

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 public 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("pandas==1.0.3")
sc.install_pypi_package("matplotlib==3.2.1")
sc.install_pypi_package("seaborn==0.11.1")
```

Starting Spark application

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1619562798071_0001	pyspark	idle	Link	Link	✓

SparkSession available as 'spark'.

```
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)
Collecting python-dateutil>=2.6.1 (from pandas==1.0.3)
  Downloading https://files.pythonhosted.org/packages/d4/70/d60450c3dd48ef87586924207ae8907090de0b306af2bce5d134d78615cb/
python_dateutil-2.8.1-py2.py3-none-any.whl (227kB)
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: python-dateutil, pandas
Successfully installed pandas-1.0.3 python-dateutil-2.8.1
```

```
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)
Requirement already satisfied: python-dateutil>=2.1 in /mnt/tmp/1619563056840-0/lib/python3.7/site-packages (from matplotlib==3.2.1)
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/d2/46/231de802ade4225b76b96cff419cf3ce52bbe92e3b092cf12db7d11c207/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: pyparsing, cycler, kiwisolver, matplotlib
Successfully installed cycler-0.10.0 kiwisolver-1.3.1 matplotlib-3.2.1 pyparsing-2.4.7

Collecting seaborn==0.11.1
    Downloading https://files.pythonhosted.org/packages/68/ad/6c2406ae175f59ec616714e408979b674fe27b9587f79d59a528ddfbc5b/seaborn-0.11.1-py3-none-any.whl (285kB)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib64/python3.7/site-packages (from seaborn==0.11.1)
Collecting scipy>=1.0 (from seaborn==0.11.1)
    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>=2.2 in /mnt/tmp/1619563056840-0/lib/python3.7/site-packages (from seaborn==0.11.1)
Requirement already satisfied: pandas>=0.23 in /mnt/tmp/1619563056840-0/lib/python3.7/site-packages (from seaborn==0.11.1)
Requirement already satisfied: python-dateutil>=2.1 in /mnt/tmp/1619563056840-0/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn==0.11.1)
Requirement already satisfied: pyparsing!=2.0.4,!>=2.1.2,!>=2.1.6,>=2.0.1 in /mnt/tmp/1619563056840-0/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn==0.11.1)
Requirement already satisfied: cycler>=0.10 in /mnt/tmp/1619563056840-0/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn==0.11.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /mnt/tmp/1619563056840-0/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn==0.11.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/site-packages (from pandas>=0.23->seaborn==0.11.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil>=2.1->matplotlib>=2.2->seaborn==0.11.1)
Installing collected packages: scipy, seaborn
Successfully installed scipy-1.6.3 seaborn-0.11.1
```

Importing

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

In [2]:

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

```
import seaborn as sns
import pyspark.sql.functions as F
from pyspark.sql.functions import explode, split, col, udf, when, mean
from pyspark.sql.types import IntegerType, StringType, DoubleType
```

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]: business_df = spark.read.json('s3://sta9760-spark-project02-datasets/yelp_academic_dataset_business.json')
```

```
In [4]: review_df = spark.read.json('s3://sta9760-spark-project02-datasets/yelp_academic_dataset_review.json')
```

Overview of Data

Display the number of rows and columns in our dataset.

```
In [5]: print("Columns:", len(business_df.columns), "|", "Rows:", business_df.count())
```

Columns: 14 | Rows: 160585

Display the DataFrame schema below.

```
In [6]: business_df.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)
-- BYOCorkage: 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 [7]:

```
business_df_select = business_df.select("business_id", "name", "city", "state", "categories")
business_df_select.show(5)
```

business_id	name	city	state	categories
6iYb2HFDywm3zjuRg...	Oskar Blues Taproom	Boulder	CO	Gastropubs, Food,...
tCbdrrPZA0oiIYSmH...	Flying Elephants ...	Portland	OR	Salad, Soup, Sand...
bvN78f1M8NLprQ1a1...	The Reclaimory	Portland	OR	Antiques, Fashion...
oaepsyvc0J17qwi8c...	Great Clips	Orange City	FL	Beauty & Spas, Ha...
PE9uqAjdw0E4-8mjG...	Crossfit Terminus	Atlanta	GA	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 [8]: business_df_select2 = business_df.select("business_id","categories")
```

```
In [9]: business_df_explode = business_df_select2 \
    .withColumn('categories', explode(split('categories', ','))) \
    .withColumnRenamed('categories','category')
```

Display the first 5 rows of your association table below.

```
In [10]: business_df_explode.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 [11]: business_df_explode.select("category").distinct().count()
```

```
2487
```

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 [12]:

```
business_df_grouped=business_df_explode.groupby("category")
business_df_grouped.count().show(20)
```

category	count
Paddleboarding	12
Dermatologists	68
Tires	1456
Historical Tours	60
Hakka	4
Hobby Shops	135
Bubble Tea	184
Tanning	147
Propane	83
Handyman	87
Macarons	50
Japanese	2039
Convenience Stores	1340
Car Dealers	1013
Lawyers	422
IV Hydration	47
Rolfing	28
Falafel	19
Psychics	62
Tasting Classes	40

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!

```
%matplotlib plt
```

In [13]:

```
top_cat_df = business_df_grouped.count().orderBy('count', ascending=False)
top_cat_df.show(20)
```

category	count
Restaurants	36340
Food	22094
Shopping	20056
Restaurants	14423
Home Services	12001
Beauty & Spas	11633
Health & Medical	11390
Nightlife	9808
Local Services	9299
Bars	8914
Event Planning &...	7617
Food	7375
Active Life	7039
Automotive	6785
Shopping	6149
Coffee & Tea	5735
Sandwiches	5697
American (Tradit...	5235
Fashion	5231
Beauty & Spas	4941

only showing top 20 rows

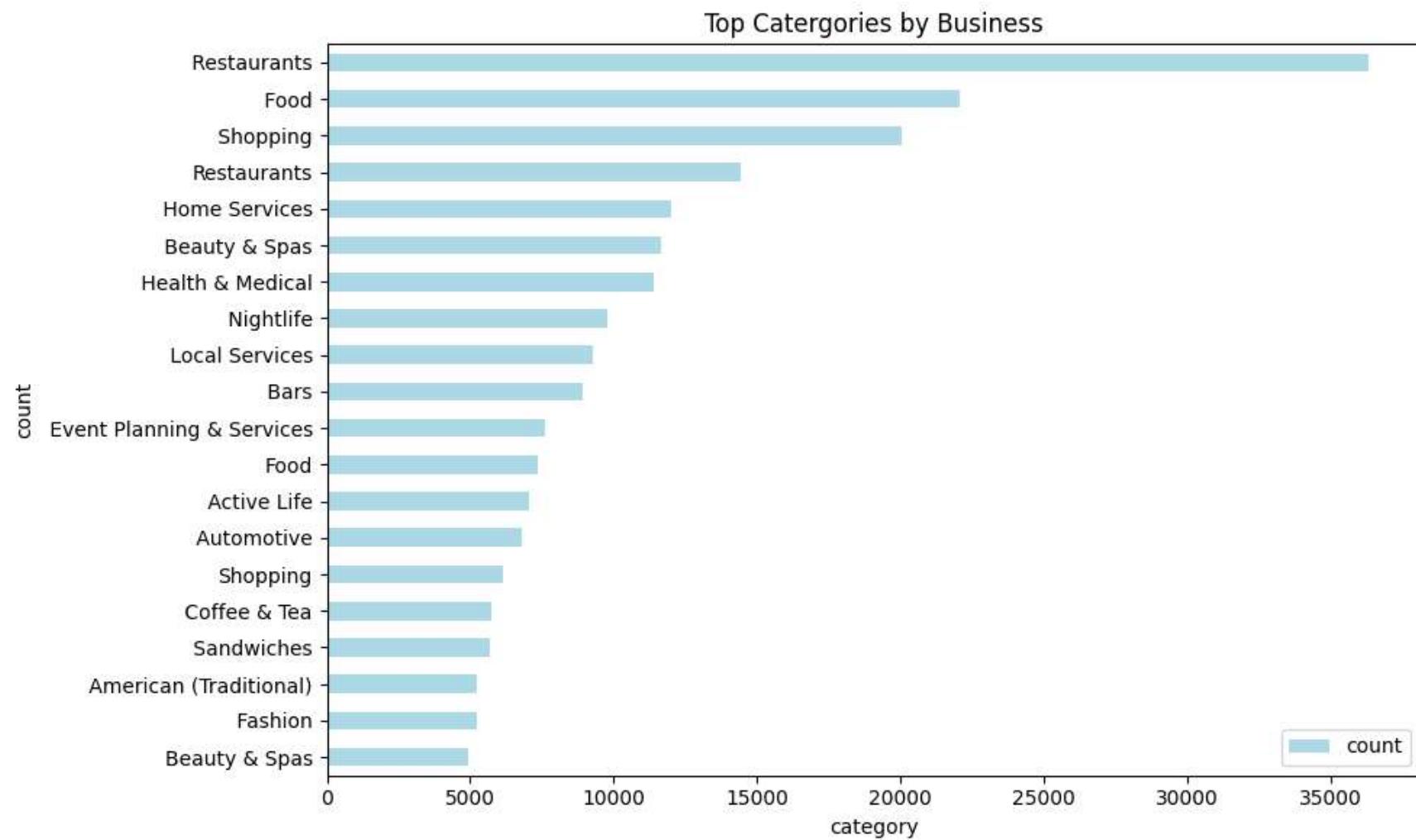
In [14]:

```
tcf = top_cat_df.toPandas().head(20)
tcf=tcf.sort_values("count")
```

In [15]:

```
plt.figure(figsize =(15,9))
ax=tcf.plot(kind='barh',
            x='category',
            figsize=(10,6),
            title="Top Categoricals by Business",color="lightblue")
ax.set_xlabel("category")
ax.set_ylabel("count")

plt.tight_layout()
%matplotlib 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 [16]:

```
user_df = spark.read.json('s3://sta9760-spark-project02-datasets/yelp_academic_dataset_user.json')
user_df.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)
```

In [17]:

```
review_df = spark.read.json('s3://sta9760-spark-project02-datasets/yelp_academic_dataset_review.json')
review_df.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 [18]:

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

business_id	stars
buF9druCkbuXLX526...	4.0
RA4V8pr014UyUbDvI...	4.0
_sS2LBIGNT5NQb6PD...	5.0
0AzLzHf0JgL7R0whd...	2.0
8zehGz9jnxPqXt0c7...	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 [19]:

```
avg_star=review_df \
    .where(col("text").isNotNull()) \
    .select("business_id", "stars") \
    .groupBy("business_id") \
    .agg(mean('stars').alias("avg(stars)")) \
    .show(5)
```

business_id	avg(stars)
yHtuNALYKtRZni080...	4.714285714285714
R0IJhEI-zSJpYT1YN...	3.606060606060606
uEUweopM30lHcVxj0...	3.0
L3WCfeVozu5etMhz4...	4.2
XzXcpPCb8Y5huk1EN...	4.666666666666667

only showing top 5 rows

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

In [20]:

```
join_df=business_df.join(avg_star,'business_id')
join_df.show(5)
```

business_id	latitude	longitude	address	name	attributes	categories	city	hours	is_o
pen				postal_code	review_count	stars	state		avg(stars)
--JuLhLvq3gyjNnXT...	111	Foster St, Ste 5 [,,,,,,	True,, T...	Beauty & Spas, Ha...	Peabody [9:0-17:0,, 9:0-1...				
1 42.5218228	-70.932435		CheraBella Salon	01960	12	5.0	MA		5.0
--_nBudP0b1lNRgKf...		1993 Hard Rd [,, u'full_bar', ...	Mexican, Gastropu...	Columbus [16:0-22:0,, 16:0...					
0 40.1172748	-83.0684812	Mezcal Cantina & ...		43235	8	4.0	OH		3.875
--ky0k0waSrCDlbSv...	6000 S Congress, ...	[,, u'none', {'ro...	Coffee & Tea, Res...	Austin [6:30-22:30, 6:30...					
0 30.200264	-97.7766101	Red Table Coffee		78745	30	4.0	TX	3.8666666666666667	
--z9usx6Fin8P_f0v...	7717 Southwest Pkwy	[,,,,,, False,...	Education, Presch...	Austin [7:30-16:0, 0:0-0...					
1 30.2562943	-97.8765403	WonderWell		78735	14	5.0	TX		5.0
-0qeY1293steyCqYh...	1 Avalon Dr [,,,,,, True,....	Real Estate, Home...	Wilmington [9:0-17:0, 9:30-1...						
1 42.580983	-71.158824	Avalon Oaks		01887	15	3.5	MA		3.375

only showing top 5 rows

Let's see a few of these:

```
In [21]: join_select=join_df.select("avg(stars)","stars","name","city","state")
join_select.show(5)
```

avg(stars)	stars	name	city	state
5.0	5.0	CheraBella Salon	Peabody	MA
3.875	4.0	Mezcal Cantina & ...	Columbus	OH
3.8666666666666667	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']
```

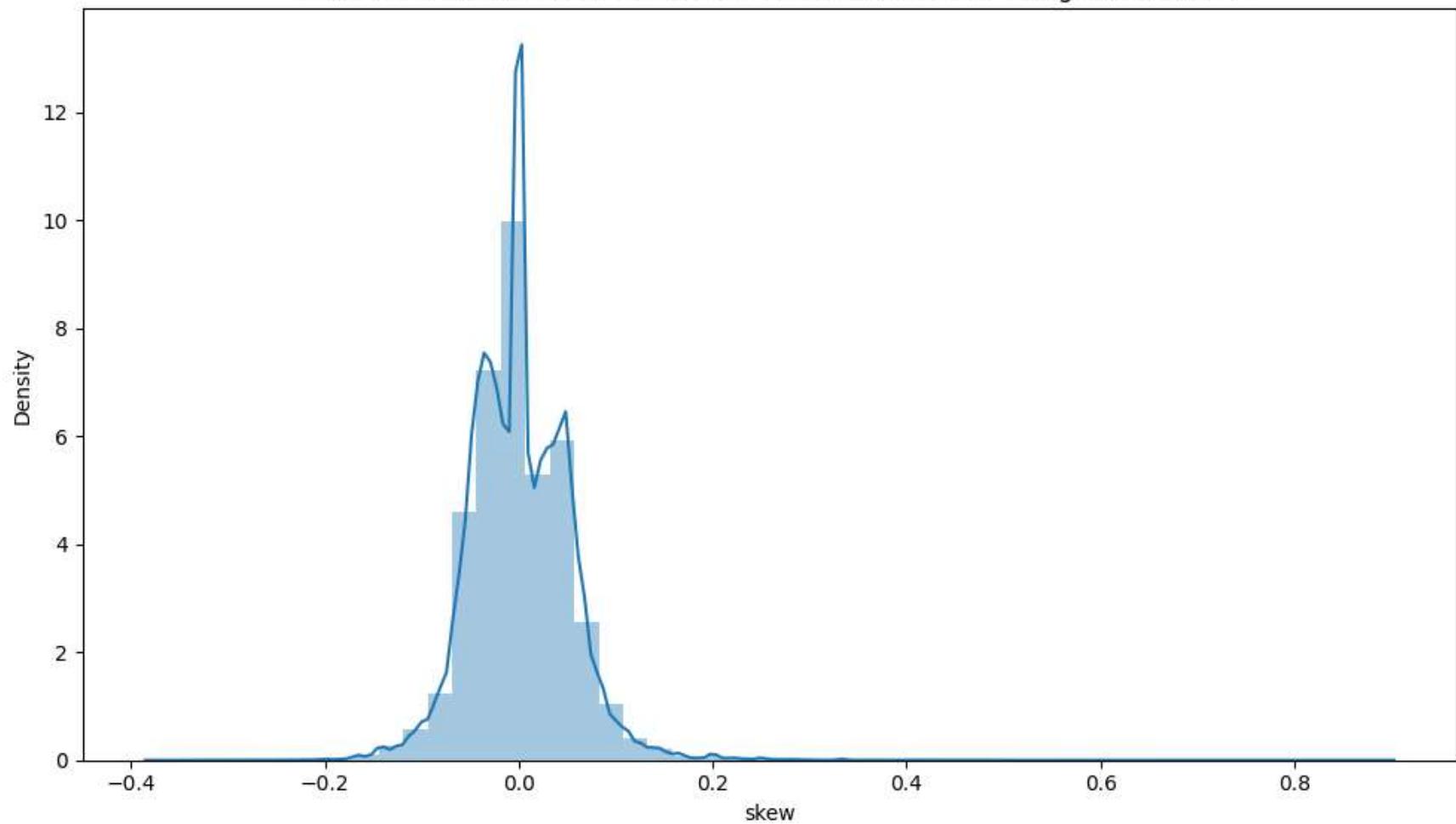
If the **skew** is negative, we can interpret that to be: reviewers who left a written response were more dissatisfied than normal. If **skew** is positive, we can interpret that to be: reviewers who left a written response were more satisfied than normal.

```
In [22]: skew_df = join_select.withColumn("skew", (join_select['avg(stars)'] - join_select['stars']) / join_select['stars'])
```

```
In [23]: skew_df=skew_df.toPandas()
```

```
In [24]: plt.figure(figsize =(10,6))
ax = sns.distplot(skew_df["skew"]).set_title('Skew between the stars from written reviews and actual rating of a business')
plt.tight_layout()
%matplotlib
```

Skew between the stars from written reviews and actual rating of a business



And finally, graph it!

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

According to the above graph, the review skew is positive. It means the reviewer who left a written response were inclined to give higher stars than the average rating.

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

Load the user data

In [25]:

```
user_df = spark.read.json('s3://sta9760-spark-project02-datasets/yelp_academic_dataset_user.json')
user_df.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)
```

Overview of Data

```
In [26]: user_df.show(1)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|average_stars|compliment_cool|compliment_cute|compliment_funny|compliment_hot|compliment_list|compliment_more|compliment_note|compliment_photos|compliment_plain|compliment_profile|compliment_writer|cool|elite|fans|
|friends|funny|name|review_count|useful|user_id|yelping_since|
+-----+-----+-----+-----+-----+-----+-----+-----+
| 3.85| 2541| 361| 2541| 1710| 147| 163|
|1212| 323| 5691| 190| 815|11291|2006,2007,2008,20...|1357|xBDpTUbai0DX
|rvxCe...|10030|Jane| 1220| 15038|q_QQ5kBBwlCcbL1s4...|2005-03-14 20:26:35|
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 1 row
```

Split the elite column

```
In [27]: user_df_elite = user_df \
    .select("user_id", "elite") \
    .withColumn('elite', explode(split('elite', ","))) \
```

```
In [28]: user_df_elite.select("elite").distinct().show()
```

```
+---+
|elite|
+---+
| 2014|
| 2018|
|
| 2010|
| 2009|
| 2006|
| 2015|
| 2011|
| 2008|
```

```
| 20  
| 2016  
| 2012  
| 2019  
| 2017  
| 2013  
| 2007  
+----+
```

Join the datasets

```
In [29]: user_review_join=review_df.join(user_df_elite,'user_id')  
user_review_star=user_review_join.select("elite","stars","business_id")
```

```
In [ ]: user_review_star.show(5)
```

```
In [ ]: elite_star=user_review_star \  
          .where(col("elite")!="") \  
          .select("business_id","stars") \  
          .groupBy("business_id") \  
          .agg(mean('stars').alias("elite_stars"))  
elite_star.show(5)
```

```
In [ ]: non_elite_star=user_review_star \  
          .where(col("elite")== "") \  
          .select("business_id","stars") \  
          .groupBy("business_id") \  
          .agg(mean('stars').alias("non_elite_stars")) \  
          .sort(col("non_elite_stars").desc())  
non_elite_star.show(5)
```

```
In [ ]: elite_join=business_df.join(elite_star,'business_id')  
elite_non_elite_join=elite_join.join(non_elite_star,"business_id")
```

Boxplotting graph

```
In [ ]: elite_non_elite_join.show(5)
```

```
In [ ]: elite_df_pd=elite_non_elite_join.toPandas()
```

```
In [ ]: plt.figure(figsize =(10,6))
df = elite_df_pd[["stars", "elite_stars","non_elite_stars"]]
df.boxplot()
plt.title("Boxplot of ratings from regular/elite/non-elite users ")
plt.tight_layout()
%matplotlib plt
```

According to the boxplot we create, we can see comparing to the users who are not elite, the elite clients are inclined to give higher rating to the business. Both IQR and median rating of elite users are higher than the non elite users.

Which category of business is most/least popular?

Date filtering

```
In [ ]: business_df_copy = business_df \
    .withColumn('categories', explode(split('categories', ','))) \
    .withColumnRenamed('categories', 'category')
```

```
In [ ]: business_df_copy.show(5)
```

Categories with most review counts

```
In [ ]: top_review_counts_df=business_df_copy \
    .select("category", "review_count") \
    .groupby("category")
top_review_counts_df=top_review_counts_df.count().orderBy('count', ascending=False)
top_review_counts_df.show()
```

```
In [ ]: top_review_counts_pd=top_review_counts_df.toPandas().head(20)
```

```
In [ ]: plt.figure(figsize =(15,9))
ax=top_review_counts_pd.plot(kind='bar',
    x='category',
    figsize=(10,6),
    title="Categories by review counts ",color="lightblue")
ax.set_xlabel("category")
ax.set_ylabel("count")

plt.tight_layout()
%matplotlib plt
```

Categories with lowest ratings

```
In [ ]: top_ratings_df=business_df_copy \
         .select("category","stars") \
         .groupby("category") \
         .agg(mean('stars').alias("avg_stars"))
top_ratings_df=top_ratings_df.orderBy('avg_stars',ascending=False)
top_ratings_df.show(40)
```

```
In [ ]: top_ratings_pd=top_ratings_df.toPandas().tail(20)
```

```
In [ ]: plt.figure(figsize =(15,9))
ax=top_ratings_pd.plot(kind='barh',
    x='category',
    figsize=(10,6),
    title="Categories by ratings ",color="lightblue")
ax.set_xlabel("category")
ax.set_ylabel("rating")

plt.tight_layout()
%matplotlib plt
```