Relational Databases

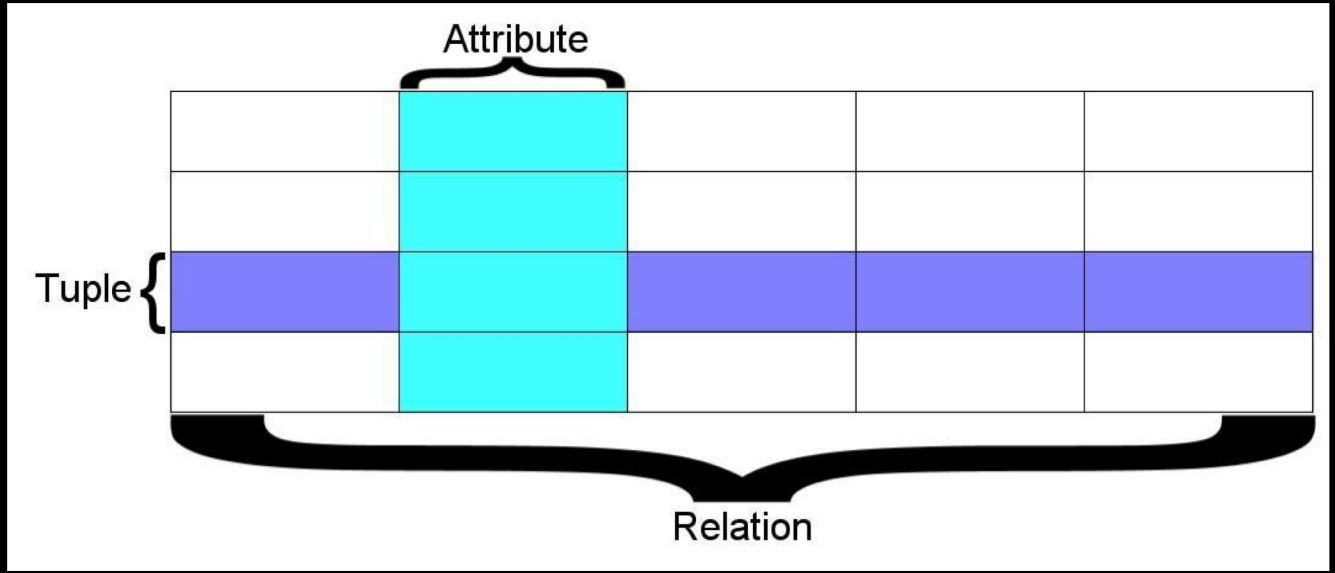
Relational databases model data by storing rows and columns in tables. The power of the relational database lies in its ability to efficiently retrieve data from those tables and in particular where there are multiple tables and the relationships between those tables involved in the query.

http://en.wikipedia.org/wiki/Relational_database

Terminology

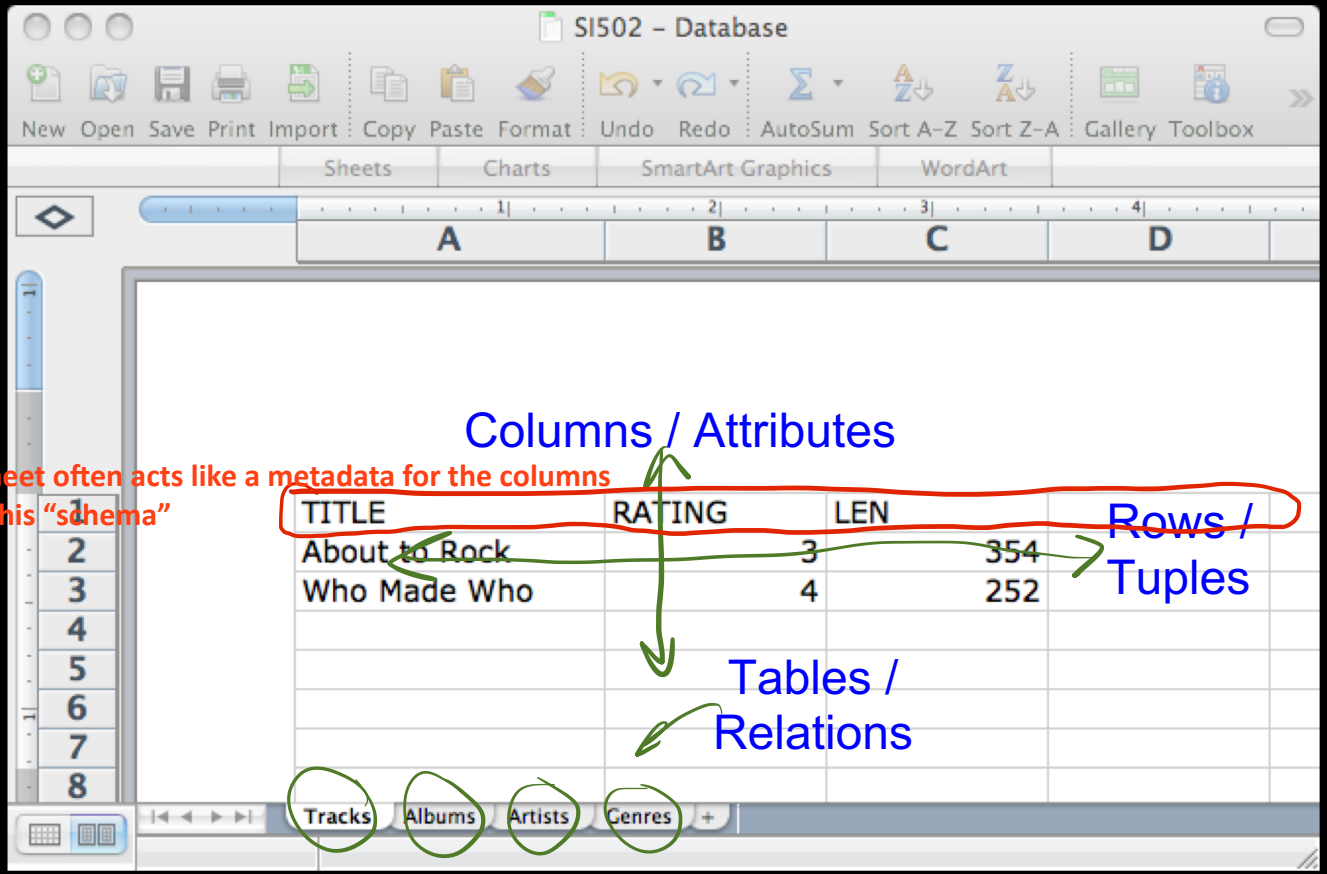
these terms are coined from Math

- Database - contains many tables
- Relation (or table) - contains tuples and attributes
- Tuple (or row) - a set of fields that generally represents an “object” like a person or a music track
- Attribute (also column or field) - one of possibly many elements of data corresponding to the object represented by the row



A relation is defined as a set of tuples that have the same attributes. A tuple usually represents an object and information about that object. Objects are typically physical objects or concepts. A relation is usually described as a table, which is organized into rows and columns. All the data referenced by an attribute are in the same domain and conform to the same constraints. (Wikipedia)

>1st row of a spreadsheet often acts like a metadata for the columns
>in database, we call this “schema”



Data Analysis

Brandon Krakowsky

Penn Engineering

Data Analysis

Loading Data – pandas Module

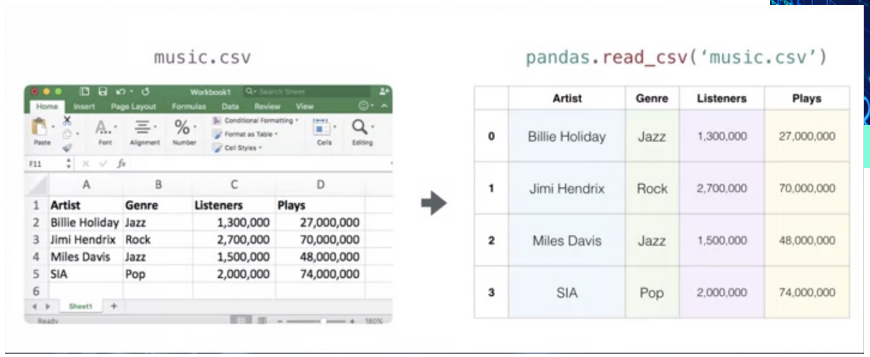
- The *pandas* module provides data analysis tools
- It can connect to and interact with a database
- It can also read and write Excel files
- It provides a useful *read_excel* method which reads an Excel file into a *DataFrame*

```
import pandas as pd
df = pd.read_excel('my_file.xlsx')
```

this method read the entire excel file into a DataFrame-object

```
#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

get and return a list of data sheet names
>use the ExcelFile class and the sheet_names attribute
>>> xl = pd.ExcelFile('foo.xls')
>>> xl.sheet_names

https://stackoverflow.com/questions/17977540/pandas-looking-up-the-list-of-sheets-in-an-excel-file

For reference:

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

.ExcelFile() vs .read_excel() method

https://stackoverflow.com/questions/26474693/excelfile-vs-read-excel-in-pandas

we will use this database for Teaching & Demo

Our Data – Yelp Dataset

It includes information about local businesses in 13 cities in Pennsylvania and Nevada.

- Information about local businesses in 13 cities in PA and NV
- Courtesy Yelp Dataset Challenge (https://www.yelp.com/dataset/)
- "yelp_data" tab data columns:

  - name: Name of business
  - category_0: 1st user-assigned business category
  - category_1: 2nd user-assigned business category

  In our data, these are the top two level business categories.

  - 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

	A	B	C	D	E	F	G	H
1	name	category_0	category_1	take_out	review_count	stars	city_id	state_id
2	China Sea Chinese Restaurant	Restaurants	Chinese	TRUE	11	2.5	1	1
3	Discount Tire Center	Tires	Automotive	FALSE	24	4.5	1	1
4	Frankfurters	Restaurants	Hot Dogs	TRUE	3	4.5	1	1
5	Fred Dietz Floral	Shopping	Flowers & Gifts	FALSE	6	4	1	1
6	Kuhn's Market	Food	Grocery	FALSE	8	3.5	1	1
7	Lincoln Bakery	Food	Bakeries	TRUE	25	4	1	1
8	Luigi's Pizzeria	Restaurants	Pizza	TRUE	18	4	1	1
9	Mane Attractions Unlimited	Hair Salons	Beauty & Spas	FALSE	4	3	1	1
10	R & B's Pizza Place	Restaurants	Pizza	TRUE	17	4	1	1
11	Rusty Nail	Restaurants	American (Traditional)	TRUE	32	3.5	1	1
12	Star Nails	Beauty & Spas	Nail Salons	FALSE	7	2.5	1	1
13	Vivo	Restaurants	Italian	FALSE	3	5	1	1
14	Emil's Lounge	Bars	American (New)	TRUE	26	4.5	2	1
15	Grand View Golf Club	Active Life	Golf	FALSE	3	5	2	1
16	Advance Auto Parts	Automotive	Auto Parts & Supplies	FALSE	3	3.5	3	1
17	Alexion's Bar & Grill	Bars	American (Traditional)	TRUE	23	4	3	1
18	Alteration World	Local Services	Sewing & Alterations	FALSE	10	4.5	3	1
19	Amerifit	Active Life	Boxing	FALSE	8	3	3	1
20	Barb's Country Junction Cafe	Restaurants	Cafes	TRUE	9	4	3	1
21	Carnegie Free Library	Public Services & Government	Libraries	FALSE	4	4.5	3	1
22	Don Don Chinese Restaurant	Restaurants	Chinese	TRUE	10	2.5	3	1
23	Extended Stay America - Pittsburgh - Carnegie	Hotels & Travel	Event Planning & Services	FALSE	11	3.5	3	1
24	Flynn's Tire & Auto Service	Auto Repair	Automotive	FALSE	9	2.5	3	1
25	Forsythe Miniature Golf & Snacks	Active Life	Mini Golf	FALSE	4	4	3	1
26	Gab & Eat	Breakfast & Brunch	Sandwiches	TRUE	69	4.5	3	1
27	Heidelberg B P	Automotive	Gas & Service Stations	FALSE	4	3	3	1
28	Kings Family Restaurant	Burgers	Breakfast & Brunch	TRUE	10	3.5	3	1
29	Knorr's Sunoco Service	Automotive	Gas & Service Stations	FALSE	3	3.5	3	1
30	Paddy's Pour House	Pubs	Irish	TRUE	8	3.5	3	1
31	Papa J's	Restaurants	Italian	TRUE	81	3.5	3	1
32	Porto Fino Pizzaria & Gyro	Restaurants	Pizza	FALSE	4	2.5	3	1
33	Quaker State Construction	Home Services	Roofing	FALSE	3	2.5	3	1

yelp_data | cities | states | +

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
```

.parse() method
>usage: parse an individual worksheet/tab inside an excel file
>i.e. parse the "yelp_data" worksheet/tab and return a DataFrame

- *df* is a DataFrame

```
type(df)
```

- Get a count of rows

```
len(df)
```

- Get the size (rows, columns)

```
df.shape
```

get the size (rows and columns of the data table) by accessing the "shape" attribute

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()` **>display the 1st 5 rows of the data table**
- Or the first 100 rows of data
  `df.head(100)`
- Drop the duplicates (based on all columns) from df
  `df = df.drop_duplicates()`

.head() method
>syntax: DataFrame_variable.head(n=5)
>called by a DataFrame object (or a variable referring to it)
parameter
>`n`: an integer for numbers of rows to select, Default = 5.
> If provide an argument, ex. 100, it will display the given number of rows. This displays the first 100 rows of data.
Return
>get and returns the first n rows for the caller object, the returned object is the same type as the called
>For negative values of n, this function returns all rows except the last n rows, equivalent to df[:-n].
Usage:
>It is useful for quickly testing if your object has the right type of data in it.

>By default, this will look at all columns to identify and drop duplicate rows in the data.
>to drop duplicates based on a specific column , use the `subset=` parameter to specify the column name you want to use to compare and drop duplicates based on

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop_duplicates.html#pandas.DataFrame.drop_duplicates

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

Querying Data

查询

- Select just the business names using the “name” of the attribute in between square brackets []

```
df[”name"] #returns name for every record
```

the “name” column

select all the data in the “name” column

- 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)
```

column

column

To select some columns of data in DataFrame “df”

get/select and return specific columns of a Dataframe that you want
>Syntax: DataFrame_variable[["column1, column2, column3..."]]

  - This only shows the id for each city and state
  - How can we get the actual values?

>To select specific columns of data, specify the names of the columns inside of square brackets.
>To select just the names of the businesses, use the “name” attribute.
>To query the location for the first 100 businesses:
1) we'll create a list of column names,
2) then put that list inside of square brackets and
3) get the first 100 records by calling the head method with an argument of 100.
This will only show the ID for each city and state but we want to see the actual cities and states.
So, how can we get the actual values? We need to join this data to other data sets.

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”

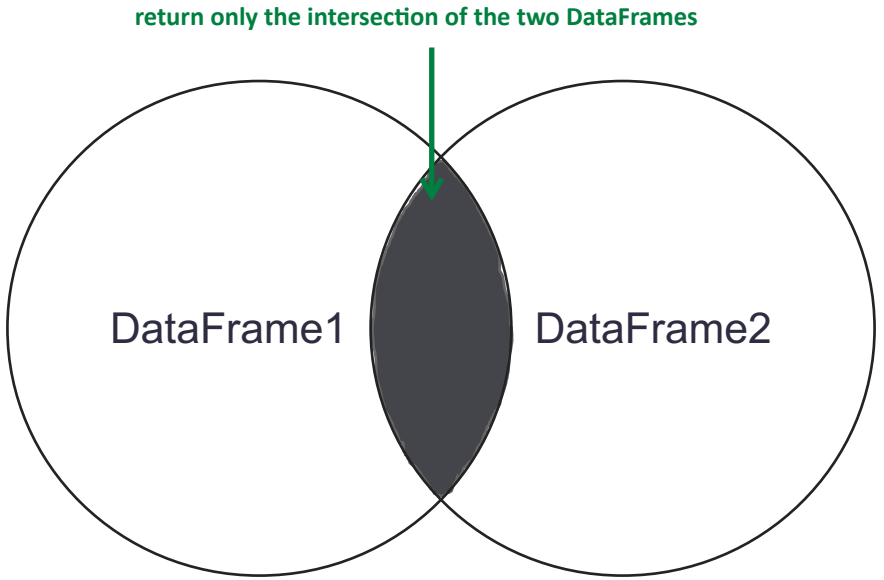
id	city
1	Bellevue
7	Munhall
8	Pittsburgh
9	West Homestead
10	West Mifflin
11	Henderson
12	Las Vegas

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

see Jupyter

- 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()
  ```

.merge() method

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.merge.html

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*]
  - inclusive / exclusive
  - *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)
    (up-to not including)

Querying Data - Slicing Rows

see Jupyter

- 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

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html#pandas.Series

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
```

`|` vertical bar means “OR”
>if condition 1 is True or condition 2 is True
“&” means “AND”
>if condition 1 is True and condition 2 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

see Jupyter

- 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()*

see Jupyter

- 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
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.contains.html?highlight=str%20contain

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*? https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html
  `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`

.value_counts() method
syntax: `Series`.value_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True)
>Return a Series-object containing counts of unique values.
>The resulting object will be in descending order so that the first element is the most frequently-occurring element

Other useful methods.
>What are the unique cities in our data set? This returns the unique values in the city column.
>How many businesses are there in each city? This counts the records for each city.
>How many unique user assigned business categories are there in category 0? This 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")]
```

You can easily create new columns in a data frame by just naming them uniquely and using them.
For example, some new column in my df equals some value. Add a new categories column that combines category 0 and category 1 as a comma separated list. This concatenates the string value of category 0 with category 1 and separates them by a comma. Now we can look up businesses based on a single categories column. This looks for pizza restaurants by checking if the categories column contains the string Pizza.

Updating & Creating Data - Exercise

see Jupyter

- 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()
```

Quiz 11 - Loading, Querying, Joining, & Filtering Data

Q1. What's the result of the following code?
```
import pandas as pd
xls = pd.ExcelFile('yelp.xlsx')
df = xls.parse('yelp_data')
print(type(df))
```
Ans: DataFrame

Q2. How can you quickly examine the first 5 rows of a DataFrame?
Ans: df.head()

Q3. How can you get the type of data stored in each column of a DataFrame df?
Ans: df.dtypes

Q4. A DataFrame df has exactly the data shown as below
How many rows will there be after calling df.drop_duplicates("AUTHOR", inplace = True)?
Ans: 1

	AUTHOR	TITLE	DATE
0	AACHEN, Hans von	Venus and Adonis	1574-88
1	AACHEN, Hans von	Allegory	1598
2	AACHEN, Hans von	Allegory of Peace, Art and Abundance	1602
3	AACHEN, Hans von	Jupiter, Antiope and Cupid	1595-98
4	AACHEN, Hans von	Pallas Athena, Venus and Juno	1593

Q5. Which is the correct way to select some columns of data in DataFrame df?
Ans: df[["id", "name", "gender"]]

Q6. Which method do we use for joining two tables?
Ans: pd.merge(left=df, right=df_cities, how='inner', left_on='city_id', right_on='id')

Q7. What does the following code do?
```
df[len(df) – 1 : ]
```
Ans: Returns the last row

Q8. What is the result of the following code?
```
s = df["some_attribute"] == "some_value"
type(s)
```
Ans: Series

Q9. What does the following code do if "df" is a DataFrame that stores the movie information?
```
genre = df["genre"].isin(["Comedy", "Action"])
year = df["release_year"] == 2000
rating = df["rating"]  > 4
df[genre & (year | rating)]
```
Ans: Select the comedy movies and action movies that are released in 2000 or have ratings above 4.

Q10. Which one is correct if we want to get the average rating for action movies?
Ans: df[df["genre"] == "Action"]["rating"].mean()

Querying Data - Summarizing Groups

+Jupyter

- *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()
```

get the keys for all the k/v pairs => i.e. city names

- 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

The groupby method splits data into different groups based on variables 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 their corresponding values. For example, when we group by city, the keys will be all possible cities. We can use the groups attribute to get a specific group of records. How many businesses are there in Las Vegas? But this kind of info in itself isn't actually all that useful. What we really want to be able to do is to perform aggregate computations on the data within each group.

see next slide

HW05-Q16 requires this section

Querying Data – agg()

+Jupyter

- 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/)

NumPy is a fundamental package for scientific computing in Python

- 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"]
```

> The egg method allows us to do that, to perform aggregate computations on group data.
>You can pass a list of aggregate functions as arguments. To do this, you can use methods from the Numpy module, a popular package for scientific computing.
Let's get the sum, mean and standard deviation for the star ratings of each city.
>First, we import Numpy.
>Then we group our data on city and call the agg method. As arguments, we pass Numpy some function, mean function and standard deviation function.
>Finally, we display the result of applying those aggregate computations to the star's attribute.

use `.to_frame() method to convert a Series-object to DataFrame-object

Pivot Tables

+Jupyter

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

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

Pivot Tables – Using index

+Jupyter

- 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

> Let's create a pivot table and use city as the index. By default, the pivot table calculates the average or mean for every column in the result. The type of pivot_city is still a DataFrame.

Pivot Tables – Using index

+Jupyter

- 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”

> It's also possible to use more than one index called indices. The pivot table will sort the data for you. Here we use indices, state and take_out. The cells display the average or mean values for each take_out value within each state.

Pivot Tables - Exercise

see Jupyter

- 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_table()
>create a spreadsheet-style pivot table as a DataFrame

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

Let's do an exercise with pivot tables. Create a pivot table that displays the average or mean review count and star rating for bars and restaurants in each city. Only use category 0 for simplicity. For some hints, filter for bars and restaurants then pivot on state, city, and category 0.

Let's make a data frame that only contains bars and restaurants. So from the data frame, let's look at the category 0 column and let's see if the value is in a list.
> The list has bars, restaurants. This checks to see if the value in the category 0 column is either bars or restaurants. Let's store that condition in a variable, bars_rest =, and then let's use that to filter the data frame. So from the data frame, there's our condition. Let's store the new data frame in a variable, df_bars_rest =. And then if I run that, I should have a new data frame with just bars and restaurants, and this is only based on the category 0 values.
Play video starting at :1:18 and follow transcript1:18
Now, let's create a pivot table and pivot along state, city, and category. So using our pandas module, let's call pivot table. Let's give it the data frame, df_bars_rest_ and then we're going to index on the following columns, state, city, and category_0. We'll store that pivot table in pivot_state_cat. If I run that, I have a data frame here, which is actually a pivot table. And I'm going to look at certain columns without a relevant within that pivot table. I'm going to look at review_count and stars. If I run that, I now have a pivot table. I can see for each state and city and category the average review count and the average stars rating.

Pivot Tables – aggfunc()

+Jupyter

- 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
```

To display summary statistics other than the mean, use the aggfunc parameter to specify the aggregation functions,
Play video starting at ::9 and follow transcript0:09
use the values parameter to specify the columns for aggfunc. Like before, use the aggregation methods from the NumPy package. In our dataset, how many reviews does each city have? Create a pivot_table in index, on state and city. For the values to aggregate, specify the review_count column, because that's the column we're going to get the sum of. For aggfunc, specify the aggregation method to apply, we'll use the sum method from the NumPy package.

Pivot Tables – aggfunc()

+Jupyter

- 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

It's possible to further segment the results by using the columns parameter.
Here again, we create a pivot_table in index, on state and city.
Then we specify the values to aggregate, review_count and the aggregation method to apply sum.
We also specify the columns parameter for further segmenting the results based on take_out.
We can see the number of review_counts per city, and then separated based on whether or not the business provides take_out.

Pivot Tables – aggfunc()

+Jupyter

- 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
```

We can also pass as an argument to aggfunc, in the form of a dictionary object containing differen
ggregate functions to perform on different values. You can have multiple keys and values. If we
vant to see the total number of review_counts and average ratings, we can specify it using this
ormat. Create a pivot_table passing a dictionary object with the name of a column as the key, anc
he aggregate method to apply to that column as the value. In this case, review_count is the key,
nd NumPy sum method is the value to apply to the review_count column. Stars is another key,
nd NumPy's mean method is the value to apply to the stars column

Visualization

Jupyter Notebook – “Magic Functions”

+Jupyter

- 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:

Jupyter Notebook has built-in magic functions that are helpful for handling meta or other functionalities in Python. The magic function %pylab inline allows the pylab library to load and let's our visualization to show up inside of a notebook. Run this code in its own cell.

```
%pylab inline
```

Visualization - matplotlib

+Jupyter

- *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

Matplotlib is a popular plotting library for Python. Matplotlib is included in the PyLab package. It's a very powerful tool that can plot a wide variety of figures and even animate them. We'll cover histograms, which represent the distribution of values in a dataset, and scatter plots, which display a set of data points or observations for two variables 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

+Jupyter

- 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
```

How do review patterns differ across different cities? Which city has a higher number of five-star reviews or one-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. First, we prepare the data and create the appropriate DataFrames. Then we plot the data, setting the various options for pyplot.

First, import pyplot

`pyplot`
>the plotting framework in matplotlib.
>this imports matplotlib.pyplot and creates an alias named "pd".

Visualization - Histogram

see Jupyter

- 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

see Jupyter

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

This creates two overlapping histograms in one figure.

histogram-object 1

```
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
```

opacity

set the number of equal-width bins in the range

Visualization - Histogram

see Jupyter

- Add another *histogram* for Las Vegas

```
# Same as before, but with red
```

histogram-object 2

```
plt.hist(series_vegas, alpha=0.3, color='red', label='Vegas’,
bins=‘auto’)
```

- Now we have two *histograms* inside our plot! Code on these two slides creates two overlapping histograms in one figure.
  - 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

see Jupyter

Other Configuration (graph setting)

- 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

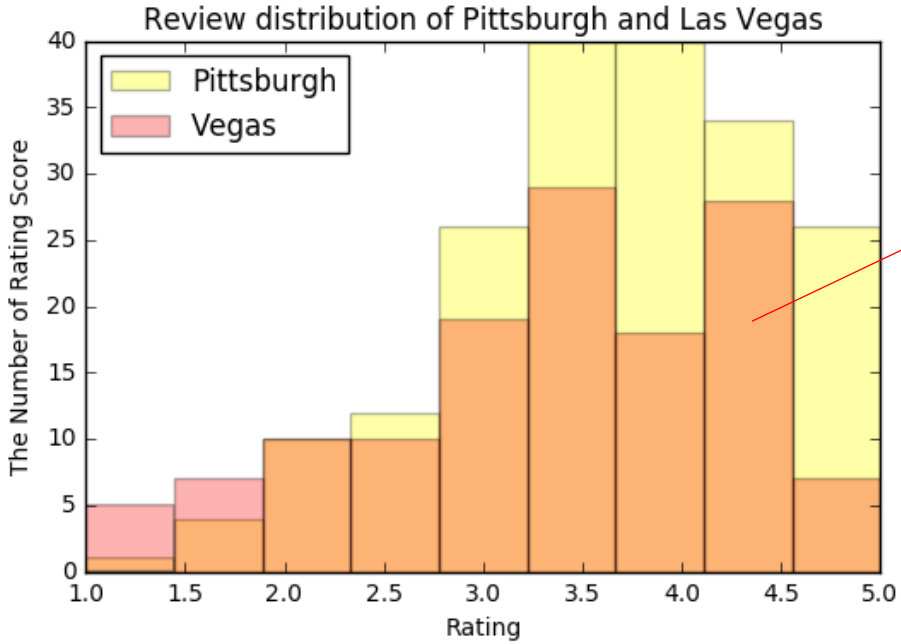
```
plt.title("Review distribution of Pittsburgh and Las Vegas")
```

Visualization - Histogram

see Jupyter

- 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

see Jupyter

- Having two plots overlap doesn't seem very appealing. Let's place the *histogram* bars side by side using *lists*

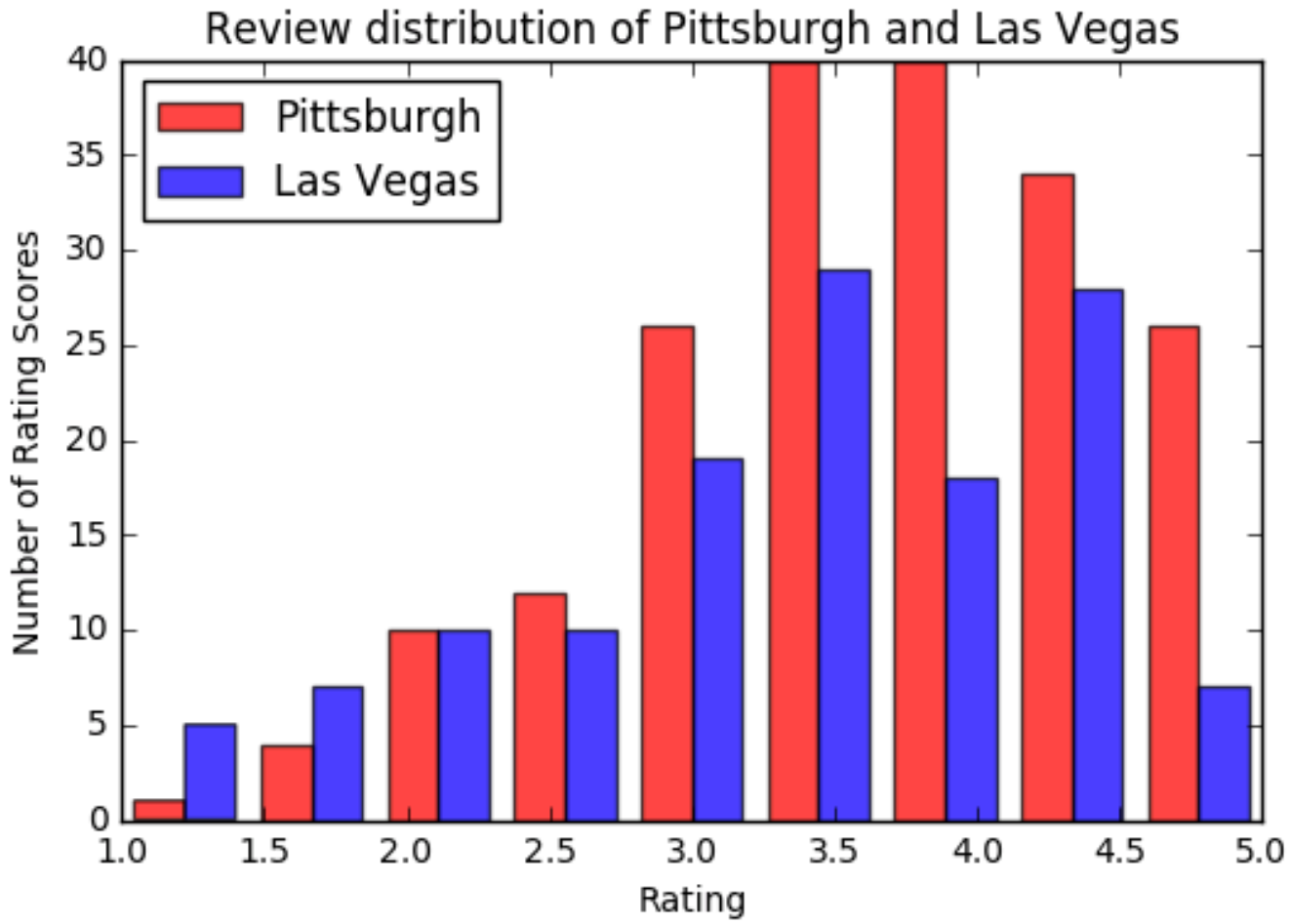
This plots two histograms whose bars are side by side in one figure.

- 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()
```

see Jupyter

Visualization - Histogram



Visualization - Scatterplot

+Jupyter

- 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']
```

Scatter plots display a set of data points or observations for two variables in an x-y plane. We can use a scatter plot 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 and medical, fast food, and breakfast and brunch. Let's start by creating new data frames for each category.
Note, we'll just use category_0 and keep things simple.

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
)
```

data for x-axis

data for y-axis

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

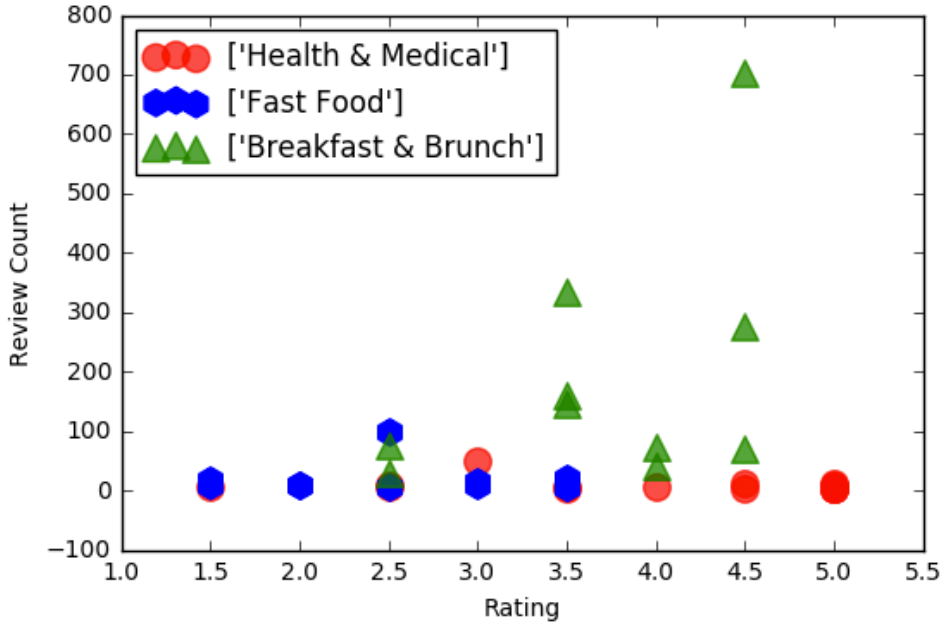
```
plt.xlabel('Rating’)

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

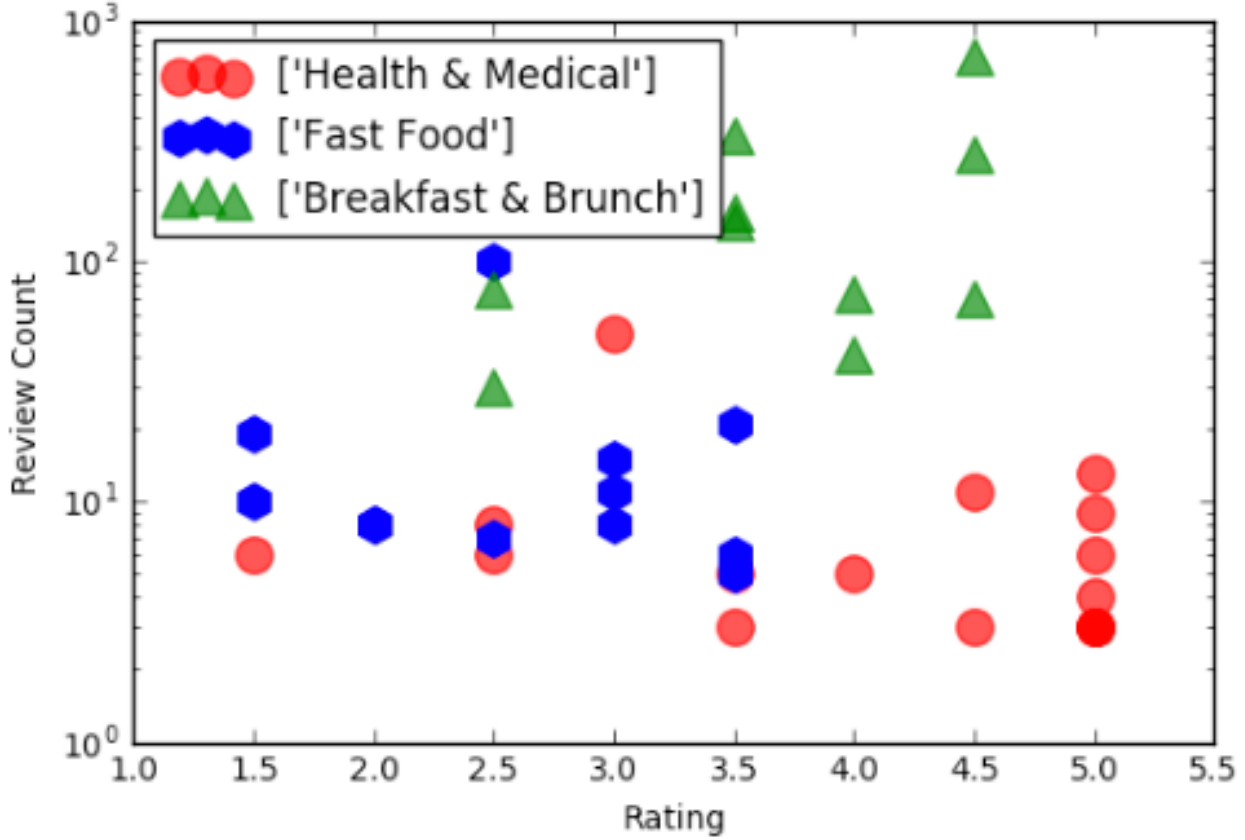
- The outliers in the y axis (review_count) are squashing the other data points
  - Let's make a little tweak to represent the y-axis in a *logarithmic* scale, so that review count is displayed in an *order of magnitude* (groups of 10)
  - All the values will be visualized in a more manageable range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

https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xscale.html#matplotlib.axes.Axes.set_xscale

Visualization - Scatterplot

- Looks much better!



Quiz 12 - Summarizing & Visualizing Data

Q1. What does df.groupby(['genre']) do?
Ans: It splits the data into different groups by 'genre'.

Q2. Is the following statement True or False?
We can pivot a DataFrame without an index column.
Ans: False

Q3. Is the following statement True or False?
We can pass a dict object as an argument to aggfunc.
Ans: True

Q4. When creating a pivot table, what would happen if we don't specify any aggregation function?
Ans: By default, the pivot table calculates the average for every column in the result.

Q5.Which one is correct if we want to create a pivot table for DataFrame df indexing on column "author" and "title", and display the summation of the "res" column?
Ans:
import pandas as pd
import numpy as np
df_res = pd.pivot_table(df, index=["author", "title"], values=["res"], aggfunc=[np.sum])

Q6. Assume you have imported the package pyplot as plt and created a histogram in the program. Now you want to display the histogram. What is the code you will write to show the plot?
plt.show()

Q7. Which method do we use to give an x-axis a label?
Ans: .xlable()

Q8. Assume we want to add a legend in a histogram. Which parameter should we specify if we want to place the legend at the center of the figure?
Ans: loc

Q9. What does the following script do?
plt.hist([series_pitt, series_vegas], alpha=0.7, color=['red','blue'], label=['Pittsburgh','Las Vegas'], bins="auto")
Ans: It plots two histograms whose bars are side by side in one figure.

Q10. What do the first two arguments specify in the method scatter()?
Ans: The data for the x and y axis

Q11. Is the following code the right way to set the scale of a y-axis as logarithmic?
import matplotlib.pyplot as plt
plt.set_yscale('log')
Ans: No

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/

after “querying data”, before “cleaning data”, before “coding practice 5” before “joining data”

Casting Data

practice_code_6 needs this section

HW05-Analyze Celebrity Death requires this section

HW05-Q8 requires this section

Casting Data with the *to_numeric()* Method

- In order to do numeric calculations (e.g. mean) on a DataFrame, the data type of the relevant columns need to be numeric
- Sometimes pandas will load data as a different data type, for example as an "object"
  - In this case, we need to cast the DataFrame to a numeric value
- To cast a DataFrame column to a numeric value, you can use the *to_numeric()* method
- For example, to cast the *salary* column in a DataFrame *df*

```
df['salary'] = pd.to_numeric(df['salary'], errors='coerce')
#The "errors" argument will catch (and ignore) any records where salary cannot be converted to a number
```

Casting Data with the *astype()* Method

- In order to work with raw text in a DataFrame, the relevant columns need to be strings
  - An example of this would be, checking if a word appears in a column, or replacing specific text in a column
- Depending on the data type of the column, we may need to cast to a string
- To do this, you can use the *astype(str)* method to cast the values in a column to a string
- For example, to cast the *description* column in a DataFrame *df* to a string

```
df['description'] = df['description'].astype(str)
```

after “casting data”, before “coding practice 5” before “joining data”

Cleaning Data & Dealing With Missing Values

practice_code_6 needs this section

Cleaning Data

- You can update or replace instances of text in a column
  - You'll need to make sure the relevant columns are strings
  - If necessary, cast to a string using the *astype()* method
- To replace text, use the *str.replace* method
- For example, to replace the $ character in the *description* column of a DataFrame *df*

```
df["description"] = df["description"].str.replace('$','')
```

parameter: (str_to_be_replaced, str_replace_to)

Dealing with Missing Values

HW05-Q15 requires this section

- Many datasets have missing values in particular columns
- Pandas provides multiple ways of dealing with this. The easiest way is to just drop the rows with the missing values

```
df.dropna(inplace = True)
#inplace means the changes are made in the DataFrame itself
```

- Another way is to fill-in the missing values using *fillna()*

```
df.fillna(0)
#the 0 means all NaN (Not a Number) elements will be replaced with 0s
```

after the data visualization section

Bar Charts & Plotting Pivot Tables

Bar Charts

- Bar charts are very similar to histograms which are used to represent the distribution of values in data
- While a histogram typically represent the *frequency distribution of continuous variables*, a bar chart is a *comparison of discrete variables*
- Another way to think about it is, a histogram *presents numerical data* and a bar chart *shows categorical data*

Creating a Bar Chart Using the *bar* Method

- One way to create a bar chart is by using matplotlib's *bar* method
- This is similar to the *hist* method, where you provide the data to plot and some additional parameter configurations

```
import pandas as pd
import matplotlib.pyplot as plt

plt.bar(
    data_for_x_axis,
    data_for_y_axis,
    #other optional parameters ...
)
```

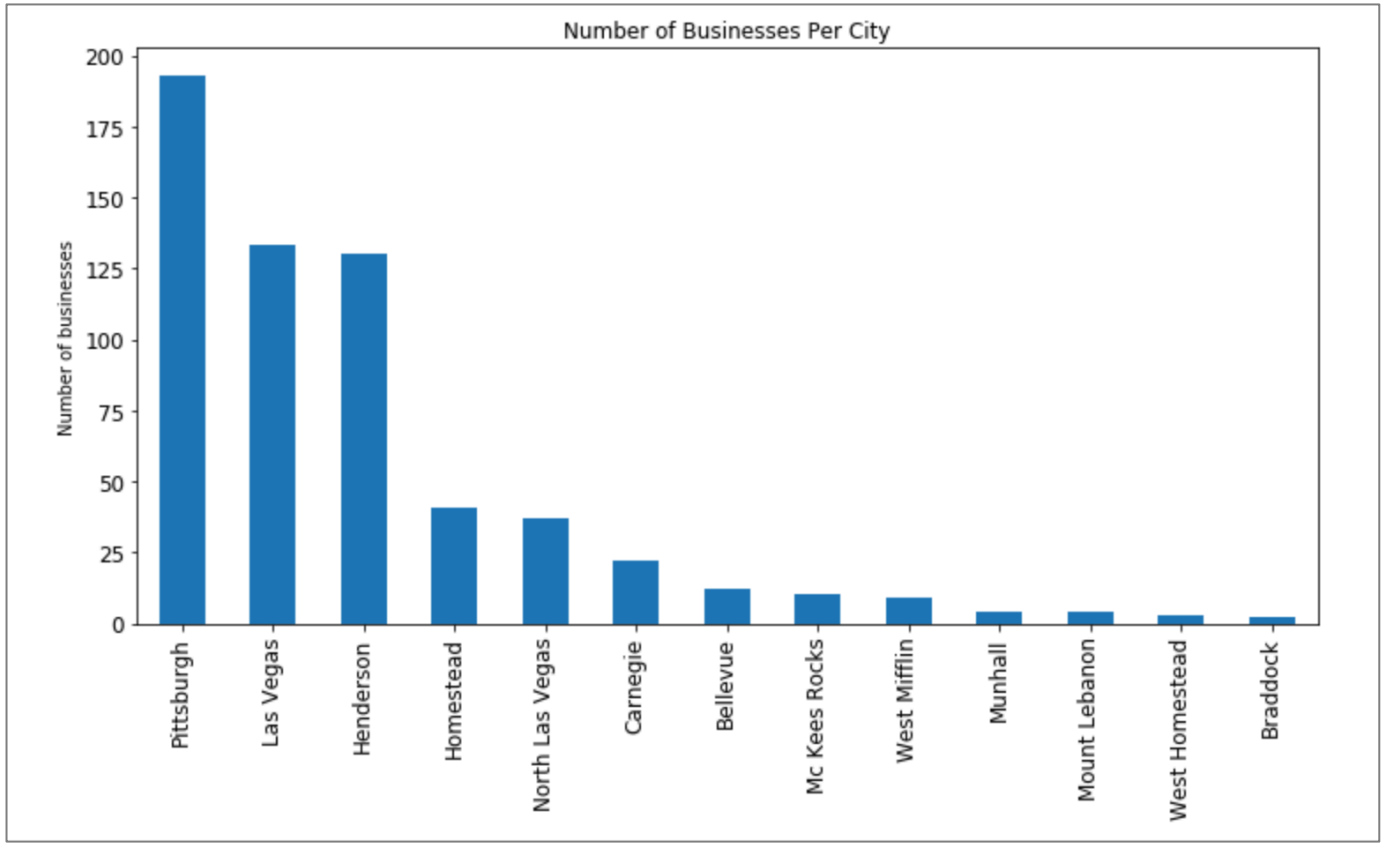
Creating a Bar Chart Using the *plot* Method

- Another *very easy* way to create a bar chart, is by using a DataFrame's built-in *plot* method
- Remember when we were working with the Yelp data, and we asked the question "How many businesses are there in each *city*?" Here's an example where we visualize that data:

```
#count the records for each city and get a new DataFrame
df_city_value_counts = df['city'].value_counts()

#call the plot method and set the kind parameter to 'bar'
df_city_value_counts.plot(kind='bar', figsize=(12, 6), fontsize=12,
   legend=False, title="Number of Businesses Per City")

plt.ylabel("Number of businesses")
plt.show()
```

Creating a Bar Chart Using the *plot* Method

- Another *very easy* way to create a bar chart, is by using a DataFrame's built-in *plot* method
- Remember when we were working with the Yelp data, and we asked the question "How many businesses are there in each *city*?" Here's an example where we visualize that data:



Plotting a Pivot Table

- Here's a visualization of a pivot table that displays the average (mean) star rating for bars and restaurants:

```
bar_rest = df["category_0"].isin(["Bars", "Restaurants"])
df_bar_rest = df[bar_rest]

#pivot along category
pivot_state_cat = pd.pivot_table(df_bar_rest, index=["category_0"])

#filter the df_bar_rest DataFrame columns
pivot_state_cat = pivot_state_cat[["stars"]]

#call the plot method and set the kind parameter to 'bar'
pivot_state_cat.plot(kind='bar', figsize=(12, 6), fontsize=12,
    legend=False, title="Average Star Rating for Bars & Restaurants")

plt.xlabel("Category")
plt.ylabel("Average star rating")
plt.show()
```

Plotting a Pivot Table

- Here's a visualization of a pivot table that displays the average (mean) star rating for bars and restaurants: