cat_0 | cat_1) & pitts]["stars"].mean() #returns the average
“stars” value for the rows where cat_0 is True | (OR) cat_1 is True &
(AND) pitts is True
```

For reference: http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-stats

Other Methods

- What cities are in our dataset?
  `df["city"].unique() #returns the unique values in city`
- How many businesses are there in each *city*?
  `df['city'].value_counts() #counts the records for each city`
- How many unique user assigned business categories are there in *category_0*?
  `df['category_0'].nunique() #counts the non-null unique values in category_0`

For reference: http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-stats

Updating & Creating Data

- You can easily create new columns in *DataFrames* by just naming and using them
  - Example: my_df[”new_column”] = <some_value>
- Add a new “categories” column that combines “category_0” and “category_1” as a comma-separated list

```
df["categories"] = df["category_0"].str.cat(df["category_1"], sep=',’)
#concatenates the string value of category_0 with category_1, separated by a ‘,’
```

- Now we can look up businesses based on the single “categories” column!

```
df[df["categories"].str.contains("Pizza")]
```

Updating & Creating Data - Exercise

- Add a new “rating” column that converts “stars” to a comparable value in the 10-point system
  ```
  df["rating"] = df["stars"] * 2
  ```
- Now, update the new “rating” column so that it displays the rating as ”x out of 10”
- First, create a helper function that will take a rating value as an argument and concatenate a string to it
  ```
  def convert_to_rating(x):
      return (str(x) + " out of 10") #casts x (rating) to a string, then concatenates another string
  ```
- Second, use the *apply()* method to run the helper function for the rating in each row
  ```
  df["rating"] = df[”rating"].apply(convert_to_rating) #applies function

  df.head()
  ```

Querying Data - Summarizing Groups

- *groupby* splits data into different groups based on variable(s) of your choice
  - Note: This process of grouping data is similar to what we do with *Group By* in SQL
- *groupby* returns a *GroupBy* object, which provides a dictionary whose keys are the computed unique groups and corresponding values
- When we group by “city”, the *keys* will be all possible cities
  `df.groupby(['city']).groups.keys()`
- We can use the *groups* attribute to get a specific group of records. How many businesses in Las Vegas?
  `len(df.groupby([‘city']).groups[’Las Vegas’])`
- But this isn’t super useful!
  - We really want to be able to perform aggregate computations on the data in each group

Querying Data – agg()

- The *agg()* method performs aggregate computations on the data in each group
  - You can pass a *list* of aggregate functions as arguments
  - Use methods from NumPy, a popular package for scientific computing (http://www.numpy.org/)
- Let's find out the *sum*, *mean*, and *standard deviation* for the star ratings of each city

```
import numpy as np
df.groupby(['city']).agg([np.sum, np.mean, np.std])["stars"]
```

Pivot Tables

Pivot Tables – Using index

- A *pivot table* is a useful data summarization tool that creates a new table from the contents in the DataFrame
- In order to *pivot* the DataFrame, we need at least one *index* column, to group by
  - The *index* is just like the variable(s) you group by in the *groupby* method
  - The *pivot table* will provide useful summaries along that *index*, such as summation or average

Pivot Tables – Using index

- Let's use city as the index

```
pivot_city = pd.pivot_table(df,index=["city"])
pivot_city
```

- The type of *pivot_city* is still a DataFrame

```
type(pivot_city)
```

	city_id	id_x	id_y	review_count	stars	state_id	take_out
city							
Bellevue	1.0	1.0	1.0	13.166667	3.750000	1.0	0.500000
Braddock	2.0	2.0	1.0	14.500000	4.750000	1.0	0.500000
Carnegie	3.0	3.0	1.0	13.590909	3.454545	1.0	0.409091
Henderson	11.0	11.0	2.0	33.323077	3.419231	2.0	0.238462

- Note: By default, the pivot table calculates average (mean) for each column

Pivot Tables – Using index

- It is also possible to use more than one *index* (*indices*)
  - The pivot table will sort the data for you
- Use indices “state” and “take_out”

```
pivot_state_take = pd.pivot_table(df,index=["state", ”take_out"])

pivot_state_take
```

		city_id	id_x	id_y	review_count	stars	state_id
state	**take_out**						
NV	**False**	11.698276	11.698276	2.0	16.900862	3.409483	2.0
	True	11.661765	11.661765	2.0	118.161765	3.198529	2.0
PA	**False**	6.643678	6.643678	1.0	11.580460	3.695402	1.0
	True	6.769841	6.769841	1.0	49.936508	3.535714	1.0

- The cells display the average (mean) values for each ”take_out” value in each “state”

Pivot Tables - Exercise

- Create a pivot table that displays the average (mean) review count and star rating for bars and restaurants in each city (only use *category_0* for simplicity)
  - Hints:
    - Filter for “Bars” and “Restaurants”
    - Pivot on *state*, *city*, and *category_0*

```
#Make a dataframe that only contains bars and restaurants

rest = df["category_0"].isin(["Bars", "Restaurants"])

df_rest = df[rest]

#Pivot along state, city, and category_0

pivot_state_cat = pd.pivot_table(df_rest,index=["state", "city", "category_0"])

#Bonus: since the pivot table is also a dataframe, we can filter the columns

pivot_state_cat[["review_count", "stars"]]
```

Pivot Tables - Exercise

- Since the pivot table shows the average (mean) value by default, we now have a new table of average review count and star ratings for each city.

			review_count	stars
state	city	category_0		
NV	Henderson	Bars	171.000000	3.000000
		Restaurants	102.454545	3.181818
	Las Vegas	Bars	15.500000	4.000000
		Restaurants	221.153846	3.153846
	North Las Vegas	Bars	7.000000	3.500000
		Restaurants	12.000000	3.000000
PA	Bellevue	Restaurants	14.000000	3.916667
	Braddock	Bars	26.000000	4.500000
	Carnegie	Bars	16.500000	4.000000
		Restaurants	26.000000	3.125000
	Homestead	Bars	23.000000	2.500000
		Restaurants	6.000000	2.500000
	Mc Kees Rocks	Bars	9.000000	3.500000
		Restaurants	7.333333	3.333333

Pivot Tables – aggfunc()

- To display summary statistics other than the average (mean)
  - Use the *aggfunc* parameter to specify the aggregation function(s)
  - Use the *values* parameter to specify the column(s) for the *aggfunc*
- As before, use the aggregation methods from the NumPy package
- In our dataset, how many (sum) reviews does each city have?

```
import numpy as np
pivot_agg = pd.pivot_table(
      df,index=["state", "city"],
      values=["review_count"], #Specify the column(s) for the aggfunc
      aggfunc=[np.sum] #Specify the aggregation function(s)
)
pivot_agg
```

Pivot Tables – aggfunc()

- It's possible to further *segment* our results using the "columns" parameter

```
pivot_a2 = pd.pivot_table(
        df,index=["state", "city"],
        values=["review_count"],
        #Specify the columns to separate the results
        columns=["take_out"],
        aggfunc=[np.sum]
)
pivot_a2
```

		sum
		review_count
state	city	
NV	Henderson	4332
	Las Vegas	7226
	North Las Vegas	398
	Bellevue	158
	Braddock	29
	Carnegie	299

		sum	
		review_count	
	take_out	False	True
state	city		
NV	Henderson	2009	2323
	Las Vegas	1619	5607
	North Las Vegas	293	105
	Bellevue	52	106
	Braddock	3	26
	Carnegie	74	225

Pivot Tables – aggfunc()

- We can also pass as an argument to *aggfunc()*, a *dict* object containing different aggregate functions to perform on different values
- If we want to see the total number of review counts and average ratings

```
pivot_agg3 = pd.pivot_table(
     df,index=["state", "city"],
     columns=["take_out"],
     aggfunc={"review_count":np.sum, "stars":np.mean} #Let's use np.sum
for review_count and np.mean for stars

)

pivot_agg3
```

Visualization

Jupyter Notebook – “Magic Functions”

- Jupyter Notebook (iPython) has built-in “magic functions” that are helpful for handling “meta” (other) functionalities in Python
- The magic function “%pylab inline” allows the PyLab library to load and lets our visualizations show up inside our notebook
- Run this:

```
%pylab inline
```

Visualization - matplotlib

- *matplotlib* is a popular plotting library for Python, which is included in the PyLab package
  - It's a very powerful tool that can plot a large variety of figures (and even animate them)
- Examples covered:
  - Histograms: Represents the distribution of values in the data
  - Scatterplots: Displays a set of data points (observations) for 2 variables (bivariate), in an x-y plane

Documentation for pyplot: http://matplotlib.org/api/pyplot_api.html

Visualization - Histogram

- How do review patterns differ across different cities? Which city has a higher number of 5 star reviews or 1 star reviews, Pittsburgh or Las Vegas?
- Let's find out by plotting a *histogram* that compares the distribution of *rating* scores between businesses in Pittsburgh and Las Vegas
- This can be done in two steps:
    1. Prepare the data: create appropriate DataFrame or Series
    2. Plot: set various options for *pyplot*

Visualization - Histogram

- How do review patterns differ across different cities? Which city has a higher number of 5 star reviews or 1 star reviews, Pittsburgh or Las Vegas?
- Let's find out by plotting a *histogram* that compares the distribution of *rating* scores between businesses in Pittsburgh and Las Vegas
- This can be done in two steps:
  1. Prepare the data: create appropriate DataFrame or Series
  2. Plot: set various options for *pyplot*
- Step 0: import

```
#Import pyplot

import matplotlib.pyplot as plt
```

Visualization - Histogram

- Step 1: data prep

```
#Create new dataframes for each city

df_pitt = df[df["city"] == 'Pittsburgh']

df_vegas = df[df["city"] == 'Las Vegas']

df_vegas


#Extract the ”stars” column into series

series_vegas = df_vegas["stars"]

series_pitt = df_pitt["stars"]

series_vegas
```

- Now we are ready to plot!

Visualization - Histogram

- Step 2: Let's place a *histogram* for the Pittsburgh business ratings

```
plt.hist(
      series_pitt, #star ratings for Pittsburgh
      alpha=0.3, #transparency
      color='yellow', #histogram color
      label='Pittsburgh', #label
      bins='auto' #sets the bins (dividers for the x-axis automatically
)

plt.show() #show the plot
```

Visualization - Histogram

- Add another *histogram* for Las Vegas

```
# Same as before, but with red
plt.hist(series_vegas, alpha=0.3, color='red', label='Vegas’,
bins=‘auto’)
```

- Now we have two *histograms* inside our plot!
  - Note: Always make sure to re-run the *plt.show()* method

```
plt.show() #show the plot
```

- Let’s add more features to our plot

Visualization - Histogram

- We can add labels for our x and y axes

```
plt.xlabel('Rating') # x-axis represents each rating score
plt.ylabel('Number of Rating Scores') # y-axis represents the number rating scores
```

- Add a legend

```
plt.legend(loc='best') #Location for legend: 'best' lets PyPlot decide where to place the legend
```

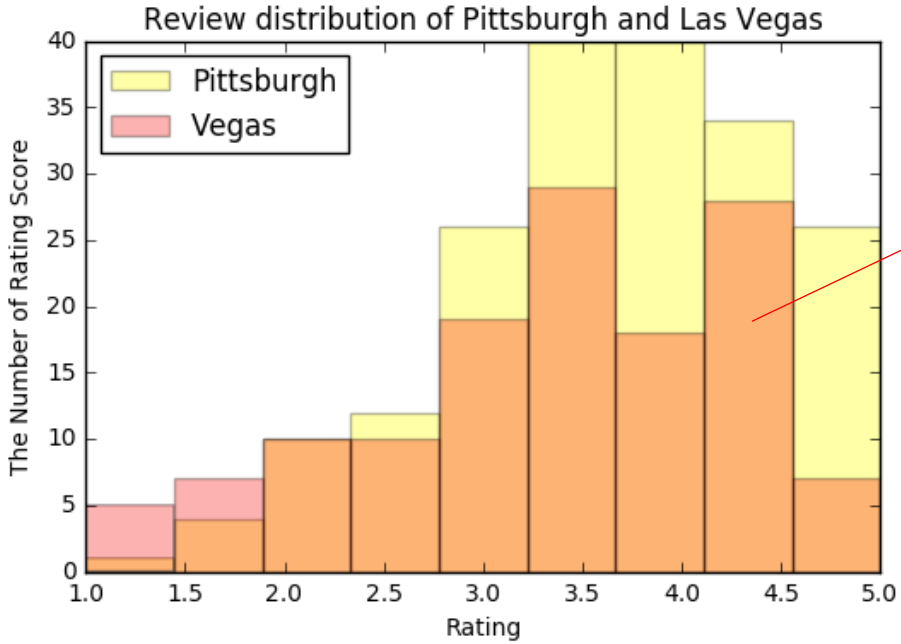
- Add a title

```
plt.title("Review distribution of Pittsburgh and Las Vegas")
```

Visualization - Histogram

- When everything is set, we run *plt.show()* to display our work!

```
plt.show()
```



The "orange" portion represents the overlap between Pittsburgh and Las Vegas

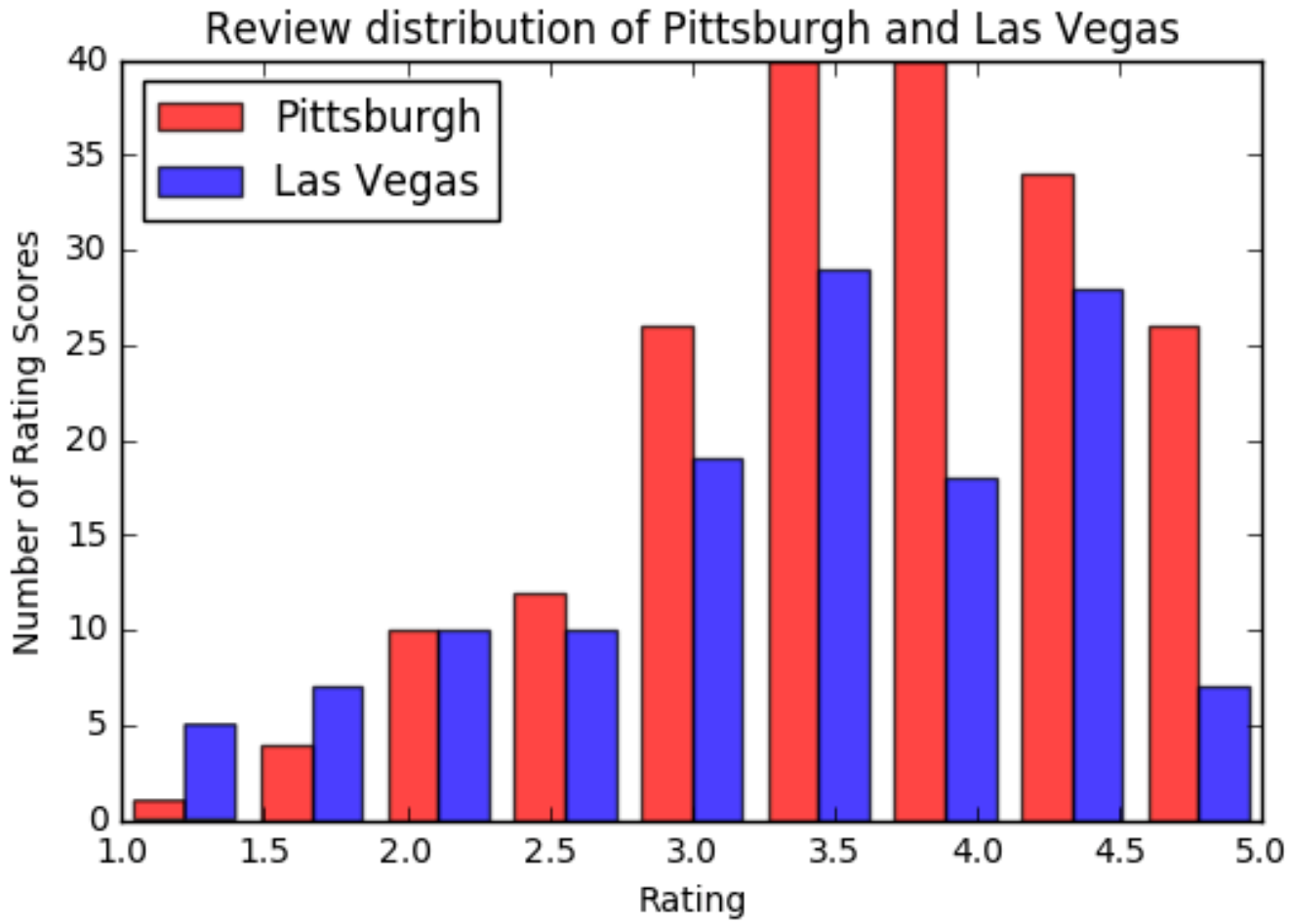
- Try tweaking the parameters to adjust the plot to your preference

Visualization - Histogram

- Having two plots overlap doesn't seem very appealing. Let's place the *histogram* bars side by side using *lists*
- This time, we only have one *histogram* object with a *list* of values

```
plt.hist([series_pitt, series_vegas], alpha=0.7, color=['red','blue'],
     label=['Pittsburgh','Las Vegas'],bins="auto")
plt.xlabel('Rating')
plt.ylabel('Number of Rating Scores')
plt.legend(loc='best')
plt.title("Review distribution of Pittsburgh and Las Vegas")
plt.show()
```

Visualization - Histogram



Visualization - Scatterplot

- Scatterplot displays a set of data points (observations) for 2 variables (bivariate), in an x-y plane
- We can use scatterplot to compare multiple categories on two different dimensions
- This time, let's visualize the review counts and star ratings for businesses in the following categories: 'Health & Medical', 'Fast Food', 'Breakfast & Brunch'

```
# Creating new dataframes for each category

df_health = df[df["category_0"] == 'Health & Medical']

df_fast = df[df["category_0"] == 'Fast Food']

df_brunch = df[df["category_0"] == 'Breakfast & Brunch']
```

Visualization - Scatterplot

- We can place a *scatterplot* object in the plot using *plt.scatter()*

```
# The first two arguments are the data for the x and y axis, respectively
plt.scatter(df_health['stars'], df_health['review_count'],
        marker='o', #marker shape
        color='r', #color: red
        alpha=0.7, #alpha (tranparency)
        s = 124, #size of each marker
        label=['Health & Medical'] #label
)
```

Visualization - Scatterplot

- Place two more *scatterplot* objects within the plot

```
# The first two arguments are the data for the x and y axis, respectively
plt.scatter(
      df_fast['stars'], df_fast['review_count'],
      marker='h’,
      color='b’,
            alpha=0.7,
      s = 124,
      label=['Fast Food']
)
plt.scatter(df_brunch['stars'],df_brunch['review_count'],
      marker=‘^',
      color='g',
      alpha=0.7,
      s = 124,
      label=['Breakfast & Brunch']
)
```

Visualization - Scatterplot

- Add the axis labels and the legend

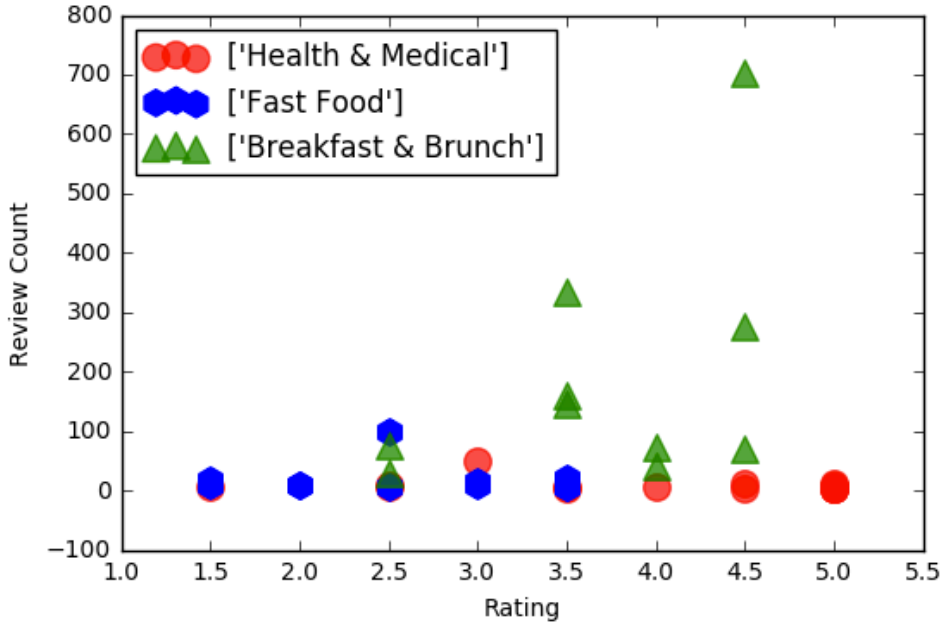
```
plt.xlabel('Rating’)

plt.ylabel('Review Count’)

plt.legend(loc='upper left’)
```

- Show the plot!

```
plt.show()
```

Visualization - Scatterplot

- We now have a visualization of review count and star rating for each category.
- Each marker represents a single business. We can observe some clustering in review count and star rating, depending on the categories.



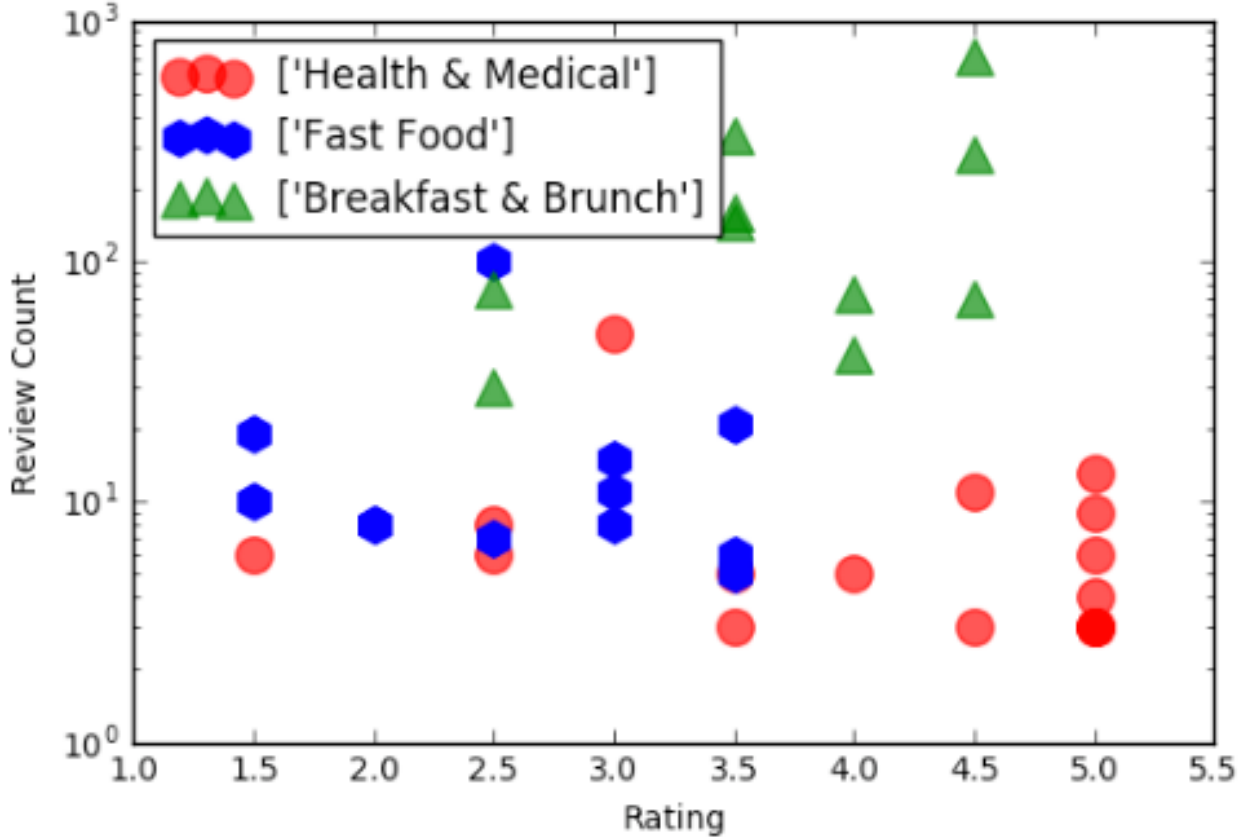
Visualization - Scatterplot

- The outliers in the y axis (review_count) are squashing the other data points
  - Let's make a little tweak to represent the y-axis in a *logarithmic* scale, so that review count is displayed in an *order of magnitude* (groups of 10)
  - All the values will be visualized in a more manageable range
- Get the *axes* attribute from pyplot to define the scale of each axis

```
axes = plt.gca() # get current axes

axes.set_yscale('log') # set the scale of the y-axis as logarithmic

plt.show()
```

Visualization - Scatterplot

- Looks much better!



For Reference: Seaborn

- Seaborn is a Python data visualization library based on matplotlib
  - It provides a high-level interface for drawing *more attractive and informative statistical graphics*
  - It can also be used to *enhance matplotlib graphics*
- More info: https://seaborn.pydata.org/