Project: Yelp Rating Regression Predictor

The restaurant industry is tougher than ever, with restaurant reviews blazing across the Internet from day one of a restaurant's opening. But as a lover of food, you and your friend decide to break into the industry and open up your own restaurant, Danielle's Delicious Delicacies. Since a restaurant's success is highly correlated with its reputation, you want to make sure Danielle's Delicious Delicacies has the best reviews on the most queried restaurant review site: Yelp! While you know your food will be delicious, you think there are other factors that play into a Yelp rating and will ultimately determine your business's success. With a dataset of different restaurant features and their Yelp ratings, you decide to use a Multiple Linear Regression model to investigate what factors most affect a restaurant's Yelp rating and predict the Yelp rating for your restaurant!

In this project we'll be working with a real dataset provided by Yelp. We have provided six files, listed below with a brief description:

- `yelp_business.json`: establishment data regarding location and attributes for all businesses in the dataset
- `yelp_review.json`: Yelp review metadata by business
- `yelp_user.json`: user profile metadata by business
- `yelp_checkin.json`: online checkin metadata by business
- `yelp_tip.json`: tip metadata by business
- `yelp_photo.json`: photo metadata by business

For a more detailed explanation of the features in each `.json` file, see the accompanying explanatory feature document (https://docs.google.com/document/d/1V6FjJpKspVBOOBs4E7fBfp_yzHn0--XJkC2uUtWuRgM/edit).

Let's get started by exploring the data in each of these files to see what we are working with.

Load the Data and Take a Peek

To get a better understanding of the dataset we can use Pandas to explore the data in DataFrame form. In the code block below we have imported Pandas for you. The `read_json()` method reads data from a json file into a DataFrame, as shown below:

```
df = pd.read_json('file_name.json', lines=True)
```

Load the data from each of the json files with the following naming conventions:

- `yelp_business.json` into a DataFrame named `businesses`
- `yelp_review.json` into a DataFrame named `reviews`
- `yelp_user.json` into a DataFrame named `users`
- `yelp_checkin.json` into a DataFrame named `checkins`
- `yelp_tip.json` into a DataFrame named `tips`
- `yelp_photo.json` into a DataFrame named `photos`

Importing that data could take 10 to 20 seconds to run depending on your computer, but don't worry, once it's loaded in you're ready to go!

In [1]:
```
import pandas as pd
businesses = pd.read_json('yelp_business.json', lines=True)
reviews = pd.read_json('yelp_review.json', lines=True)
users = pd.read_json('yelp_user.json', lines=True)
checkins = pd.read_json('yelp_checkin.json', lines=True)
tips = pd.read_json('yelp_tip.json', lines=True)
photos = pd.read_json('yelp_photo.json', lines=True)
```

In order to more clearly see the information in our DataFrame, we can adjust the number of columns shown (`max_columns`) and the number of characters shown in a column (`max_colwidth`) with the below code:

```
pd.options.display.max_columns = number_of_columns_to_display
pd.options.display.max_colwidth = number_of_characters_to_display
```

Set `max_columns` to `60` and `max_colwidth` to `500`. We are working with some BIG data here!

In [3]:
```
pd.options.display.max_columns = 60
pd.options.display.max_colwidth = 500
```

Inspect the first five rows of each DataFrame using the `.head()` method to get an overview of the data (make sure to check each DataFrame in a separate cell in order to view it properly).

In [4]:
```
businesses.head(5)
```

Out[4]:

	address	alcohol?	attributes	business_id	categories	c
0	1314 44 Avenue NE	0	{'BikeParking': 'False', 'BusinessAcceptsCreditCards': 'True', 'BusinessParking': '{'garage': False, 'street': True, 'validated': False, 'lot': False, 'valet': False}', 'GoodForKids': 'True', 'HasTV': 'True', 'NoiseLevel': 'average', 'OutdoorSeating': 'False', 'RestaurantsAttire': 'casual', 'RestaurantsDelivery': 'False', 'RestaurantsGoodForGroups': 'True', 'RestaurantsPriceRange2': '2', 'RestaurantsReservations': 'True', 'RestaurantsTakeOut': 'True'}	Apn5Q_b6Nz61Tq4XzPdf9A	Tours, Breweries, Pizza, Restaurants, Food, Hotels & Travel	Calgary
1		0	{'Alcohol': 'none', 'BikeParking': 'False', 'BusinessAcceptsCreditCards': 'True', 'BusinessParking': '{'garage': False, 'street': True, 'validated': False, 'lot': True, 'valet': False}', 'Caters': 'True', 'DogsAllowed': 'True', 'DriveThru': 'False', 'GoodForKids': 'True', 'GoodForMeal': '{'dessert': False, 'latenight': False, 'lunch': False, 'dinner': False, 'breakfast': False, 'brunch': False}', 'HasTV': 'False', 'OutdoorSeating': 'True', 'RestaurantsAttire': 'casual', 'RestaurantsDelivery'...	AjEbIBw6ZFfln7ePHha9PA	Chicken Wings, Burgers, Caterers, Street Vendors, Barbeque, Food Trucks, Food, Restaurants, Event Planning & Services	Henders
2	1335 rue Beaubien E	1	{'Alcohol': 'beer_and_wine', 'Ambience': '{'romantic': False, 'intimate': False, 'classy': False, 'hipster': False, 'touristy': False, 'trendy': False, 'upscale': False, 'casual': False}', 'BikeParking': 'True', 'BusinessAcceptsCreditCards': 'False', 'BusinessParking': '{'garage': False, 'street': False, 'validated': False, 'lot': False, 'valet': False}', 'Caters': 'False', 'GoodForKids': 'True', 'GoodForMeal': '{'dessert': False, 'latenight': False, 'lunch': False, 'dinner': False,	O8S5hYJ1SMc8fA4QBtVujA	Breakfast & Brunch, Restaurants, French, Sandwiches, Cafes	Montréa

In [5]:
```
reviews.head(5)
```

Out[5]:

	average_review_age	average_review_length	average_review_sentiment	business_
0	524.458333	466.208333	0.808638	--1UhMGODdWsrMastO9D
1	1199.589744	785.205128	0.669126	--6MefnULPED_I942VcFNA
2	717.851852	536.592593	0.820837	--7zmmkVg-IMGaXbuVd0S
3	751.750000	478.250000	0.170925	--8LPVSo5i0Oo61X01sV9A
4	978.727273	436.181818	0.562264	--9QQLMTbFzLJ_oT-ON3X

In [6]:
```
users.head(5)
```

Out[6]:

	average_days_on_yelp	average_number_fans	average_number_friends	average_number_years_e
0	1789.750000	1.833333	18.791667	0.833333
1	2039.948718	49.256410	214.564103	1.769231
2	1992.796296	19.222222	126.185185	1.814815
3	2095.750000	0.500000	25.250000	0.000000
4	1804.636364	1.000000	52.454545	0.090909

In [7]:
```
checkins.head(5)
```

Out[7]:

	business_id	time	weekday_checkins	weekend_checkins
0	7KPBkxAOEtb3QeIL9PEErg	{'Fri-0': 2, 'Sat-0': 1, 'Sun-0': 1, 'Wed-0': 2, 'Fri-1': 1, 'Sat-1': 3, 'Thu-1': 1, 'Wed-1': 1, 'Sat-2': 1, 'Sun-2': 2, 'Thu-2': 1, 'Wed-2': 1, 'Fri-3': 1, 'Sun-3': 3, 'Mon-4': 1, 'Thu-4': 1, 'Tue-4': 2, 'Wed-4': 2, 'Sun-6': 1, 'Wed-6': 1, 'Thu-7': 1, 'Fri-10': 3, 'Mon-10': 1, 'Sat-10': 3, 'Sun-10': 3, 'Tue-10': 2, 'Mon-11': 1, 'Thu-11': 1, 'Wed-11': 2, 'Mon-12': 1, 'Sat-12': 1, 'Tue-12': 1, 'Sat-13': 3, 'Thu-13': 1, 'Tue-13': 2, 'Wed-13': 3, 'Fri-14': 2, 'Mon-14': 1, 'Sat-14': 1, 'Sun-14':...	76	75
1	kREVIrSBbtqBhIYkTccQUg	{'Mon-13': 1, 'Thu-13': 1, 'Sat-16': 1, 'Wed-17': 1, 'Sun-19': 1, 'Thu-20': 1, 'Sat-21': 1}	4	3
2	tJRDll5yqpZwehenzE2cSg	{'Thu-0': 1, 'Mon-1': 1, 'Mon-12': 1, 'Sat-16': 1, 'Sun-22': 1, 'Fri-23': 1}	3	3
3	tZccfdl6JNw-j5BKnCTIQQ	{'Sun-14': 1, 'Fri-18': 1, 'Mon-20': 1}	1	2
4	r1p7RAMzCV_6NPF0dNoR3g	{'Sat-3': 1, 'Sun-18': 1, 'Sat-21': 1, 'Sat-23': 1, 'Thu-23': 1}	1	4

In [8]:
```
tips.head(5)
```

Out[8]:

	average_tip_length	business_id	number_tips
0	79.000000	--1UhMGODdWsrMastO9DZw	1
1	49.857143	--6MefnULPED_I942VcFNA	14
2	52.500000	--7zmmkVg-IMGaXbuVd0SQ	10
3	136.500000	--9QQLMTbFzLJ_oT-ON3Xw	2
4	68.064935	--9e1ONYQuAa-CB_Rrw7Tw	154

In [9]:
```
photos.head(5)
```

Out[9]:

	average_caption_length	business_id	number_pics
0	0.000000	--1UhMGODdWsrMastO9DZw	1
1	67.500000	--6MefnULPED_I942VcFNA	2
2	30.426471	--9e1ONYQuAa-CB_Rrw7Tw	136
3	0.000000	--DaPTJW3-tB1vP-PfdTEg	1
4	5.500000	--FBCX-N37CMYDfs790Bnw	4

How many different businesses are in the dataset? What are the different features in the review DataFrame?

In [17]:
```
print(businesses.business_id.nunique())
businesses.info()  # range index value matches uunique value above = 188,593 busine
sses in the dataset
reviews.info()  # 7 features or columns present in the dataset

# below also works from solution
print(len(businesses))
print(reviews.columns)
```

```
188593
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188593 entries, 0 to 188592
Data columns (total 22 columns):
address                 188593 non-null object
alcohol?                188593 non-null int64
attributes              162807 non-null object
business_id             188593 non-null object
categories              188052 non-null object
city                    188593 non-null object
good_for_kids           188593 non-null int64
has_bike_parking        188593 non-null int64
has_wifi                188593 non-null int64
hours                   143791 non-null object
is_open                 188593 non-null int64
latitude                188587 non-null float64
longitude               188587 non-null float64
name                    188593 non-null object
neighborhood            188593 non-null object
postal_code             188593 non-null object
price_range             188593 non-null int64
review_count            188593 non-null int64
stars                   188593 non-null float64
state                   188593 non-null object
take_reservations       188593 non-null int64
takes_credit_cards      188593 non-null int64
dtypes: float64(3), int64(9), object(10)
memory usage: 24.5+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188593 entries, 0 to 188592
Data columns (total 7 columns):
average_review_age          188593 non-null float64
average_review_length       188593 non-null float64
average_review_sentiment    188593 non-null float64
business_id                 188593 non-null object
number_cool_votes           188593 non-null int64
number_funny_votes          188593 non-null int64
number_useful_votes         188593 non-null int64
dtypes: float64(3), int64(3), object(1)
memory usage: 9.4+ MB
188593
Index(['average_review_age', 'average_review_length',
       'average_review_sentiment', 'business_id', 'number_cool_votes',
       'number_funny_votes', 'number_useful_votes'],
      dtype='object')
```

What is the range of values for the features in the user DataFrame?

In [18]:
```
users.describe()
```

Out[18]:

	average_days_on_yelp	average_number_fans	average_number_friends	average_number_yea
count	188593.000000	188593.000000	188593.000000	188593.000000
mean	2005.367009	11.590148	105.132000	0.923313
std	554.174540	25.901801	162.653680	1.109289
min	76.000000	0.000000	1.000000	0.000000
25%	1647.000000	0.666667	26.666667	0.000000
50%	1957.150000	3.583333	59.384615	0.583333
75%	2312.238095	11.555556	117.666667	1.400000
max	4860.000000	1174.666667	4219.000000	10.666667

What is the Yelp rating, or `stars`, of the establishment with `business_id` = `5EvUIR4IzCWUOm0PsUZXjA`. Use Pandas boolean indexing to find the Yelp rating, using the syntax below:

```
df[df['column_we_know'] == 'value_we_know']['column_we_want']
```

In [19]:
```
businesses[businesses['business_id'] == '5EvUIR4IzCWUOm0PsUZXjA']['stars']
```

Out[19]:
```
30781    3.0
Name: stars, dtype: float64
```

What feature, or column, do the DataFrames have in common?

- my answer: `business_id` is a common key

Merge the Data

Since we are working with data from several files, we need to combine the data into a single DataFrame that allows us to analyze the different features with respect to our target variable, the Yelp rating. We can do this by merging the multiple DataFrames we have together, joining them on the columns they have in common. In our case, this unique identifying column is the `business_id`. We can merge two DataFrames together with the following syntax:

```
pd.merge(left, right, how='inner/outer/left/right', on='column(s)_to_merge_on')
```

- `left` is the DataFrame on the left side of our merge
- `right` is the DataFrame on the right side of our merge
- `how` describes the style of merge we want to complete (similar to inner/outer/left/right joins in SQL)
- `on` is the column or columns to perform the merge on (the column connecting the two tables)

Given our six DataFrames, we will need to perform 5 merges to combine all the data into one DataFrame. In the cell below we merged the business table and the review table into a new DataFrame, `df`, for you. After the merge we've added all the rows from `businesses` and `reviews` together, but kept the same total number of rows! Run the cell to perform the merge and confirm the number of rows in `df`.

In [25]:
```
df = pd.merge(businesses, reviews, how='left', on='business_id')
print(len(df))
```

```
188593
```

Merge each of the other 4 DataFrames into our new DataFrame `df` to combine all the data together. Make sure that `df` is the left DataFrame in each merge and `how=left` since not every DataFrame includes every business in the dataset (this way we won't lose any data during the merges). Once combined, print out the columns of `df`. What features are in this new DataFrame?

In [26]:
```
df = pd.merge(df, users, how='left', on='business_id')
df = pd.merge(df, checkins, how='left', on='business_id')
df = pd.merge(df, tips, how='left', on='business_id')
df = pd.merge(df, photos, how='left', on='business_id')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 188593 entries, 0 to 188592
Data columns (total 40 columns):
address                       188593 non-null object
alcohol?                      188593 non-null int64
attributes                    162807 non-null object
business_id                   188593 non-null object
categories                    188052 non-null object
city                          188593 non-null object
good_for_kids                 188593 non-null int64
has_bike_parking              188593 non-null int64
has_wifi                      188593 non-null int64
hours                         143791 non-null object
is_open                       188593 non-null int64
latitude                      188587 non-null float64
longitude                     188587 non-null float64
name                          188593 non-null object
neighborhood                  188593 non-null object
postal_code                   188593 non-null object
price_range                   188593 non-null int64
review_count                  188593 non-null int64
stars                         188593 non-null float64
state                         188593 non-null object
take_reservations             188593 non-null int64
takes_credit_cards            188593 non-null int64
average_review_age            188593 non-null float64
average_review_length         188593 non-null float64
average_review_sentiment      188593 non-null float64
number_cool_votes             188593 non-null int64
number_funny_votes            188593 non-null int64
number_useful_votes           188593 non-null int64
average_days_on_yelp          188593 non-null float64
average_number_fans           188593 non-null float64
average_number_friends        188593 non-null float64
average_number_years_elite    188593 non-null float64
average_review_count          188593 non-null float64
time                          157075 non-null object
weekday_checkins              157075 non-null float64
weekend_checkins              157075 non-null float64
average_tip_length            121526 non-null float64
number_tips                   121526 non-null float64
average_caption_length        32976 non-null float64
number_pics                   32976 non-null float64
dtypes: float64(17), int64(12), object(11)
memory usage: 51.1+ MB
```

Clean the Data

We are getting really close to the fun analysis part! We just have to clean our data a bit so we can focus on the features that might have predictive power for determining an establishment's Yelp rating.

In a Linear Regression model, our features will ideally be continuous variables that have an affect on our dependent variable, the Yelp rating. For this project with will also be working with some features that are binary, on the scale [0,1]. With this information, we can remove any columns in the dataset that are not continuous or binary, and that we do not want to make predictions on. The cell below contains a list of these unnecessary features. Drop them from `df` with Pandas' drop syntax, provided below:

```
df.drop(list_of_features_to_remove, axis=1, inplace=True)
```

- `list_of_features_to_remove` is, you guessed it, the list of features we want to remove!
- `axis=1` lets Pandas know we want to drop columns, not rows, from our DataFrame (axis=0 is used for computations along rows!)
- `inplace=True` lets us drop the columns right here in our DataFrame, instead of returning a new DataFrame that we could store in a new variable

In [27]:
```
features_to_remove = ['address','attributes','business_id','categories','city','hou
rs','is_open','latitude','longitude','name','neighborhood','postal_code','state','t
ime']
df.drop(features_to_remove, axis=1, inplace=True)
```

Now we just have to check our data to make sure we don't have any missing values, or `NaN`s, which will prevent the Linear Regression model from running correctly. To do this we can use the statement `df.isna().any()`. This will check all of our columns and return `True` if there are any missing values or `NaN`s, or `False` if there are no missing values. Check if `df` is missing any values.

In [28]:
```
df.isna().any()
```

Out[28]:
```
alcohol?                       False
good_for_kids                  False
has_bike_parking               False
has_wifi                       False
price_range                    False
review_count                   False
stars                          False
take_reservations              False
takes_credit_cards             False
average_review_age             False
average_review_length          False
average_review_sentiment       False
number_cool_votes              False
number_funny_votes             False
number_useful_votes            False
average_days_on_yelp           False
average_number_fans            False
average_number_friends         False
average_number_years_elite     False
average_review_count           False
weekday_checkins                True
weekend_checkins                True
average_tip_length              True
number_tips                     True
average_caption_length          True
number_pics                     True
dtype: bool
```

As you can see, there are a few columns with missing values. Since our dataset has no information recorded for some businesses in these columns, we will assume the Yelp pages did not display these features. For example, if there is a `NaN` value for `number_pics`, it means that the associated business did not have any pictures posted on its Yelp page. Thus we can replace all of our `NaN`s with `0`s. To do this we can use the `.fillna()` method, which takes a dictionary as shown below:

```
df.fillna({'column_1':val_to_replace_na,
           'column_2':val_to_replace_na,
           'column_3':val_to_replace_na},
          inplace=True)
```

- `column_1`, `column_2`, and `column_3` are the columns with missing values that we want to fill. We can include as many columns as we like in the dictionary that is passed to `.fill_na()`
- `val_to_replace_na` is the value that will replace the missing values, or `NaN`s
- `inplace=True` since we want to perform our changes in place and not return a new DataFrame

Fill the missing values in `df` with `0`. Afterwards, confirm the missing values have been filled with `df.isna().any()`.

In [29]:
```
df.fillna({'weekday_checkins':0, 'weekend_checkins':0, 'average_tip_length':0, 'num
ber_tips':0, 'average_caption_length':0, 'number_pics':0}, inplace=True)
df.isna().any()
```

Out[29]:
```
alcohol?                      False
good_for_kids                 False
has_bike_parking              False
has_wifi                      False
price_range                   False
review_count                  False
stars                         False
take_reservations             False
takes_credit_cards            False
average_review_age            False
average_review_length         False
average_review_sentiment      False
number_cool_votes             False
number_funny_votes            False
number_useful_votes           False
average_days_on_yelp          False
average_number_fans           False
average_number_friends        False
average_number_years_elite    False
average_review_count          False
weekday_checkins              False
weekend_checkins              False
average_tip_length            False
number_tips                   False
average_caption_length        False
number_pics                   False
dtype: bool
```

Exploratory Analysis

Now that our data is all together, let's investigate some of the different features to see what might correlate most with our dependent variable, the Yelp rating (called `stars` in our DataFrame). The features with the best correlations could prove to be the most helpful for our Linear Regression model! Pandas DataFrames have a really helpful method, `.corr()`, that allows us to see the correlation coefficients for each pair of our different features. Remember, a correlation of `0` indicates that two features have no linear relationship, a correlation coefficient of `1` indicates two features have a perfect positive linear relationship, and a correlation coefficient of `-1` indicates two features have a perfect negative linear relationship. Call `.corr()` on `df`. You'll see that `number_funny_votes` has a correlation coefficient of `0.001320` with respect to `stars`, our Yelp rating. This is a very weak correlation. What features best correlate, both positively and negatively, with Yelp rating?

In [30]:
```
df.corr()
```

Out[30]:

	alcohol?	good_for_kids	has_bike_parking	has_wifi	price_range	rev
alcohol?	1.000000	0.305284	0.213318	0.345032	0.349004	0.2
good_for_kids	0.305284	1.000000	0.271788	0.258887	0.205513	0.1
has_bike_parking	0.213318	0.271788	1.000000	0.235138	0.416044	0.1
has_wifi	0.345032	0.258887	0.235138	1.000000	0.240796	0.1
price_range	0.349004	0.205513	0.416044	0.240796	1.000000	0.1
review_count	0.259836	0.162469	0.155505	0.195737	0.148277	1.0
stars	-0.043332	-0.030382	0.068084	-0.039857	-0.052565	0.0
take_reservations	0.601670	0.318729	0.160129	0.312217	0.316105	0.1
takes_credit_cards	0.190738	0.150360	0.286298	0.155098	0.400742	0.1
average_review_age	0.139108	0.055847	-0.080443	-0.034258	0.189623	0.0
average_review_length	0.037369	-0.079183	-0.116295	-0.037712	0.003850	0.0
average_review_sentiment	0.097188	0.073806	0.130448	0.054699	0.089349	0.0
number_cool_votes	0.188598	0.113262	0.114094	0.147320	0.119422	0.8
number_funny_votes	0.117472	0.060658	0.060595	0.082213	0.073215	0.5
number_useful_votes	0.165775	0.083832	0.094000	0.120622	0.098990	0.7
average_days_on_yelp	0.129901	0.045057	-0.045849	0.000448	0.176133	0.0
average_number_fans	0.017794	0.024901	0.018120	0.023913	0.104221	0.0
average_number_friends	0.015261	0.016557	0.028307	0.015937	0.087231	0.0
average_number_years_elite	0.099141	0.094233	0.083062	0.082863	0.210487	0.0
average_review_count	0.026846	0.040692	0.031203	0.044006	0.122982	-0.
weekday_checkins	0.094398	0.068960	0.082474	0.107467	0.057877	0.5
weekend_checkins	0.131175	0.079808	0.093579	0.126861	0.081321	0.6
average_tip_length	0.098037	0.121948	0.144163	0.104742	0.129212	0.0
number_tips	0.208856	0.156536	0.147115	0.173542	0.119632	0.8
average_caption_length	0.305570	0.291413	0.180468	0.258938	0.170171	0.2
number_pics	0.252523	0.175058	0.109552	0.210583	0.143570	0.6

To further visualize these relationships, we can plot certain features against our dependent variable, the Yelp rating. In the cell below we have provided the code to import Matplotlib. We can use Matplotlib's `.scatter()` method with the below syntax to plot what these correlations look like:

```
plt.scatter(x_values_to_plot, y_values_to_plot, alpha=blending_val)
```

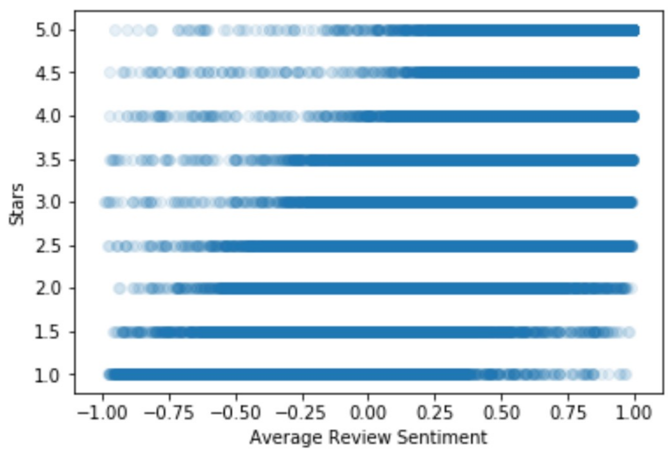
- `x_values_to_plot` are the values to be plotted along the x-axis
- `y_values_to_plot` are the values to be plotted along the y-axis
- `alpha=blending_val` is the blending value, or how transparent (0) or opaque (1) a plotted point is. This will help us distinguish areas of the plot with high point densities and low point densities

Plot the three features that correlate most with Yelp rating (`average_review_sentiment`, `average_review_length`, `average_review_age`) against `stars`, our Yelp rating. Then plot a lowly correlating feature, such as `number_funny_votes`, against `stars`.

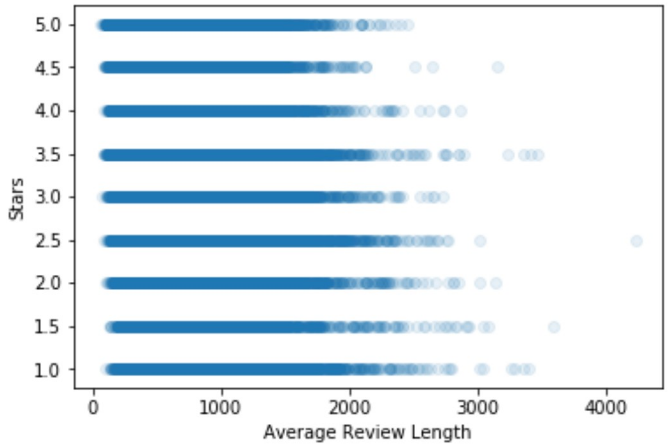
> What is `average_review_sentiment`, you ask? `average_review_sentiment` is the average sentiment score for all reviews on a business' Yelp page. The sentiment score for a review was calculated using the sentiment analysis tool [VADER (https://github.com/cjhutto/vaderSentiment)](https://github.com/cjhutto/vaderSentiment). VADER uses a labeled set of positive and negative words, along with codified rules of grammar, to estimate how positive or negative a statement is. Scores range from `-1`, most negative, to `+1`, most positive, with a score of `0` indicating a neutral statement. While not perfect, VADER does a good job at guessing the sentiment of text data!

What kind of relationships do you see from the plots? Do you think these variables are good or bad features for our Yelp rating prediction model?

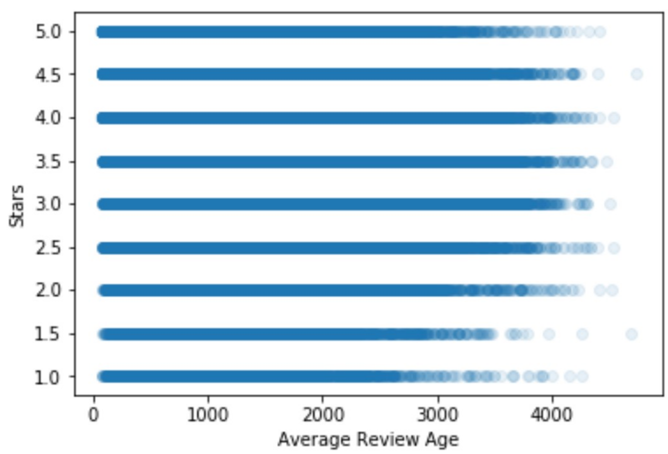
In [34]:
```
from matplotlib import pyplot as plt

# plot average_review_sentiment against stars here
plt.scatter(df['average_review_sentiment'],df['stars'],alpha=0.1)
plt.ylabel('Stars')
plt.xlabel('Average Review Sentiment')
plt.show()
```



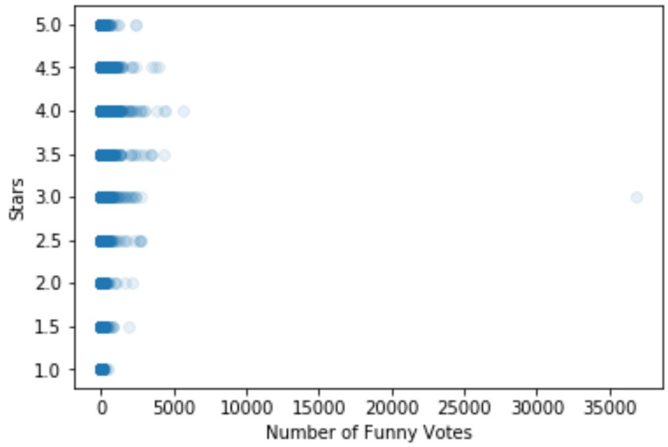
In [35]:
```
# plot average_review_length against stars here
plt.scatter(df['average_review_length'],df['stars'],alpha=0.1)
plt.ylabel('Stars')
plt.xlabel('Average Review Length')
plt.show()
```



In [36]:
```
# plot average_review_age against stars here
plt.scatter(df['average_review_age'],df['stars'],alpha=0.1)
plt.ylabel('Stars')
plt.xlabel('Average Review Age')
plt.show()
```



In [37]:
```
# plot number_funny_votes against stars here
plt.scatter(df['number_funny_votes'],df['stars'],alpha=0.1)
plt.ylabel('Stars')
plt.xlabel('Number of Funny Votes')
plt.show()
```



Why do you think `average_review_sentiment` correlates so well with Yelp rating?

my answer: It's already parsing written reviews to see if people think positively or negatively of the business. It really ought to be the primary indicator in predicting Yelp Stars.

Data Selection

In order to put our data into a Linear Regression model, we need to separate out our features to model on and the Yelp ratings. From our correlation analysis we saw that the three features with the strongest correlations to Yelp rating are `average_review_sentiment`, `average_review_length`, and `average_review_age`. Since we want to dig a little deeper than `average_review_sentiment`, which understandably has a very high correlation with Yelp rating, let's choose to create our first model with `average_review_length` and `average_review_age` as features.

Pandas lets us select one column of a DataFrame with the following syntax:

```
subset_of_data = df['feature_to_select']
```

Pandas also lets us select multiple columns from a DataFrame with this syntax:

```
subset_of_data = df[list_of_features_to_select]
```

Create a new DataFrame `features` that contains the columns we want to model on: `average_review_length` and `average_review_age`. Then create another DataFrame `ratings` that stores the value we want to predict, Yelp rating, or `stars` in `df`.

In [38]:
```
features = df[['average_review_length', 'average_review_age']]
ratings = df['stars']
```

Split the Data into Training and Testing Sets

We are just about ready to model! But first, we need to break our data into a training set and a test set so we can evaluate how well our model performs. We'll use scikit-learn's `train_test_split` function to do this split, which is provided in the cell below. This function takes two required parameters: the data, or our features, followed by our dependent variable, in our case the Yelp rating. Set the optional parameter `test_size` to be `0.2`. Finally, set the optional parameter `random_state` to `1`. This will make it so your data is split in the same way as the data in our solution code.

Remember, this function returns 4 items in this order:

1. The training data (features), which we can assign to `X_train`
2. The testing data (features), which we can assign to `X_test`
3. The training dependent variable (Yelp rating), which we can assign to `y_train`
4. The testing dependent variable (Yelp rating), which we can assign to `y_test`

In [39]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(features, ratings, train_size =
0.8, test_size = 0.2, random_state=1)
```

Create and Train the Model

Now that our data is split into training and testing sets, we can finally model! In the cell below we have provided the code to import `LinearRegression` from scikit-learn's `linear_model` module. Create a new `LinearRegression` object named model. The `.fit()` method will fit our Linear Regression model to our training data and calculate the coefficients for our features. Call the `.fit()` method on `model` with `X_train` and `y_train` as parameters. Just like that our model has now been trained on our training data!

In [40]:
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
model = lm.fit(x_train, y_train)
```

Evaluate and Understand the Model

Now we can evaluate our model in a variety of ways. The first way will be by using the `.score()` method, which provides the R^2 value for our model. Remember, R^2 is the coefficient of determination, or a measure of how much of the variance in our dependent variable, the predicted Yelp rating, is explained by our independent variables, our feature data. R^2 values range from `0` to `1`, with `0` indicating that the created model does not fit our data at all, and with `1` indicating the model perfectly fits our feature data. Call `.score()` on our model with `X_train` and `y_train` as parameters to calculate our training R^2 score. Then call `.score()` again on model with `X_test` and `y_test` as parameters to calculate R^2 for our testing data. What do these R^2 values say about our model? Do you think these features alone are able to effectively predict Yelp ratings?

In [42]:
```
print(lm.score(x_train, y_train))  # 0.0825...
print(lm.score(x_test, y_test))    # 0.0808...
# the two fetaures included seem to account for 8% of the fit, certainly not enough
to effectively predict Yelp ratings on
```

```
0.0825030956654
0.0808308121006
```

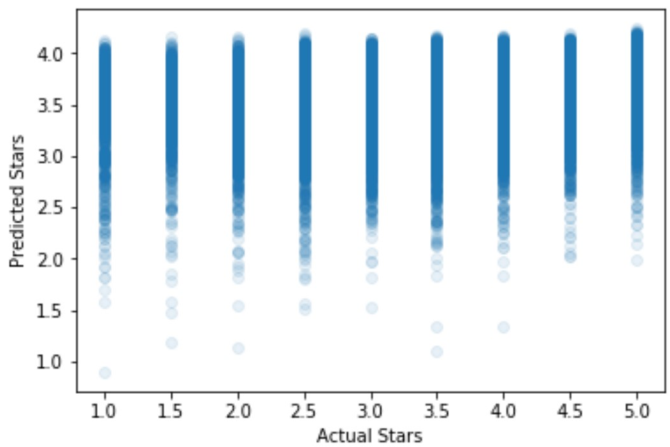
After all that hard work, we can finally take a look at the coefficients on our different features! The model has an attribute `.coef_` which is an array of the feature coefficients determined by fitting our model to the training data. To make it easier for you to see which feature corresponds to which coefficient, we have provided some code in the cell that `zip`s together a list of our features with the coefficients and sorts them in descending order from most predictive to least predictive.

In [43]:
```
sorted(list(zip(['average_review_length','average_review_age'],model.coef_)),key =
lambda x: abs(x[1]),reverse=True)
```

Out[43]:
```
[('average_review_length', -0.0009977176852074562 7),
 ('average_review_age', -0.00011621626836366434)]
```

Lastly we can calculate the predicted Yelp ratings for our testing data and compare them to their actual Yelp ratings! Our model has a `.predict()` method which uses the model's coefficients to calculate the predicted Yelp rating. Call `.predict()` on `X_test` and assign the values to `y_predicted`. Use Matplotlib to plot `y_test` vs `y_predicted`. For a perfect linear regression model we would expect to see the data plotted along the line `y = x`, indicating homoscedasticity. Is this the case? If not, why not? Would you call this model heteroscedastic or homoscedastic?

In [46]:
```
y_predicted = model.predict(x_test)
plt.scatter(y_test, y_predicted,alpha=0.1)
plt.ylabel('Predicted Stars')
plt.xlabel('Actual Stars')
plt.show()
```



Define Different Subsets of Data

After evaluating the first model, you can see that `average_review_length` and `average_review_age` alone are not the best predictors for Yelp rating. Let's go do some more modeling with different subsets of features and see if we can achieve a more accurate model! In the cells below we have provided different lists of subsets of features that we will model with and evaluate. What other subsets of features would you want to test? Why do you think those feature sets are more predictive of Yelp rating than others? Create at least one more subset of features that you want to predict Yelp ratings from.

In [47]:
```
# subset of only average review sentiment
sentiment = ['average_review_sentiment']
```

In [48]:
```
# subset of all features that have a response range [0,1]
binary_features = ['alcohol?','has_bike_parking','takes_credit_cards','good_for_kid
s','take_reservations','has_wifi']
```

In [49]:
```
# subset of all features that vary on a greater range than [0,1]
numeric_features = ['review_count','price_range','average_caption_length','number_pics','average_review_age','average_review_length','average_review_sentiment','number_funny_votes','number_cool_votes','number_useful_votes','average_tip_length','number_tips','average_number_friends','average_days_on_yelp','average_number_fans','average_review_count','average_number_years_elite','weekday_checkins','weekend_checkins']
```

In [50]:
```
# all features
all_features = binary_features + numeric_features
```

In [51]:
```
# add your own feature subset here
feature_subset = ['take_reservations', 'number_pics']
```

Further Modeling

Now that we have lists of different feature subsets, we can create new models from them. In order to more easily compare the performance of these new models, we have created a function for you below called `model_these_features()`. This function replicates the model building process you just completed with our first model! Take some time to review how the function works, analyzing it line by line. Fill in the empty comments with an explanation of the task the code beneath it is performing.

In [54]:
```
import numpy as np

# take a list of features to model as a parameter
def model_these_features(feature_list):

    #
    ratings = df.loc[:,'stars']
    features = df.loc[:,feature_list]

    #
    X_train, X_test, y_train, y_test = train_test_split(features, ratings, test_siz
e = 0.2, random_state = 1)

    # don't worry too much about these lines, just know that they allow the model t
o work when
    # we model on just one feature instead of multiple features. Trust us on this o
ne :)
    if len(X_train.shape) < 2:
        X_train = np.array(X_train).reshape(-1,1)
        X_test = np.array(X_test).reshape(-1,1)

    #
    model = LinearRegression()
    model.fit(X_train,y_train)

    #
    print('Train Score:', model.score(X_train,y_train))
    print('Test Score:', model.score(X_test,y_test))

    # print the model features and their corresponding coefficients, from most pred
ictive to least predictive
    print(sorted(list(zip(feature_list,model.coef_)),key = lambda x: abs(x[1]),reve
rse=True))

    #
    y_predicted = model.predict(X_test)

    #
    plt.scatter(y_test,y_predicted,alpha=0.1)
    plt.xlabel('Yelp Rating')
    plt.ylabel('Predicted Yelp Rating')
    plt.ylim(1,5)
    plt.show()
```

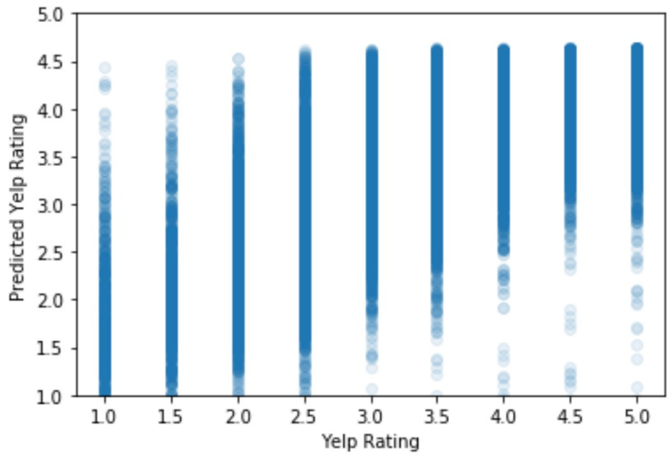
Once you feel comfortable with the steps of the function, run models on the following subsets of data using `model_these_features()`:

- `sentiment`: only `average_review_sentiment`
- `binary_features`: all features that have a response range [0,1]
- `numeric_features`: all features that vary on a greater range than [0,1]
- `all_features`: all features
- `feature_subset`: your own feature subset

How does changing the feature sets affect the model's R^2 value? Which features are most important to predicting Yelp rating in the different models? Which models appear more or less homoscedastic?

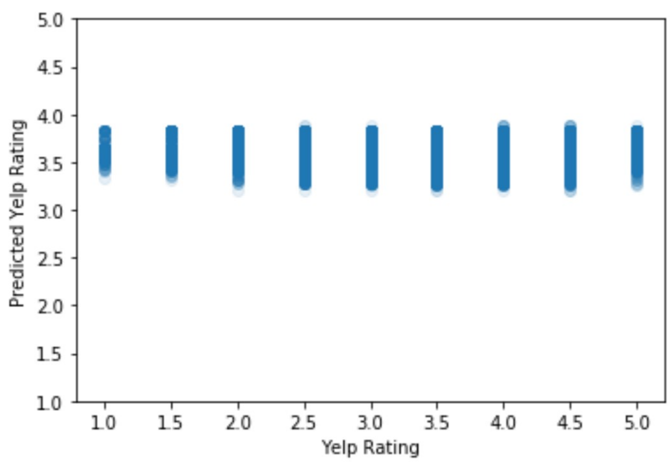
In [55]:
```
# create a model on sentiment here
model_these_features(sentiment)
```

```
Train Score: 0.611898095044
Test Score: 0.611402104692
[('average_review_sentiment', 2.3033908433749879)]
```



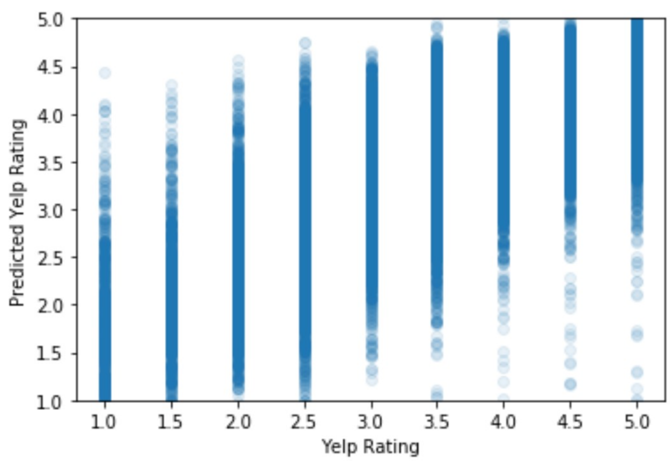
In [56]:
```
# create a model on all binary features here
model_these_features(binary_features)
```

```
Train Score: 0.0122231807096
Test Score: 0.0101195422023
[('has_bike_parking', 0.19003008208047414), ('alcohol?', -0.14549670708131
94), ('has_wifi', -0.13187397577759247), ('good_for_kids', -0.086324859903
96292), ('takes_credit_cards', 0.071755364921951156), ('take_reservations'
, 0.045265585304516562)]
```



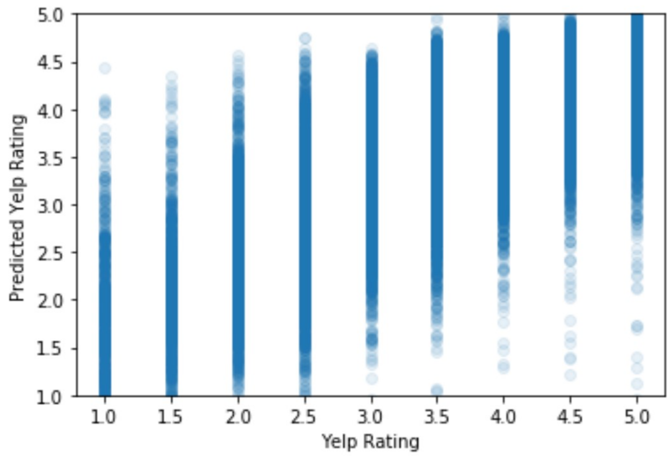
In [57]:
```
# create a model on all numeric features here
model_these_features(numeric_features)
```

```
Train Score: 0.673499259377
Test Score: 0.671331879812
[('average_review_sentiment', 2.2721076642096278), ('price_range', -0.0804608096
27016946), ('average_number_years_elite', -0.071903662880541883), ('average_capt
ion_length', -0.0033470660077858055), ('number_pics', -0.0029565028128971751), (
'number_tips', -0.0015953050789037789), ('number_cool_votes', 0.0011468839227085
428), ('average_number_fans', 0.0010510602097440632), ('average_review_length',
-0.00058136556920946637), ('average_tip_length', -0.00053220320634608505), ('num
ber_useful_votes', -0.00023203784758731028), ('average_review_count', -0.0002243
1702895059404), ('average_review_age', -0.00016930608165074462), ('average_days_o
n_yelp', 0.00012878025876703232), ('weekday_checkins', 5.9185807544714919e-05),
('weekend_checkins', -5.5181762069832219e-05), ('average_number_friends', 4.8269
921115993899e-05), ('review_count', -3.4834837637279341e-05), ('number_funny_vot
es', -7.8843956739498705e-06)]
```



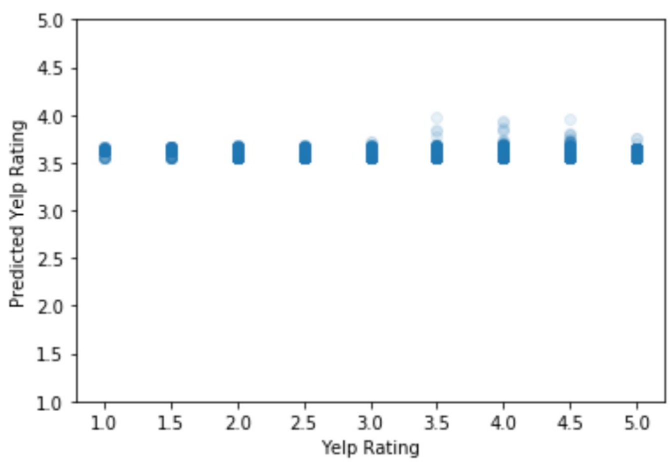
In [58]:
```
# create a model on all features here
model_these_features(all_features)
```

```
Train Score: 0.68078288619
Test Score: 0.678212904587
[('average_review_sentiment', 2.280845699662422), ('alcohol?', -0.14991498593462
138), ('has_wifi', -0.12155382629258796), ('good_for_kids', -0.11807814422014297
), ('price_range', -0.064867301500426272), ('average_number_years_elite', -0.062
789397138954553), ('has_bike_parking', 0.027296969912311748), ('takes_credit_car
ds', 0.024451837853664511), ('take_reservations', 0.01413455917298124), ('number
_pics', -0.0013133612300805335), ('average_number_fans', 0.0010267986822655923),
('number_cool_votes', 0.00097237227344138628), ('number_tips', -0.00085465633208
795726), ('average_caption_length', -0.00064727497981929762), ('average_review_l
ength', -0.00058962579202722406), ('average_tip_length', -0.00042052175034053827
), ('number_useful_votes', -0.0002715064125617837), ('average_review_count', -0.
00023398356902507478), ('average_review_age', -0.00015776544111326118), ('averag
e_days_on_yelp', 0.00012326147662884875), ('review_count', 0.0001011225937738620
4), ('weekend_checkins', -9.2396174696432465e-05), ('weekday_checkins', 6.153909
1231476578e-05), ('number_funny_votes', 4.8479351025272795e-05), ('average_numbe
r_friends', 2.0695840373706215e-05)]
```



In [59]:
```
# create a model on your feature subset here
model_these_features(feature_subset)
```

```
Train Score: 0.000605976998994
Test Score: 0.000647066119144
[('take_reservations', -0.083658617735313467), ('number_pics', 0.000888443935213
20998)]
```



Danielle's Delicious Delicacies' Debut

You've loaded the data, cleaned it, modeled it, and evaluated it. You're tired, but glowing with pride after all the hard work. You close your eyes and can clearly see opening day of Danielle's Delicious Delicacies with a line out the door. But what will your Yelp rating be? Let's use our model to make a prediction.

Our best model was the model using all features, so we'll work with this model again. In the cell below print `all_features` to get a reminder of what features we are working with.

In [60]:
```
print(all_features)
```

```
['alcohol?', 'has_bike_parking', 'takes_credit_cards', 'good_for_kids', 'take_re
servations', 'has_wifi', 'review_count', 'price_range', 'average_caption_length'
, 'number_pics', 'average_review_age', 'average_review_length', 'average_review_
sentiment', 'number_funny_votes', 'number_cool_votes', 'number_useful_votes', 'a
verage_tip_length', 'number_tips', 'average_number_friends', 'average_days_on_ye
lp', 'average_number_fans', 'average_review_count', 'average_number_years_elite'
, 'weekday_checkins', 'weekend_checkins']
```

Run the cell below to grab all the features and retrain our model on them.

In [61]:
```
features = df.loc[:,all_features]
ratings = df.loc[:,'stars']
X_train, X_test, y_train, y_test = train_test_split(features, ratings, test_size =
0.2, random_state = 1)
model = LinearRegression()
model.fit(X_train,y_train)
```

Out[61]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

To give you some perspective on the restaurants already out there, we have provided the mean, minimum, and maximum values for each feature below. Will Danielle's Delicious Delicacies be just another average restaurant, or will it be a 5 star behemoth amongst the masses?

In [62]:
```
pd.DataFrame(list(zip(features.columns,features.describe().loc['mean'],features.describe().loc['min'],features.describe().loc['max'])),columns=['Feature','Mean','Min','Max'])
```

Out[62]:

	Feature	Mean	Min	Max
0	alcohol?	0.140610	0.000000	1.000000
1	has_bike_parking	0.350692	0.000000	1.000000
2	takes_credit_cards	0.700243	0.000000	1.000000
3	good_for_kids	0.279029	0.000000	1.000000
4	take_reservations	0.106086	0.000000	1.000000
5	has_wifi	0.134968	0.000000	1.000000
6	review_count	31.797310	3.000000	7968.000000
7	price_range	1.035855	0.000000	4.000000
8	average_caption_length	2.831829	0.000000	140.000000
9	number_pics	1.489939	0.000000	1150.000000
10	average_review_age	1175.501021	71.555556	4727.333333
11	average_review_length	596.463567	62.400000	4229.000000
12	average_review_sentiment	0.554935	-0.995200	0.996575
13	number_funny_votes	15.617091	0.000000	36822.000000
14	number_cool_votes	18.495973	0.000000	6572.000000
15	number_useful_votes	43.515279	0.000000	38357.000000
16	average_tip_length	45.643426	0.000000	500.000000
17	number_tips	6.285217	0.000000	3581.000000
18	average_number_friends	105.132000	1.000000	4219.000000
19	average_days_on_yelp	2005.367009	76.000000	4860.000000
20	average_number_fans	11.590148	0.000000	1174.666667
21	average_review_count	122.110660	0.666667	6335.000000
22	average_number_years_elite	0.923313	0.000000	10.666667
23	weekday_checkins	45.385094	0.000000	73830.000000
24	weekend_checkins	49.612515	0.000000	64647.000000

Based on your plans for the restaurant, how you expect your customers to post on your Yelp page, and the values above, fill in the blanks in the NumPy array below with your desired values. The first blank corresponds with the feature at `index=0` in the DataFrame above, `alcohol?`, and the last blank corresponds to the feature at `index=24`, `weekend_checkins`. Make sure to enter either `0` or `1` for all binary features, and if you aren't sure of what value to put for a feature, select the mean from the DataFrame above. After you enter the values, run the prediction cell below to receive your Yelp rating! How is Danielle's Delicious Delicacies debut going to be?

In [63]:
```
danielles_delicious_delicacies = np.array([1,1,1,0,0,0,32,2,3,2,72,600,0.75,10,20,45,45,6,105,100,25,100,0,50,100]).reshape(1,-1)
```

In [64]:
```
model.predict(danielles_delicious_delicacies)
```

Out[64]:
```
array([ 3.96635797])
```

Next Steps

You have successfully built a linear regression model that predicts a restaurant's Yelp rating! As you have seen, it can be pretty hard to predict a rating like this even when we have a plethora of data. What other questions come to your mind when you see the data we have? What insights do you think could come from a different kind of analysis? Here are some ideas to ponder:

- Can we predict the cuisine of a restaurant based on the users that review it?
- What restaurants are similar to each other in ways besides cuisine?
- Are there different restaurant vibes, and what kind of restaurants fit these conceptions?
- How does social media status affect a restaurant's credibility and visibility?

As you progress further into the field of data science, you will be able to create models that address these questions and many more! But in the meantime, get back to working on that burgeoning restaurant business plan.