Analysis of Yelp Business Intelligence Data

We will analyze a subset of Yelp's business, reviews and user data. This dataset comes to us from Kaggle although we have taken steps to pull this data into a publis s3 bucket: `s3://sta9760-yelpdataset/yelp-light/*business.json`

Installation and Initial Setup

Begin by installing the necessary libraries that you may need to conduct your analysis. At the very least, you must install `pandas` and `matplotlib`

In [1]:
```
sc.install_pypi_package("matplotlib==3.2.1")
sc.install_pypi_package("pandas==1.0.3")
sc.install_pypi_package("seaborn==0.10.0")
```

Starting Spark application

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1619497129448_0001	pyspark	idle	Link	Link	✔

```
SparkSession available as 'spark'.

Collecting matplotlib==3.2.1
  Downloading https://files.pythonhosted.org/packages/b2/c2/71fcf957710f3ba1f09088b35776a799ba7dd95f7c2b195ec800933b276b/
matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl (12.4MB)
Collecting python-dateutil>=2.1 (from matplotlib==3.2.1)
  Downloading https://files.pythonhosted.org/packages/d4/70/d60450c3dd48ef87586924207ae8907090de0b306af2bce5d134d78615cb/
python_dateutil-2.8.1-py2.py3-none-any.whl (227kB)
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 (from matplotlib==3.2.1)
  Downloading https://files.pythonhosted.org/packages/8a/bb/488841f56197b13700afd5658fc279a2025a39e22449b7cf29864669b15d/
pyparsing-2.4.7-py2.py3-none-any.whl (67kB)
Collecting cycler>=0.10 (from matplotlib==3.2.1)
  Downloading https://files.pythonhosted.org/packages/f7/d2/e07d3ebb2bd7af696440ce7e754c59dd546ffe1bbe732c8ab68b9c834e61/
cycler-0.10.0-py2.py3-none-any.whl
Requirement already satisfied: numpy>=1.11 in /usr/local/lib64/python3.7/site-packages (from matplotlib==3.2.1)
Collecting kiwisolver>=1.0.1 (from matplotlib==3.2.1)
  Downloading https://files.pythonhosted.org/packages/d2/46/231de802ade4225b76b96cffe419cf3ce52bbe92e3b092cf12db7d11c207/
kiwisolver-1.3.1-cp37-cp37m-manylinux1_x86_64.whl (1.1MB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil>=2.1->matplotlib=
=3.2.1)
Installing collected packages: python-dateutil, pyparsing, cycler, kiwisolver, matplotlib
```

```
Successfully installed cycler-0.10.0 kiwisolver-1.3.1 matplotlib-3.2.1 pyparsing-2.4.7 python-dateutil-2.8.1

Collecting pandas==1.0.3
  Downloading https://files.pythonhosted.org/packages/4a/6a/94b219b8ea0f2d580169e85ed1edc0163743f55aaeca8a44c2e8fc1e344e/
pandas-1.0.3-cp37-cp37m-manylinux1_x86_64.whl (10.0MB)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/site-packages (from pandas==1.0.3)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib64/python3.7/site-packages (from pandas==1.0.3)
Requirement already satisfied: python-dateutil>=2.6.1 in /mnt/tmp/1619497281646-0/lib/python3.7/site-packages (from panda
s==1.0.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil>=2.6.1->pandas==
1.0.3)
Installing collected packages: pandas
Successfully installed pandas-1.0.3

Collecting seaborn==0.10.0
  Downloading https://files.pythonhosted.org/packages/70/bd/5e6bf595fe6ee0f257ae49336dd180768c1ed3d7c7155b2fdf894c1c808a/
seaborn-0.10.0-py3-none-any.whl (215kB)
Requirement already satisfied: pandas>=0.22.0 in /mnt/tmp/1619497281646-0/lib/python3.7/site-packages (from seaborn==0.1
0.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib64/python3.7/site-packages (from seaborn==0.10.0)
Collecting scipy>=1.0.1 (from seaborn==0.10.0)
  Downloading https://files.pythonhosted.org/packages/7d/e8/43ffca541d2f208d516296950b25fe1084b35c2881f4d444c1346ca75815/
scipy-1.6.3-cp37-cp37m-manylinux1_x86_64.whl (27.4MB)
Requirement already satisfied: matplotlib>=2.1.2 in /mnt/tmp/1619497281646-0/lib/python3.7/site-packages (from seaborn==
0.10.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/site-packages (from pandas>=0.22.0->seaborn==0.1
0.0)
Requirement already satisfied: python-dateutil>=2.6.1 in /mnt/tmp/1619497281646-0/lib/python3.7/site-packages (from panda
s>=0.22.0->seaborn==0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /mnt/tmp/1619497281646-0/lib/python3.7/site-pa
ckages (from matplotlib>=2.1.2->seaborn==0.10.0)
Requirement already satisfied: cycler>=0.10 in /mnt/tmp/1619497281646-0/lib/python3.7/site-packages (from matplotlib>=2.
1.2->seaborn==0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /mnt/tmp/1619497281646-0/lib/python3.7/site-packages (from matplotlib
>=2.1.2->seaborn==0.10.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil>=2.6.1->pandas>=
0.22.0->seaborn==0.10.0)
Installing collected packages: scipy, seaborn
Successfully installed scipy-1.6.3 seaborn-0.10.0
```

Importing

Now, import the installed packages from the previous block below.

In [2]:

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

```
import seaborn as sns
import numpy as np
```

Loading Data

We are finally ready to load data. Using `spark` load the data from S3 into a `dataframe` object that we can manipulate further down in our analysis.

In [3]:
```
review = spark.read.json('s3://sta9760-project-datasets/yelp_academic_dataset_review.json')
business = spark.read.json('s3://sta9760-project-datasets/yelp_academic_dataset_business.json')
user = spark.read.json('s3://sta9760-project-datasets/yelp_academic_dataset_user.json')
```

Overview of Data

Display the number of rows and columns in our dataset.

In [4]:
```
print("Columns:", len(business.columns), "| Rows:", business.count())
```

```
Columns: 14 | Rows: 160585
```

Display the DataFrame schema below.

In [5]:
```
business.printSchema()
```

```
root
 |-- address: string (nullable = true)
 |-- attributes: struct (nullable = true)
 |    |-- AcceptsInsurance: string (nullable = true)
 |    |-- AgesAllowed: string (nullable = true)
 |    |-- Alcohol: string (nullable = true)
 |    |-- Ambience: string (nullable = true)
 |    |-- BYOB: string (nullable = true)
 |    |-- BYOBCorkage: string (nullable = true)
 |    |-- BestNights: string (nullable = true)
```

```
 |    |-- BikeParking: string (nullable = true)
 |    |-- BusinessAcceptsBitcoin: string (nullable = true)
 |    |-- BusinessAcceptsCreditCards: string (nullable = true)
 |    |-- BusinessParking: string (nullable = true)
 |    |-- ByAppointmentOnly: string (nullable = true)
 |    |-- Caters: string (nullable = true)
 |    |-- CoatCheck: string (nullable = true)
 |    |-- Corkage: string (nullable = true)
 |    |-- DietaryRestrictions: string (nullable = true)
 |    |-- DogsAllowed: string (nullable = true)
 |    |-- DriveThru: string (nullable = true)
 |    |-- GoodForDancing: string (nullable = true)
 |    |-- GoodForKids: string (nullable = true)
 |    |-- GoodForMeal: string (nullable = true)
 |    |-- HairSpecializesIn: string (nullable = true)
 |    |-- HappyHour: string (nullable = true)
 |    |-- HasTV: string (nullable = true)
 |    |-- Music: string (nullable = true)
 |    |-- NoiseLevel: string (nullable = true)
 |    |-- Open24Hours: string (nullable = true)
 |    |-- OutdoorSeating: string (nullable = true)
 |    |-- RestaurantsAttire: string (nullable = true)
 |    |-- RestaurantsCounterService: string (nullable = true)
 |    |-- RestaurantsDelivery: string (nullable = true)
 |    |-- RestaurantsGoodForGroups: string (nullable = true)
 |    |-- RestaurantsPriceRange2: string (nullable = true)
 |    |-- RestaurantsReservations: string (nullable = true)
 |    |-- RestaurantsTableService: string (nullable = true)
 |    |-- RestaurantsTakeOut: string (nullable = true)
 |    |-- Smoking: string (nullable = true)
 |    |-- WheelchairAccessible: string (nullable = true)
 |    |-- WiFi: string (nullable = true)
 |-- business_id: string (nullable = true)
 |-- categories: string (nullable = true)
 |-- city: string (nullable = true)
 |-- hours: struct (nullable = true)
 |    |-- Friday: string (nullable = true)
 |    |-- Monday: string (nullable = true)
 |    |-- Saturday: string (nullable = true)
 |    |-- Sunday: string (nullable = true)
 |    |-- Thursday: string (nullable = true)
 |    |-- Tuesday: string (nullable = true)
 |    |-- Wednesday: string (nullable = true)
 |-- is_open: long (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- name: string (nullable = true)
 |-- postal_code: string (nullable = true)
 |-- review_count: long (nullable = true)
```

```
 |-- stars: double (nullable = true)
 |-- state: string (nullable = true)
```

Display the first 5 rows with the following columns:

- `business_id`
- `name`
- `city`
- `state`
- `categories`

In [6]:
```
business.select("business_id", "name", "city", "state", "stars", "categories").show(5)
```

```
+--------------------+--------------------+-----------+-----+-----+--------------------+
|         business_id|                name|       city|state|stars|          categories|
+--------------------+--------------------+-----------+-----+-----+--------------------+
|6iYb2HFDywm3zjuRg...| Oskar Blues Taproom|    Boulder|   CO|  4.0|Gastropubs, Food,...|
|tCbdrRPZA0oiIYSmH...|Flying Elephants ...|   Portland|   OR|  4.0|Salad, Soup, Sand...|
|bvN78flM8NLprQ1a1...|      The Reclaimory|   Portland|   OR|  4.5|Antiques, Fashion...|
|oaepsyvc0J17qwi8c...|         Great Clips|Orange City|   FL|  3.0|Beauty & Spas, Ha...|
|PE9uqAjdw0E4-8mjG...|   Crossfit Terminus|    Atlanta|   GA|  4.0|Gyms, Active Life...|
+--------------------+--------------------+-----------+-----+-----+--------------------+
only showing top 5 rows
```

Analyzing Categories

Let's now answer this question: **how many unique categories are represented in this dataset?**

Essentially, we have the categories per business as a list - this is useful to quickly see what each business might be represented as but it is difficult to easily answer questions such as:

- How many businesses are categorized as `Active Life`, for instance
- What are the top 20 most popular categories available?

Association Table

We need to "break out" these categories from the business ids? One common approach to take is to build an association table mapping a single business id multiple times to each distinct category.

For instance, given the following:

business_id	categories
abcd123	a,b,c

We would like to derive something like:

business_id	category
abcd123	a
abcd123	b
abcd123	c

What this does is allow us to then perform a myriad of rollups and other analysis on this association table which can aid us in answering the questions asked above.

Implement the code necessary to derive the table described from your original yelp dataframe.

In [7]:

```
from pyspark.sql.functions import explode, split
category_explode = business.withColumn("category", explode(split("categories", ", ")))
```

Display the first 5 rows of your association table below.

In [8]:

```
category_explode.select("business_id", "category").show(5)
```

```
+--------------------+------------+
|         business_id|    category|
+--------------------+------------+
|6iYb2HFDywm3zjuRg...|  Gastropubs|
|6iYb2HFDywm3zjuRg...|        Food|
|6iYb2HFDywm3zjuRg...|Beer Gardens|
|6iYb2HFDywm3zjuRg...| Restaurants|
|6iYb2HFDywm3zjuRg...|        Bars|
+--------------------+------------+
only showing top 5 rows
```

Total Unique Categories

Finally, we are ready to answer the question: **what is the total number of unique categories available?**

Below, implement the code necessary to calculate this figure.

In [9]:

```
category_explode.select("category").distinct().count()
```

```
1330
```

Top Categories By Business

Now let's find the top categories in this dataset by rolling up categories.

Counts of Businesses / Category

So now, let's unroll our distinct count a bit and display the per count value of businesses per category.

The expected output should be:

category	count
a	15
b	2
c	45

Or something to that effect.

In [10]:

```
category_explode.groupby("category").count().show()
```

```
+--------------------+-----+
|            category|count|
+--------------------+-----+
|      Dermatologists|  351|
|      Paddleboarding|   67|
|        Aerial Tours|    8|
```

```
|         Hobby Shops|  610|
|          Bubble Tea|  779|
|             Embassy|    9|
|             Tanning|  701|
|            Handyman|  507|
|      Aerial Fitness|   13|
|             Falafel|  141|
|       Outlet Stores|  184|
|        Summer Camps|  308|
|     Clothing Rental|   37|
|      Sporting Goods| 1864|
|     Cooking Schools|  114|
|  College Counseling|   20|
|  Lactation Services|   47|
|Ski & Snowboard S...|   55|
|             Museums|  336|
|     Baseball Fields|   17|
+--------------------+-----+
only showing top 20 rows
```

Bar Chart of Top Categories

With this data available, let us now build a barchart of the top 20 categories.

HINT: don't forget about the matplotlib magic!

```
%matplot plt
```

In [11]:

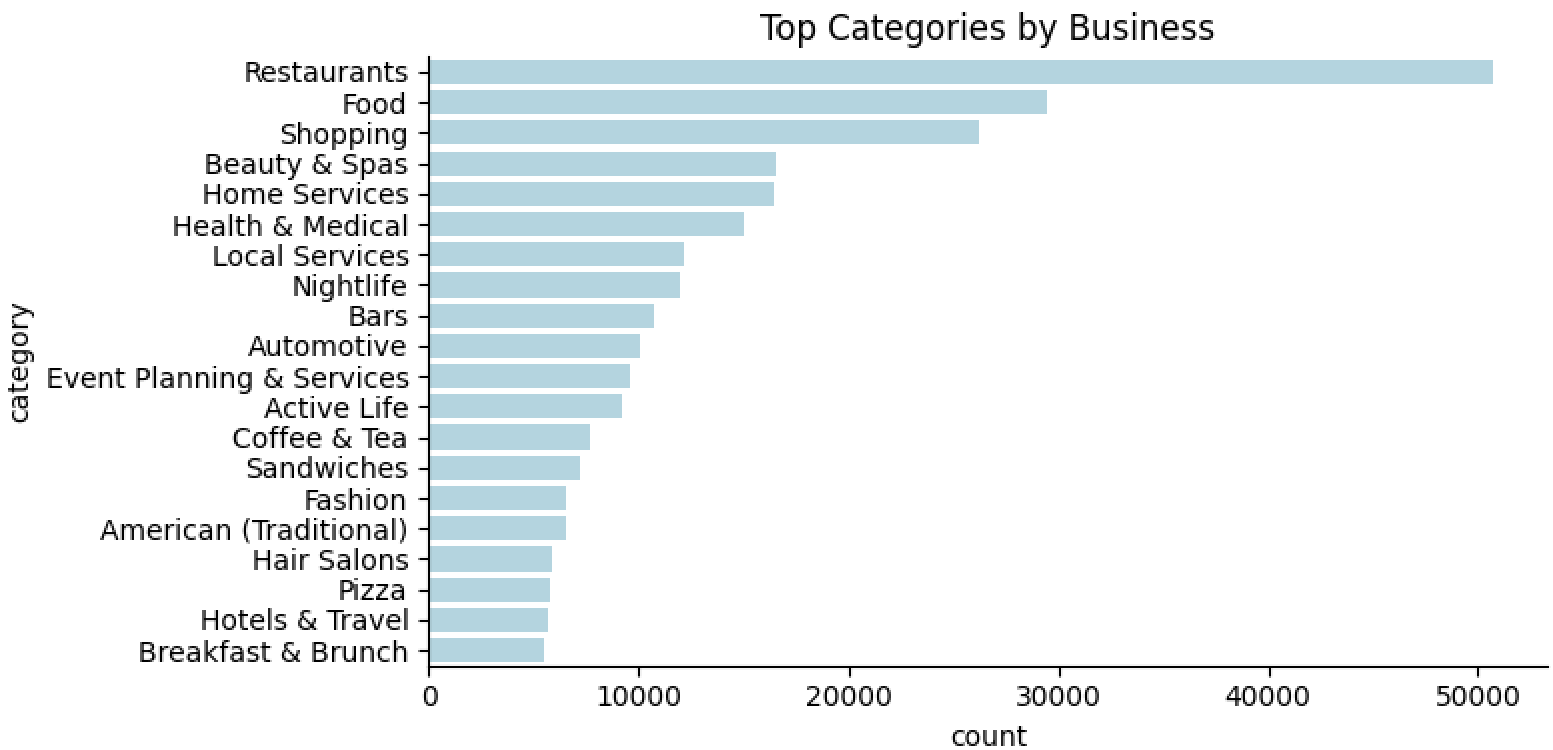
```
barchart_df = category_explode.groupby("category").count().orderBy("count", ascending = False)
```

In [12]:

```
pdf = barchart_df.toPandas()
pdf = pdf.head(20)
```

In [13]:

```
sns.factorplot(x="count", y="category", data=pdf, size=4, aspect=2, kind="bar", color="lightblue")
plt.title("Top Categories by Business")
plt.tight_layout()
%matplot plt
```



Do Yelp Reviews Skew Negative?

Oftentimes, it is said that the only people who write a written review are those who are extremely *dissatisfied* or extremely *satisfied* with the service received.

How true is this really? Let's try and answer this question.

Loading User Data

Begin by loading the user data set from S3 and printing schema to determine what data is available.

In [14]:
```
review.printSchema()
```

```
root
 |-- business_id: string (nullable = true)
 |-- cool: long (nullable = true)
 |-- date: string (nullable = true)
 |-- funny: long (nullable = true)
 |-- review_id: string (nullable = true)
 |-- stars: double (nullable = true)
 |-- text: string (nullable = true)
 |-- useful: long (nullable = true)
 |-- user_id: string (nullable = true)
```

Let's begin by listing the `business_id` and `stars` columns together for the user reviews data.

In [15]:

```
stars = review.select("business_id", "stars")
stars.show(5)
```

```
+--------------------+-----+
|         business_id|stars|
+--------------------+-----+
|buF9druCkbuXLX526...|  4.0|
|RA4V8pr014UyUbDvI...|  4.0|
|_sS2LBIGNT5NQb6PD...|  5.0|
|0AzLzHfOJgL7ROwhd...|  2.0|
|8zehGz9jnxPqXtOc7...|  4.0|
+--------------------+-----+
only showing top 5 rows
```

Now, let's aggregate along the `stars` column to get a resultant dataframe that displays *average stars* per business as accumulated by users who **took the time to submit a written review**.

In [16]:

```
average_stars = stars.groupby("business_id").avg("stars")
average_stars.show(5)
```

```
+--------------------+-----------------+
|         business_id|       avg(stars)|
+--------------------+-----------------+
|uEUweopM30lHcVxjO...|              3.0|
|wdBrDCbZopowEkIEX...|4.538461538461538|
|L3WCfeVozu5etMhz4...|              4.2|
|bOnsvrz1VkbrZM1jV...|              3.8|
|R0IJhEI-zSJpYT1YN...|3.606060606060606|
+--------------------+-----------------+
only showing top 5 rows
```

Now the fun part - let's join our two dataframes (reviews and business data) by `business_id` .

In [17]:
```
joined_df = average_stars.join(business, average_stars.business_id == business.business_id)
```

Let's see a few of these:

In [18]:
```
joined_df.select("avg(stars)", "stars", "name", "city", "state").show(5)
```

```
+-----------------+-----+-------------------+----------+-----+
|       avg(stars)|stars|               name|      city|state|
+-----------------+-----+-------------------+----------+-----+
|              5.0|  5.0|    CheraBella Salon|   Peabody|   MA|
|            3.875|  4.0|Mezcal Cantina & ...|  Columbus|   OH|
|3.866666666666667|  4.0|    Red Table Coffee|    Austin|   TX|
|              5.0|  5.0|          WonderWell|    Austin|   TX|
|            3.375|  3.5|         Avalon Oaks|Wilmington|   MA|
+-----------------+-----+-------------------+----------+-----+
only showing top 5 rows
```

Compute a new dataframe that calculates what we will call the *skew* (for lack of a better word) between the avg stars accumulated from written reviews and the *actual* star rating of a business (ie: the average of stars given by reviewers who wrote an actual review **and** reviewers who just provided a star rating).

The formula you can use is something like:

```
(row['avg(stars)'] - row['stars']) / row['stars']
```

If the **skew** is negative, we can interpret that to be: reviewers who left a written response were more dissatisfied than normal. If **skew** is positive, we can interpret that to be: reviewers who left a written response were more satisfied than normal.

In [19]:
```
skew_df = joined_df.select('stars', 'avg(stars)',
                           ((joined_df['avg(stars)'] - joined_df['stars']) / joined_df['stars']).alias('skew'))
```

In [20]:
```
skew_graph = skew_df.toPandas()
```

In [21]:

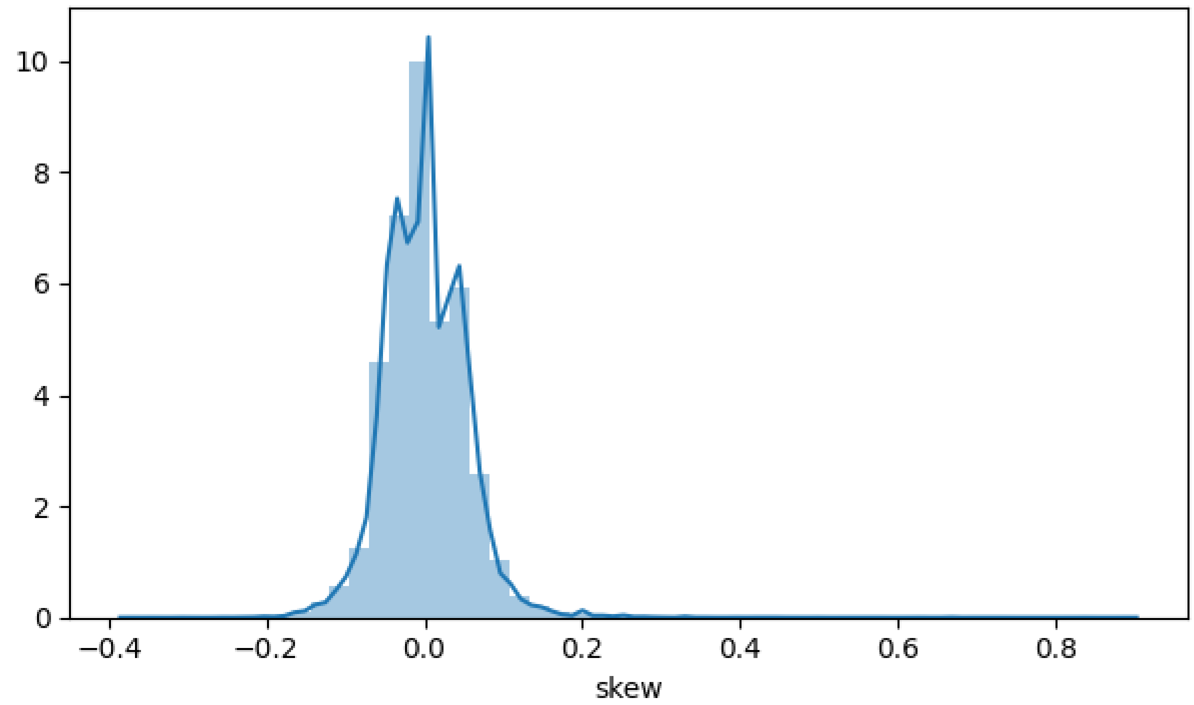
```
skew_graph.head(10)
```

```
   stars  avg(stars)      skew
0    3.0    3.000000  0.000000
1    4.5    4.538462  0.008547
2    4.0    4.200000  0.050000
3    4.0    3.800000 -0.050000
4    3.5    3.606061  0.030303
5    4.5    4.666667  0.037037
6    4.5    4.714286  0.047619
7    2.5    2.450000 -0.020000
8    4.5    4.666667  0.037037
9    3.5    3.312500 -0.053571
```

And finally, graph it!

In [22]:

```
plt.clf()
sns.distplot(skew_graph['skew'])
sns.set_style('darkgrid')
plt.show()
%matplot plt
```



So, do Yelp (written) Reviews skew negative? Does this analysis actually prove anything? Expound on implications / interpretations of this graph.

The Yelp Reviews doesn't skew negative and instead shows a normal distribution. This proves that the rating given to the restaurant by Yelp is reflective of the ratings given by the users.

Should the Elite be Trusted? (Or, some other analysis of your choice)

For the final portion - you have a choice:

- Try and analyze some interesting dimension to this data. The **ONLY** requirement is that you must use the **Users** dataset and join on either the **business* or** reviews** dataset
- Or, you may try and answer the question posed: how accurate or close are the ratings of an "elite" user (check Users table schema) vs the actual business rating.

Feel free to use any and all methodologies at your disposal - only requirement is you must render one visualization in your analysis

In [23]:
```
user.printSchema()
```

```
root
 |-- average_stars: double (nullable = true)
 |-- compliment_cool: long (nullable = true)
 |-- compliment_cute: long (nullable = true)
 |-- compliment_funny: long (nullable = true)
 |-- compliment_hot: long (nullable = true)
 |-- compliment_list: long (nullable = true)
 |-- compliment_more: long (nullable = true)
 |-- compliment_note: long (nullable = true)
 |-- compliment_photos: long (nullable = true)
 |-- compliment_plain: long (nullable = true)
 |-- compliment_profile: long (nullable = true)
 |-- compliment_writer: long (nullable = true)
 |-- cool: long (nullable = true)
 |-- elite: string (nullable = true)
 |-- fans: long (nullable = true)
 |-- friends: string (nullable = true)
 |-- funny: long (nullable = true)
 |-- name: string (nullable = true)
 |-- review_count: long (nullable = true)
 |-- useful: long (nullable = true)
 |-- user_id: string (nullable = true)
 |-- yelping_since: string (nullable = true)
```

To get reviews made by "elite" users, first separate years in which the users were elites and also extract the year the reviews were made:

In [24]:
```
elite_users = user.withColumn("elite", explode(split("elite", ",")))
```

In [25]:
```
from pyspark.sql.functions import substring
review_years = review.withColumn("year", substring("date", 0, 4))
```

Then join the two tables together by user_id and by year:

In [26]:
```
elite_reviews = elite_users.join(review_years, (elite_users.user_id == review_years.user_id) &
```

```
(elite_users.elite == review_years.year))
```

In [27]:

```
elite_reviews.select('business_id', 'year', 'elite', 'stars').show(5)
```

```
+--------------------+----+-----+-----+
|         business_id|year|elite|stars|
+--------------------+----+-----+-----+
|C_k727ws82eMe9xtJ...|2015| 2015|  3.0|
|CoqddB5aS8YX2v7iU...|2015| 2015|  2.0|
|1E_JFdI6HfEhh3T_q...|2015| 2015|  2.0|
|38uoph699c-AjXfWw...|2015| 2015|  2.0|
|LcpQtJoYa0tMZ1pFF...|2015| 2015|  4.0|
+--------------------+----+-----+-----+
only showing top 5 rows
```

Then to get the average ratings of elite users for each restaurant group by the business_id to get the average ratings:

In [28]:

```
elite_ratings = elite_reviews.groupby("business_id").avg("stars")
elite_ratings.select('business_id', 'avg(stars)').show(5)
```

```
+--------------------+------------------+
|         business_id|        avg(stars)|
+--------------------+------------------+
|iZplHglW9WL2pc6Hr...|               4.0|
|6KGBXOeSJYf9ePdyA...|3.7578947368421054|
|quL0Dqop3Ni5qcwd2...| 3.511111111111111|
|fFbkFE9awsyIZCw1h...|               5.0|
|fb0rYTlR2qDhCV9gY...|4.2272727272727275|
+--------------------+------------------+
only showing top 5 rows
```

Merge with the business table to compare the average elite ratings with the business rating and calculate the skew:

In [29]:

```
elite_vs_business = elite_ratings.join(business, elite_ratings.business_id == business.business_id)
elite_vs_business.select('stars', 'avg(stars)').show(5)
```

```
+-----+------------------+
|stars|        avg(stars)|
```

```
+-----+------------------+
|  4.0|               4.2|
|  2.5|2.3333333333333335|
|  4.5| 4.852459016393443|
|  3.5|3.7578947368421054|
|  4.0| 4.214285714285714|
+-----+------------------+
only showing top 5 rows
```

In [30]:

```
elite_skew_df = elite_vs_business.select('stars', 'avg(stars)',
                    ((elite_vs_business['avg(stars)'] - elite_vs_business['stars']) / elite_vs_business['stars']).alias('
```

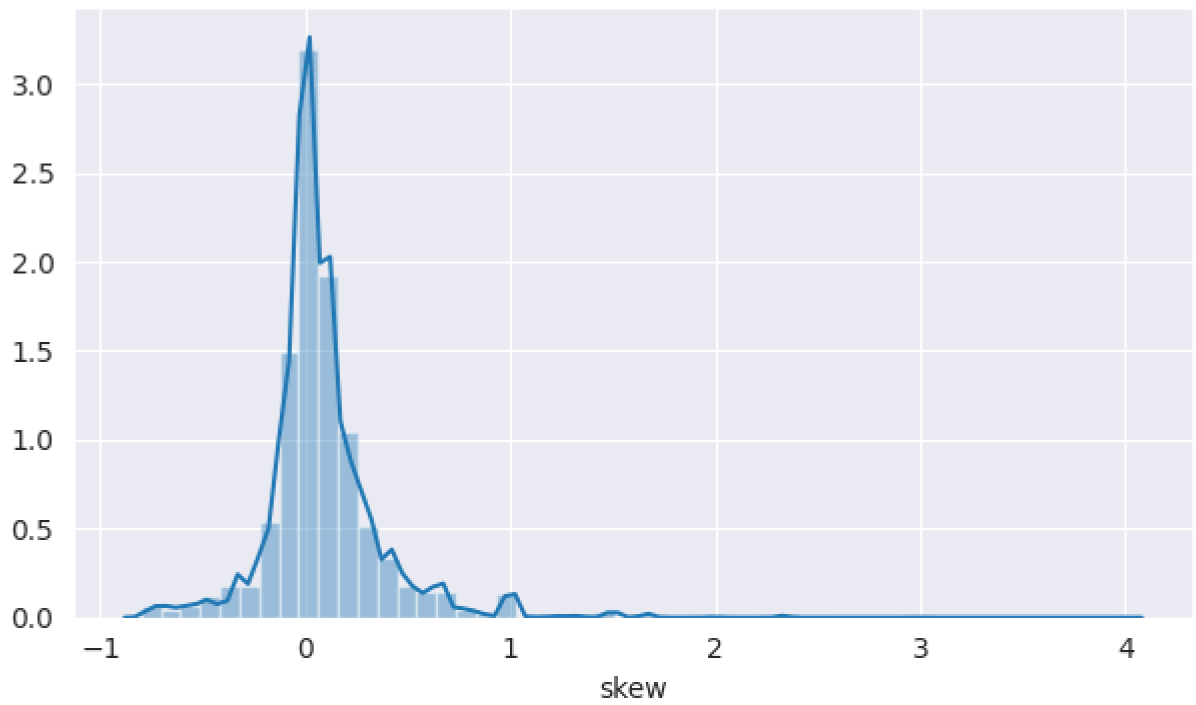
After graphing the skew you get:

In [31]:

```
elite_skew_graph = elite_skew_df.toPandas()
```

In [32]:

```
plt.clf()
sns.distplot(elite_skew_graph['skew'])
sns.set_style('darkgrid')
plt.show()
%matplot plt
```



The graph shows a normal distribution, meaning that the restaurants ratings are reflective of the reviews of the elite users. Since both the elite users and overall users have a similar distribution and reflect the overall restuarants ratings, it shows that the elite users can be trusted.