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://sta9760yelp-dataset/yelp/*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]:
```
%%info
```

Current session configs: `{'conf': {'spark.pyspark.python': 'python3', 'spark.pyspark.virtualenv.enabled': 'true', 'spark.pyspark.virtualenv.type': 'native', 'spark.pyspark.virtualenv.bin.path': '/usr/bin/virtualenv'}, 'kind': 'pyspark'}`

No active sessions.

In [2]:
```
sc.list_packages()
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
2	application_1606272122468_0003	pyspark	idle	Link	Link	✔

```
SparkSession available as 'spark'.

Package                    Version
-------------------------- ---------
beautifulsoup4             4.9.1
boto                       2.49.0
click                      7.1.2
jmespath                   0.10.0
joblib                     0.16.0
lxml                       4.5.2
mysqlclient                1.4.2
nltk                       3.5
nose                       1.3.4
numpy                      1.16.5
pip                        9.0.1
py-dateutil                2.2
python37-sagemaker-pyspark 1.4.0
pytz                       2020.1
PyYAML                     5.3.1
regex                      2020.7.14
setuptools                 28.8.0
six                        1.13.0
soupsieve                  1.9.5
tqdm                       4.48.2
wheel                      0.29.0
windmill                   1.6
```

In [3]:
```
sc.install_pypi_package("pandas==1.0.3")
```

```
Collecting pandas==1.0.3
  Using cached https://files.pythonhosted.org/packages/4a/6a/94b219b8ea0f2d580169e85ed1e
dc0163743f55aaeca8a44c2e8fc1e344e/pandas-1.0.3-cp37-cp37m-manylinux1_x86_64.whl
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/site-packages (f
rom 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)
  Using cached https://files.pythonhosted.org/packages/d4/70/d60450c3dd48ef87586924207ae
8907090de0b306af2bce5d134d78615cb/python_dateutil-2.8.1-py2.py3-none-any.whl
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
```

In [4]:
```
sc.install_pypi_package("matplotlib==3.2.1")
```

```
Collecting matplotlib==3.2.1
  Using cached https://files.pythonhosted.org/packages/b2/c2/71fcf957710f3ba1f09088b3577
6a799ba7dd95f7c2b195ec800933b276b/matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl
Requirement already satisfied: python-dateutil>=2.1 in /mnt/tmp/1606285941388-0/lib/pyth
on3.7/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)
  Using cached https://files.pythonhosted.org/packages/8a/bb/488841f56197b13700afd5658fc
279a2025a39e22449b7cf29864669b15d/pyparsing-2.4.7-py2.py3-none-any.whl
Collecting cycler>=0.10 (from matplotlib==3.2.1)
  Using cached https://files.pythonhosted.org/packages/f7/d2/e07d3ebb2bd7af696440ce7e754
c59dd546ffe1bbe732c8ab68b9c834e61/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)
  Using cached https://files.pythonhosted.org/packages/d2/46/231de802ade4225b76b96cffe41
9cf3ce52bbe92e3b092cf12db7d11c207/kiwisolver-1.3.1-cp37-cp37m-manylinux1_x86_64.whl
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
```

In [5]:
```
sc.install_pypi_package("seaborn")
```

```
Collecting seaborn
  Using cached https://files.pythonhosted.org/packages/bc/45/5118a05b0d61173e6eb12bc5804
f0fbb6f196adb0a20e0b16efc2b8e98be/seaborn-0.11.0-py3-none-any.whl
Requirement already satisfied: numpy>=1.15 in /usr/local/lib64/python3.7/site-packages
(from seaborn)
Collecting scipy>=1.0 (from seaborn)
  Using cached https://files.pythonhosted.org/packages/dc/7e/8f6a79b102ca1ea928bae8998b0
5bf5dc24a90571db13cd119f275ba6252/scipy-1.5.4-cp37-cp37m-manylinux1_x86_64.whl
Requirement already satisfied: matplotlib>=2.2 in /mnt/tmp/1606285941388-0/lib/python3.
7/site-packages (from seaborn)
Requirement already satisfied: pandas>=0.23 in /mnt/tmp/1606285941388-0/lib/python3.7/si
te-packages (from seaborn)
Requirement already satisfied: python-dateutil>=2.1 in /mnt/tmp/1606285941388-0/lib/pyth
on3.7/site-packages (from matplotlib>=2.2->seaborn)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /mnt/tmp/1606
285941388-0/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn)
Requirement already satisfied: cycler>=0.10 in /mnt/tmp/1606285941388-0/lib/python3.7/si
te-packages (from matplotlib>=2.2->seaborn)
Requirement already satisfied: kiwisolver>=1.0.1 in /mnt/tmp/1606285941388-0/lib/python
3.7/site-packages (from matplotlib>=2.2->seaborn)
```

```
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/site-packages (f
rom pandas>=0.23->seaborn)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from
python-dateutil>=2.1->matplotlib>=2.2->seaborn)
Installing collected packages: scipy, seaborn
Successfully installed scipy-1.5.4 seaborn-0.11.0
```

Importing

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

In [6]:
```
import matplotlib.pyplot as plt
import seaborn
```

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 [7]:
```
df = spark.read.json('s3://sta9760yelp-dataset/yelp/*business.json')
```

Overview of Data

Display the number of rows and columns in our dataset.

In [8]:
```
print(f'Total Columns: {len(df.dtypes)}')
print(f'Total Rows: {df.count():,}')
```

```
Total Columns: 14
Total Rows: 209,393
```

Display the DataFrame schema below.

In [9]:
```
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)
 |    |-- 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 [10]:
```
df.select('business_id', 'name', 'city', 'state', 'categories').show(5)
```

```
+--------------------+--------------------+---------------+-----+--------------------+
|         business_id|                name|           city|state|          categories|
+--------------------+--------------------+---------------+-----+--------------------+
|f9NumwFMBDn751xgF...|The Range At Lake...|      Cornelius|   NC|Active Life, Gun/...|
|Yzvjg0SayhoZgCljU...|   Carlos Santo, NMD|     Scottsdale|   AZ|Health & Medical,...|
|XNoUzKckATkOD1hP6...|             Felinus|       Montreal|   QC|Pets, Pet Service...|
|6OAZjbxqM5ol29BuH...|Nevada House of Hose|North Las Vegas|   NV|Hardware Stores, ...|
```

```
|51M2Kk903DFYI6gnB...|USE MY GUY SERVIC...|          Mesa|   AZ|Home Services, Pl...|
+--------------------+--------------------+--------------+-----+--------------------+
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 [11]:
```
from pyspark.sql.functions import explode, split
df_exploded = df.withColumn('category', explode(split('categories', ', ')))
```

Display the first 5 rows of your association table below.

In [12]:
```
df_exploded.select('business_id', 'category').show(5)
```

```
+--------------------+----------------+
|         business_id|        category|
+--------------------+----------------+
|f9NumwFMBDn751xgF...|     Active Life|
|f9NumwFMBDn751xgF...|Gun/Rifle Ranges|
```

```
|f9NumwFMBDn751xgF...|     Guns & Ammo|
|f9NumwFMBDn751xgF...|        Shopping|
|Yzvjg0SayhoZgCljU...|Health & Medical|
+--------------------+----------------+
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 [13]:
```
df_exploded.select('category').distinct().count()
```

```
1336
```

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 [14]:
```
df_exploded.groupby('category').count().show(20)
```

```
+--------------------+-----+
|            category|count|
+--------------------+-----+
|       Dermatologists|  341|
|       Paddleboarding|   36|
|         Aerial Tours|   28|
|          Hobby Shops|  828|
|           Bubble Tea|  720|
|              Embassy|   13|
|             Handyman|  682|
|              Tanning|  938|
|       Aerial Fitness|   29|
|              Tempura|    1|
|              Falafel|  159|
|        Outlet Stores|  399|
```

```
|        Summer Camps|  318|
|     Clothing Rental|   55|
|      Sporting Goods| 2311|
|     Cooking Schools|  118|
|  College Counseling|   15|
|  Lactation Services|   50|
|Ski & Snowboard S...|   50|
|             Museums|  359|
+--------------------+-----+
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!

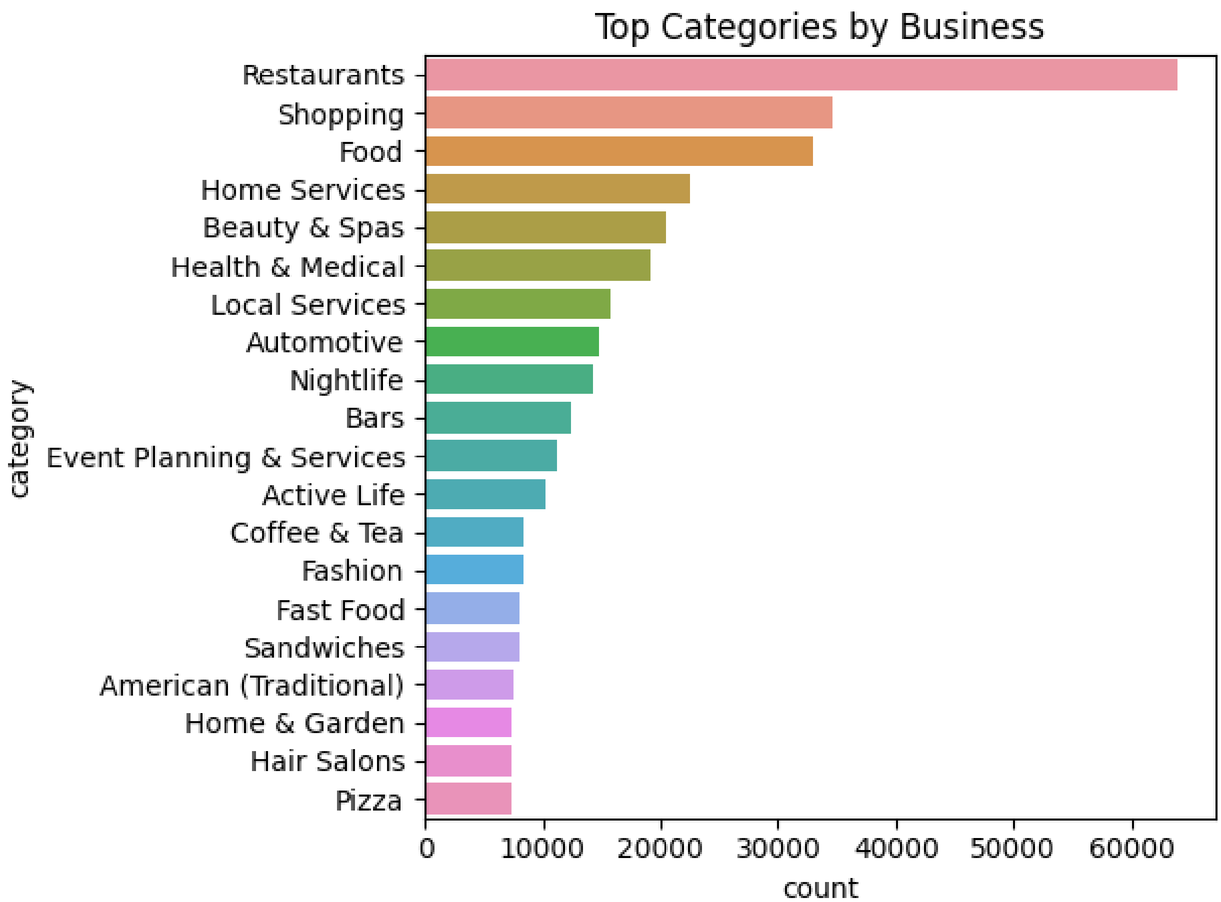
```
%matplot plt
```

In [15]:

```
dfx = df_exploded.groupby('category').count().toPandas()
```

In [16]:

```
plt.figure()
seaborn.barplot(x = 'count', y = 'category', data = dfx.sort_values('count', ascending
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 [17]:
```
dfreview = spark.read.json('s3://sta9760yelp-dataset/yelp/*review.json')
```

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

In [18]:
```
print(f'Total Columns: {len(dfreview.dtypes)}')
print(f'Total Rows: {dfreview.count():,}')
dfreview.printSchema()
```

```
Total Columns: 9
Total Rows: 8,021,122
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 [19]:
```
dfreview.select('business_id', 'stars').show(5)
```

```
+--------------------+-----+
|         business_id|stars|
+--------------------+-----+
|-MhfebM0QIsKt87iD...|  2.0|
|lbrU8StCq3yDfr-QM...|  1.0|
|HQl28KMwrEKHqhFrr...|  5.0|
|5JxlZaqCnk1MnbgRi...|  1.0|
|IS4cv902ykd8wj1TR...|  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 [20]:
```
from pyspark.sql.functions import mean, length
dfravg = dfreview.filter(length('text') > 0).groupby('business_id').agg(mean('stars').a
```

In [21]:
```
dfravg.select('business_id', 'avg(stars)').show(5)
```

```
+--------------------+------------------+
|         business_id|        avg(stars)|
+--------------------+------------------+
|RMjCnixEY5i12Ciqn...|3.5316455696202533|
|VHsNB3pdGVcRgs6C3...| 3.411764705882353|
|kpbhERZoj1eTDRnMV...| 2.033333333333333|
|ipFreSFhjClfNETuM...|               2.6|
|9A_mB7Ez3RIh26EN5...|               2.6|
+--------------------+------------------+
only showing top 5 rows
```

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

In [22]:
```
dfjoin = df.join(dfravg, on = 'business_id', how = 'inner')
```

Let's see a few of these:

In [23]:
```
dfjoin.select('avg(stars)', 'stars', 'name', 'city', 'state').show(5)
```

```
+----------------+-----+-------------------+---------+-----+
|      avg(stars)|stars|               name|     city|state|
+----------------+-----+-------------------+---------+-----+
|4.11784140969163|  4.0|Delmonico Steakhouse|Las Vegas|   NV|
```

```
|             4.5|  4.5|Mr. Pancho Mexica...|     Mesa|   AZ|
|            3.75|  4.0|Maricopa County D...|  Phoenix|   AZ|
|             4.0|  4.0|Double Play Sport...|Las Vegas|   NV|
|          2.6875|  2.5|  Impressions Dental| Chandler|   AZ|
+----------------+-----+--------------------+---------+-----+
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']
```

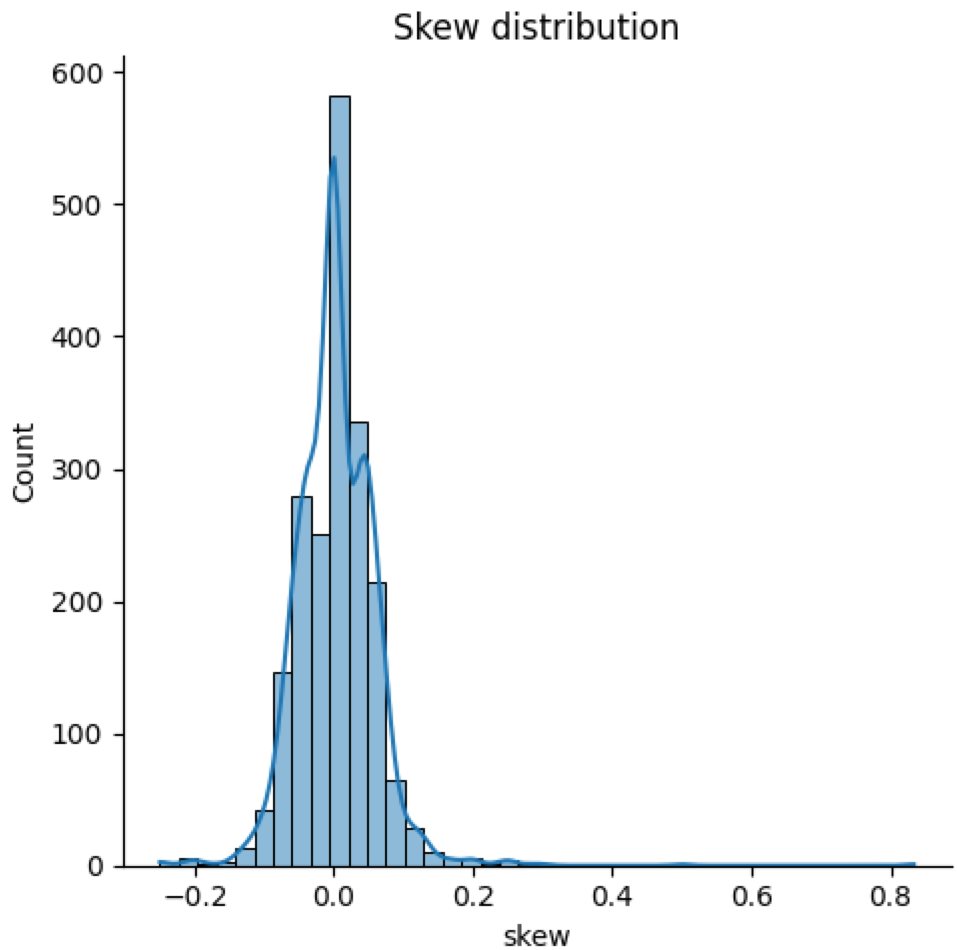
If the **skew** is negative, we can interpret that to be: reviewers who left a written response were more dissatisfied than normal. If **skew** is positive, we can interpret that to be: reviewers who left a written response were more satisfied than normal.

In [24]:
```
from pyspark.sql.functions import col
dfjoin = dfjoin.withColumn('skew', (col('avg(stars)') - col('stars')) / col('stars'))
```

And finally, graph it!

In [25]:
```
dfx = dfjoin.toPandas()
```

In [35]:
```
ax = plt.figure()
seaborn.displot(dfx.sample(n = 2000), x = 'skew', kde = True, bins = 40)
plt.title('Skew distribution')
plt.tight_layout()
%matplot plt
```



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

From the distribution graph, we find Yelp writtern reviews are slight skew positive. (the very high bar which is positive)

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 [27]:
```
dfu = spark.read.json('s3://sta9760yelp-dataset/yelp/*user.json')
```

Overview of the data

In [28]:
```
print(f'Total Columns: {len(dfu.dtypes)}')
print(f'Total Rows: {dfu.count():,}')
```

```
Total Columns: 22
Total Rows: 1,968,703
```

In [29]:
```
dfreview.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)
```

Keep reviews from elite users only

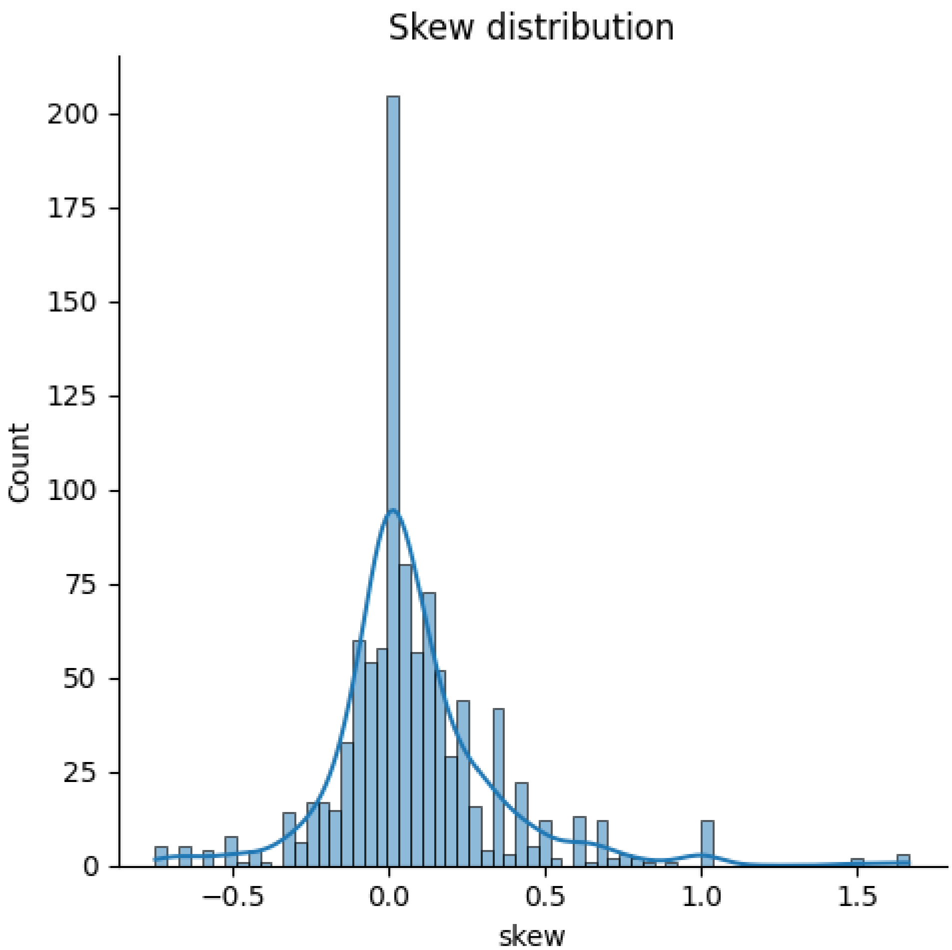
In [30]:
```
from pyspark.sql.functions import length
dfj1 = (dfreview
        .select('business_id', 'review_id', 'user_id', 'stars')
        .join(dfu.filter(length('elite') > 0).select('user_id', 'elite'), on = 'user_id
        .groupby('business_id').agg(mean('stars').alias('avg(stars)'))
       )
```

Calculate 'skew' for reviews from elite users only

In [31]:
```
dfj2 = df.join(dfj1, on = 'business_id', how = 'inner')
dfj2 = dfj2.withColumn('skew', (col('avg(stars)') - col('stars')) / col('stars'))
dfx2 = dfj2.toPandas()
```

Plot the elite skew

In [34]:
```
plt.figure()
seaborn.displot(dfx2.sample(n = 1000), x = 'skew', kde = True)
plt.title('Skew distribution')
plt.tight_layout()
%matplot plt
```



In [33]:
```
dfx2[['skew']].describe()
```

	skew
count	148225.000000
mean	0.088309
std	0.291314
min	-0.800000
25%	-0.037037
50%	0.027778
75%	0.166667
max	4.000000

Similar to full reviews, elite skews are postively skewed. However, the t-stat of skew is 0.08 / 0.29 = 0.30. The biases is not significant.

In [ ]: