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

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]:

```
# Current available packages
sc.list_packages()
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
2	application_1638321780052_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
```

Installation and Initial Setup

In [3]:

```
sc.install_pypi_package("pandas==1.0.3")
sc.install_pypi_package("matplotlib==3.2.1")
sc.install_pypi_package("scipy==1.7.1")
sc.install_pypi_package("seaborn==0.10.0")
```

```
Collecting pandas==1.0.3
  Using cached https://files.pythonhosted.org/packages/4a/6a/94b219b8ea0f2d580169e85ed1edc0163743f55aaeca8a44c2
e8fc1e344e/pandas-1.0.3-cp37-cp37m-manylinux1_x86_64.whl
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)
  Using cached https://files.pythonhosted.org/packages/36/7a/87837f39d0296e723bb9b62bbb257d0355c7f6128853c78955
f57342a56d/python_dateutil-2.8.2-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-2.8.2

Collecting matplotlib==3.2.1
  Using cached https://files.pythonhosted.org/packages/b2/c2/71fcf957710f3ba1f09088b35776a799ba7dd95f7c2b195ec8
00933b276b/matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl
Requirement already satisfied: python-dateutil>=2.1 in /mnt/tmp/1638322654330-0/lib/python3.7/site-packages (fr
om 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/a0/34/895006117f6fce0b4de045c87e154ee4a20c68ec0a4c9a36d9
00888fb6bc/pyparsing-3.0.6-py3-none-any.whl
Collecting cycler>=0.10 (from matplotlib==3.2.1)
  Using cached https://files.pythonhosted.org/packages/5c/f9/695d6bedebd747e5eb0fe8fad57b72fdf25411273a39791cde
838d5a8f51/cycler-0.11.0-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/09/6b/6e567cb2e86d4e5939a9233f8734e26021b6a9c1bc4b1edccb
a236a84cc2/kiwisolver-1.3.2-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl
```

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-packages (from python-dateutil>=2.1->m
atplotlib==3.2.1)
Installing collected packages: pyparsing, cycler, kiwisolver, matplotlib
Successfully installed cycler-0.11.0 kiwisolver-1.3.2 matplotlib-3.2.1 pyparsing-3.0.6

Collecting scipy==1.7.1
  Using cached https://files.pythonhosted.org/packages/b5/6b/8bc0b61ebf824f8c3979a31368bbe38dd247590049a994ab0e
d077cb56dc/scipy-1.7.1-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl
Requirement already satisfied: numpy<1.23.0,>=1.16.5 in /usr/local/lib64/python3.7/site-packages (from scipy==
1.7.1)
Installing collected packages: scipy
Successfully installed scipy-1.7.1

Collecting seaborn==0.10.0
  Using cached https://files.pythonhosted.org/packages/70/bd/5e6bf595fe6ee0f257ae49336dd180768c1ed3d7c7155b2fdf
894c1c808a/seaborn-0.10.0-py3-none-any.whl
Requirement already satisfied: pandas>=0.22.0 in /mnt/tmp/1638322654330-0/lib/python3.7/site-packages (from sea
born==0.10.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib64/python3.7/site-packages (from seaborn==0.10.0)
Requirement already satisfied: scipy>=1.0.1 in /mnt/tmp/1638322654330-0/lib/python3.7/site-packages (from seabo
rn==0.10.0)
Requirement already satisfied: matplotlib>=2.1.2 in /mnt/tmp/1638322654330-0/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->sea
born==0.10.0)
Requirement already satisfied: python-dateutil>=2.6.1 in /mnt/tmp/1638322654330-0/lib/python3.7/site-packages
(from pandas>=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/1638322654330-0/lib/python
3.7/site-packages (from matplotlib>=2.1.2->seaborn==0.10.0)
Requirement already satisfied: cycler>=0.10 in /mnt/tmp/1638322654330-0/lib/python3.7/site-packages (from matpl
otlib>=2.1.2->seaborn==0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /mnt/tmp/1638322654330-0/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: seaborn
Successfully installed seaborn-0.10.0
```

In [4]:

```python
# Check available pakcages again
sc.list_packages()
```

```
Package                    Version
-------------------------- ---------
```

```
beautifulsoup4             4.9.1
boto                       2.49.0
click                      7.1.2
cycler                     0.11.0
jmespath                   0.10.0
joblib                     0.16.0
kiwisolver                 1.3.2
lxml                       4.5.2
matplotlib                 3.2.1
mysqlclient                1.4.2
nltk                       3.5
nose                       1.3.4
numpy                      1.16.5
pandas                     1.0.3
pip                        9.0.1
py-dateutil                2.2
pyparsing                  3.0.6
python-dateutil            2.8.2
python37-sagemaker-pyspark 1.4.0
pytz                       2020.1
PyYAML                     5.3.1
regex                      2020.7.14
scipy                      1.7.1
seaborn                    0.10.0
setuptools                 28.8.0
six                        1.13.0
soupsieve                  1.9.5
tqdm                       4.48.2
wheel                      0.29.0
windmill                   1.6
```

Importing

Now, import the installed packages from the previous block below

In [25]:

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

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 [6]:
```
df_business = spark.read.json('s3://yelpreveiwsdataset/yelp/yelp_academic_dataset_business.json')
```

Overview of Data

Display the number of rows and columns in our dataset.

In [7]:
```
# get an overview of the dataframe
print(f'Total Columns: {len(df_business.dtypes)}')
print(f'Total Rows: {df_business.count():,}')
```

```
Total Columns: 14
Total Rows: 160,585
```

Display the DataFrame schema below.

In [8]:
```
# schema
df_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 [9]:

```
df_business.createOrReplaceTempView("bus")
df_table = spark.sql(
'''
SELECT business_id, name, city, state, stars, categories
FROM bus
'''
)

df_table.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 [10]:
```
from pyspark.sql.functions import split, explode
df_cat = df_business.withColumn('category',explode(split('categories',', ')))
```

In [11]:
```
df_cat.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
```

In [13]:

```
df_cat.createOrReplaceTempView("cat")
df_table = spark.sql(
'''
SELECT business_id, category
FROM cat
'''
)

df_table.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 [14]:

```
unique = spark.sql(
'''
SELECT DISTINCT(category)
FROM cat
''')
```

```
unique.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 [15]:
```
top_20 = spark.sql(
'''
SELECT category, COUNT(*) AS total
FROM cat
GROUP BY category
ORDER BY total DESC
LIMIT 20''')

top_20.show()
```

```
+--------------------+-----+
|            category|total|
+--------------------+-----+
|         Restaurants|50763|
|                Food|29469|
```

```
|            Shopping|26205|
|        Beauty & Spas|16574|
|        Home Services|16465|
|     Health & Medical|15102|
|       Local Services|12192|
|            Nightlife|11990|
|                 Bars|10741|
|           Automotive|10119|
|Event Planning & ...| 9644|
|          Active Life| 9231|
|         Coffee & Tea| 7725|
|           Sandwiches| 7272|
|              Fashion| 6599|
|American (Traditi...| 6541|
|          Hair Salons| 5900|
|                Pizza| 5756|
|      Hotels & Travel| 5703|
|   Breakfast & Brunch| 5505|
+--------------------+-----+
```

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
```

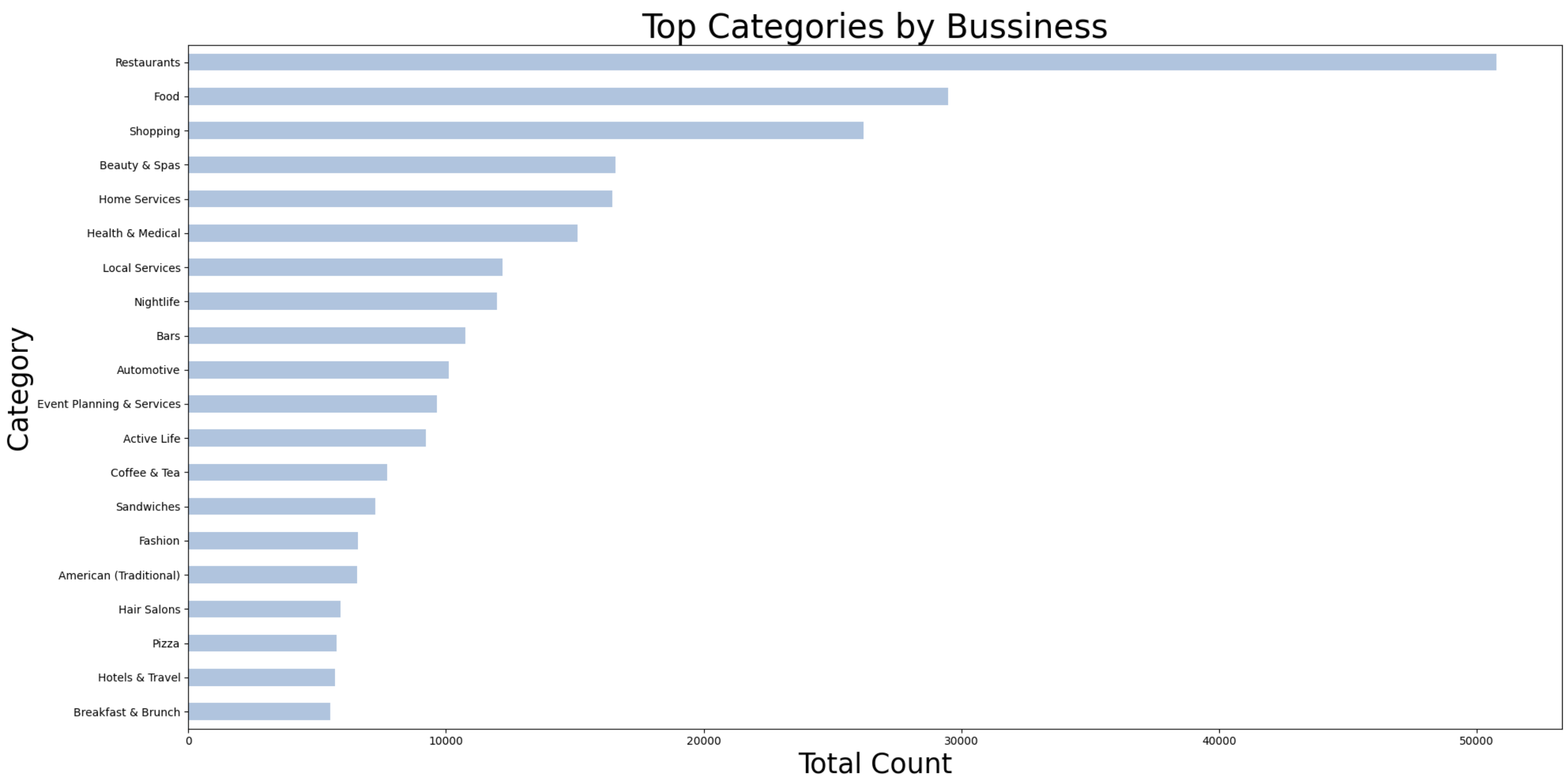
In [26]:

```
top_20_count = top_20.toPandas()
```

In [27]:

```
# plot
top_20_count.sort_values(by='total',ascending=True).plot(
x='category',
y='total',
kind='barh',
figsize=(20,10),
legend=None,
color='lightsteelblue')
plt.title('Top Categories by Bussiness', fontsize=30)
plt.xlabel("Total Count",fontsize=25)
```

```
plt.ylabel("Category",fontsize=25)
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]:
```
df_reviews = spark.read.json('s3://yelppreveiwsdataset/yelp/yelp_academic_dataset_review.json')
```

In [18]:
```
df_reviews.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 [19]:
```
df_reviews.createOrReplaceTempView("review")
df_table_review = spark.sql(
'''
SELECT business_id, stars
FROM review
'''
)

df_table_review.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 [20]:
```
avg_stars_per_business = spark.sql(
'''
SELECT DISTINCT business_id, AVG(stars) as average_star
FROM review
GROUP BY business_id
'''
)
avg_stars_per_business.createOrReplaceTempView("avg")
avg_stars_per_business.show(5)
```

```
+--------------------+-----------------+
|         business_id|     average_star|
+--------------------+-----------------+
|yHtuNAlYKtRZniO8O...|4.714285714285714|
|R0IJhEI-zSJpYT1YN...|3.606060606060606|
|uEUweopM30lHcVxjO...|              3.0|
|L3WCfeVozu5etMhz4...|              4.2|
|XzXcpPCb8Y5huklEN...|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 [21]:
```
reviews_business = spark.sql(
'''
SELECT avg.average_star, bus.stars, bus.name, bus.city, bus.state
FROM avg
INNER JOIN bus ON avg.business_id = bus.business_id
ORDER BY avg.average_star DESC
''')
reviews_business.createOrReplaceTempView("combined")
reviews_business.show(5)
```

```
+------------+-----+--------------------+---------+-----+
|average_star|stars|                name|     city|state|
```

```
+------------+-----+--------------------+---------+-----+
|         5.0|  5.0|Xtreme Gracie Jiu...| Minneola|   FL|
|         5.0|  5.0|East Van Wear & T...|Vancouver|   BC|
|         5.0|  5.0|Orlando Health Im...|  Orlando|   FL|
|         5.0|  5.0|   Big House Guitars|  Decatur|   GA|
|         5.0|  5.0|  Holy Rosary Church| Portland|   OR|
+------------+-----+--------------------+---------+-----+
only showing top 5 rows
```

Let's see a few of these:

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]:

```
# calculate skew
skew = spark.sql(
'''
SELECT (average_star - stars) / stars AS skew
FROM combined
ORDER BY skew ASC
''')
skew.show()
```

```
+--------------------+
|                skew|
+--------------------+
| -0.3714285714285714|
|               -0.35|
| -0.3090909090909091|
|-0.30681818181818177|
| -0.3061224489795919|
| -0.2962962962962963|
|-0.28846153846153844|
```

```
|            -0.265625|
|-0.25925925925925924|
| -0.2533333333333333|
|-0.25274725274725274|
|                -0.25|
|                -0.25|
|                -0.25|
|                -0.25|
|                -0.25|
|-0.24444444444444446|
|-0.24444444444444446|
|-0.23809523809523814|
|-0.23809523809523814|
+--------------------+
only showing top 20 rows
```

In [23]:

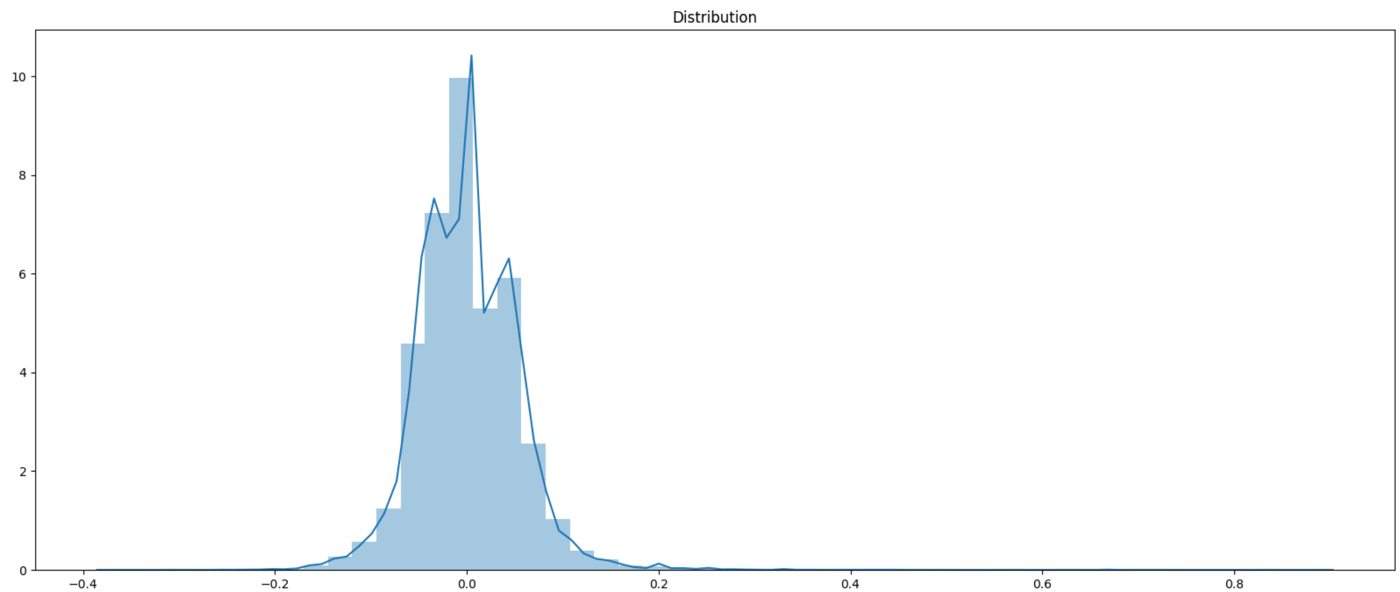
```
# convert to pandas
skew_pandas = skew.toPandas()
```

And finally, graph it!

In [33]:

```
fig, ax = plt.subplots(figsize = (20,8))
skew_plot = sns.distplot(skew_pandas)
ax.set_title('Distribution')

%matplot plt
```



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

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 [34]:
```
df_user = spark.read.json('s3://yelpreveiwsdataset/yelp/yelp_academic_dataset_user.json')
```

In [36]:

```
print(f'Total Columns: {len(df_user.dtypes)}')
print(f'Total Rows: {df_user.count():,}')
df_user.printSchema()
```

```
Total Columns: 22
Total Rows: 2,189,457
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 [37]:

```
print(f'Total Columns: {len(df_reviews.dtypes)}')
print(f'Total Rows: {df_reviews.count():,}')
df_reviews.printSchema()
```

```
Total Columns: 9
Total Rows: 8,635,403
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)
```

In [38]:

```
df_user.createOrReplaceTempView("user")
test = spark.sql(
'''
SELECT user_id, average_stars, useful, elite, yelping_since
FROM user
''')
test.show(5)
```

```
+--------------------+-------------+------+--------------------+-------------------+
|             user_id|average_stars|useful|               elite|      yelping_since|
+--------------------+-------------+------+--------------------+-------------------+
|q_QQ5kBBwlCcbL1s4...|         3.85| 15038|2006,2007,2008,20...|2005-03-14 20:26:35|
|dIIKEfOgo0KqUfGQv...|         4.09| 21272|2007,2008,2009,20...|2007-08-10 19:01:51|
|D6ErcUnFALnCQN4b1...|         3.76|   188|           2010,2011|2007-02-07 15:47:53|
|JnPIjvC0cmooNDfsa...|         3.77|  7234|2009,2010,2011,20...|2009-02-09 16:14:29|
|37Hc8hr3cw0iHLoPz...|         3.72|  1577|      2009,2010,2011|2008-03-03 04:57:05|
+--------------------+-------------+------+--------------------+-------------------+
only showing top 5 rows
```

Joining **Users** to **reviews** dataset

In [39]:

```
user_review = spark.sql(
'''
SELECT user.average_stars,user.elite,user.yelping_since, review.stars,review.business_id
FROM user
INNER JOIN review ON user.user_id = review.user_id
''')
user_review.createOrReplaceTempView("user_business")
user_review.show(5)
```

```
+-------------+-----+-------------------+-----+--------------------+
|average_stars|elite|      yelping_since|stars|         business_id|
+-------------+-----+-------------------+-----+--------------------+
|         2.62|     |2009-08-31 19:55:54|  5.0|GgR7kcKykuqXB11fW...|
```

```
|         3.67|     |2015-03-21 18:51:08|  5.0|rxNfidGLHtMYyLNeo...|
|         2.73|     |2013-09-11 04:19:10|  2.0|2OaX6XjAoI7VD6jLd...|
|         2.73|     |2013-09-11 04:19:10|  1.0|IfOj3AxPl3Exsd_Yl...|
|         2.73|     |2013-09-11 04:19:10|  2.0|bAuYOa-VuqTOnKzWN...|
+-------------+-----+-------------------+-----+--------------------+
only showing top 5 rows
```

In [40]:

```
df_business.createOrReplaceTempView("bus")
df_table = spark.sql(
'''
SELECT business_id, name, city, state, stars, categories
FROM bus
'''
)

df_table.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
```

Joining User Review and Business

In [41]:

```
user_review_bus = spark.sql(
'''
SELECT user_business.average_stars,bus.state
FROM user_business
INNER JOIN bus ON user_business.business_id = bus.business_id
''')
user_review_bus.createOrReplaceTempView("user_review_bus")
user_review_bus.show(5)
```

```
+-------------+-----+
|average_stars|state|
```

```
+-------------+-----+
|         2.62|   GA|
|         3.67|   FL|
|         2.73|   CO|
|         2.73|   CO|
|         2.73|   CO|
+-------------+-----+
only showing top 5 rows
```

Average stars by state

In [48]:

```
user_state = user_review_bus.select('average_stars','state').groupby('state').count()
user_state.show(5)
```

```
+-----+-----+
|state|count|
+-----+-----+
|   MN|    6|
|   DC|   10|
|   DE|    7|
|   IL|   85|
|   HI|    7|
+-----+-----+
only showing top 5 rows
```

In [57]:

```
# user_stars = user_state.toPandas()
# user_stars = user_stars.sort_values('count',ascending=False).head(10)

user_review_bus.createOrReplaceTempView("user_star")
user_str = spark.sql(
'''
SELECT state, AVG(average_stars) as avg_stars
FROM user_star
GROUP BY state
ORDER BY avg_stars DESC
'''
)

user_str.show(5)
```

```
+-----+------------------+
```

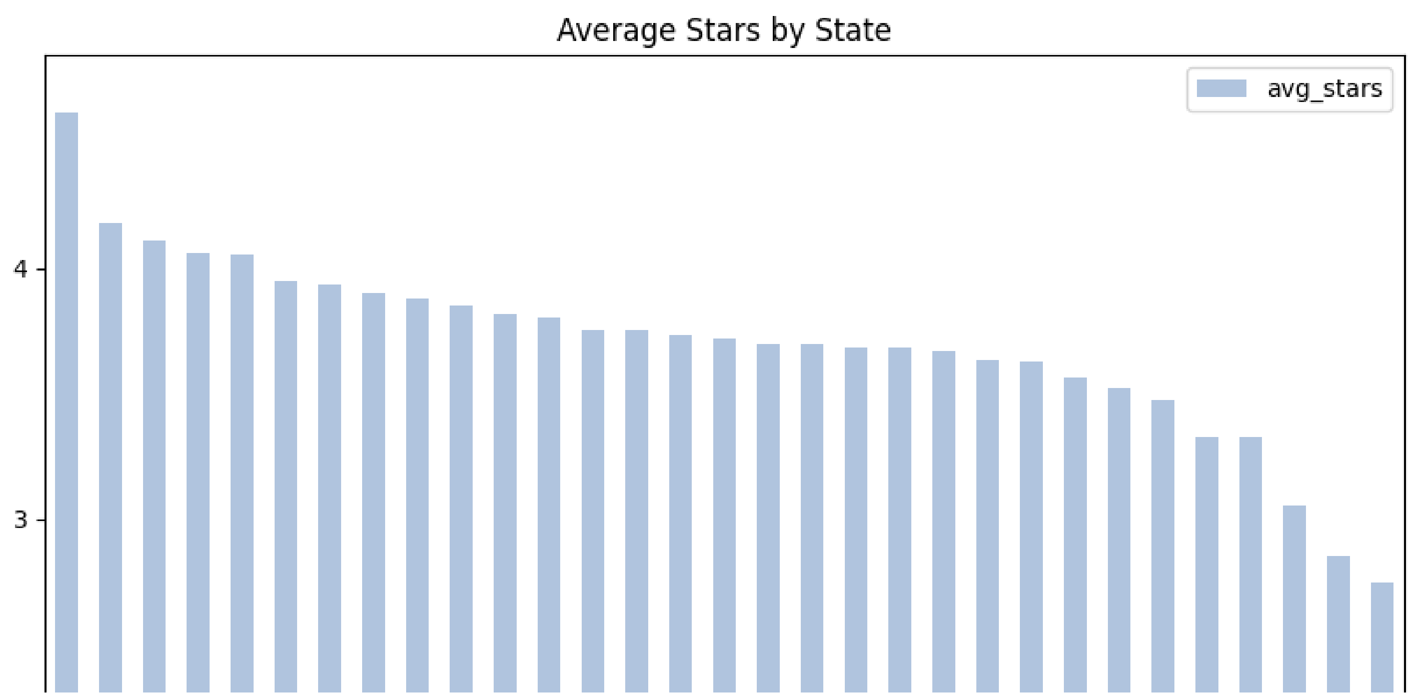
```
|state|         avg_stars|
+-----+------------------+
|   HI| 4.622857142857143|
|   NY| 4.183333333333334|
|  ABE|              4.11|
|   MN|4.0616666666666665|
|   DE| 4.057142857142857|
+-----+------------------+
only showing top 5 rows
```

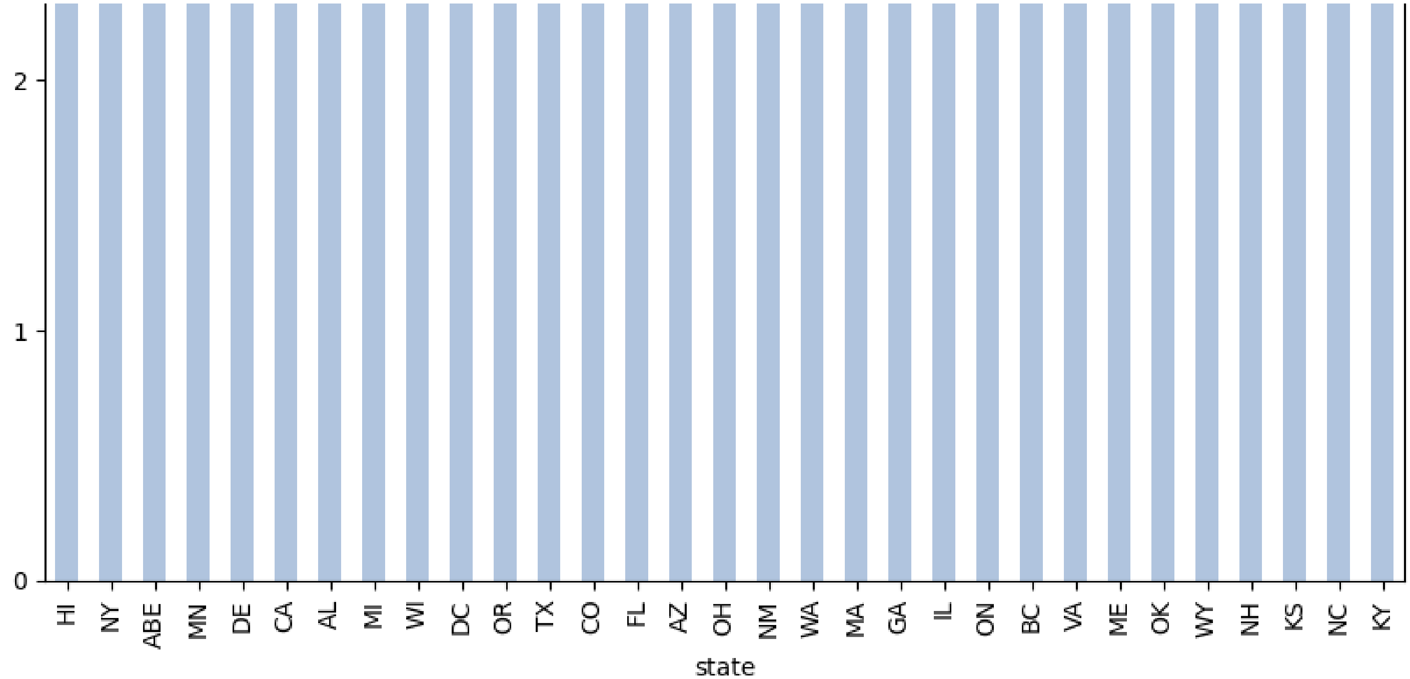
In [58]:

```
user_pandas = user_str.toPandas()
```

In [59]:

```
user_pandas.plot.bar(x='state', y='avg_stars',figsize=(10,9),title='Average Stars by State',color='lightsteelbl
%matplot plt
```




2
1
0
HI
NY
ABE
MN
DE
CA
AL
MI
WI
DC
OR
TX
CO
FL
AZ
OH
NM
WA
MA
GA
IL
ON
BC
VA
ME
OK
WY
NH
KS
NC
KY
state

In [ ]: