Database translation using Yelp Academic Dataset

By: Paul Gimeno

George Washington University

4 Parts:

1. Working with large JSON files and load it to a persistent storage

2. Yelp EDA using pandas and SQL

3. Validating data using freemium API’s (Sentiment Analysis)

4. Working with an ORM (Sql Alchemy)

Part 1 – Working with JSON and Sqlite

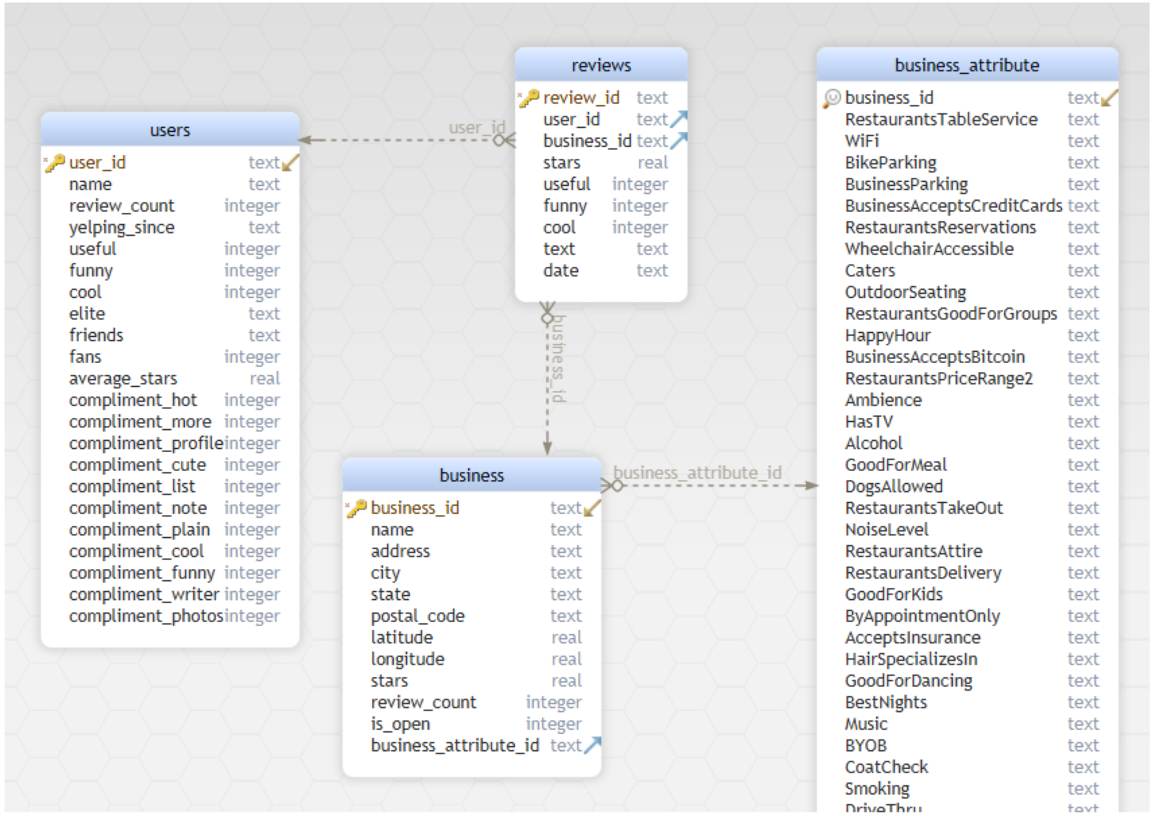
The JSON files (over 6M rows of data)

- Yelp user data set (3.43 GB)
- Yelp business data set(118.62 MB)
- Yelp review data set (6.46 GB)

Why Sqlite:

- Comes with Python’s standard library / Anaconda install
- Don’t want to set up a server
- Don’t want to set up a container
- The whole persistent storage is in one file
- Ability to build indexes and use optimized functions within the database

Part 1 – Working with JSON and Sqlite



users

user_id	text
name	text
review_count	integer
yelping_since	text
useful	integer
funny	integer
cool	integer
elite	text
friends	text
fans	integer
average_stars	real
compliment_hot	integer
compliment_more	integer
compliment_profile	integer
compliment_cute	integer
compliment_list	integer
compliment_note	integer
compliment_plain	integer
compliment_cool	integer
compliment_funny	integer
compliment_writer	integer
compliment_photos	integer

reviews

review_id	text
user_id	text
business_id	text
stars	real
useful	integer
funny	integer
cool	integer
text	text
date	text

business

business_id	text
name	text
address	text
city	text
state	text
postal_code	text
latitude	real
longitude	real
stars	real
review_count	integer
is_open	integer
business_attribute_id	text

business_attribute

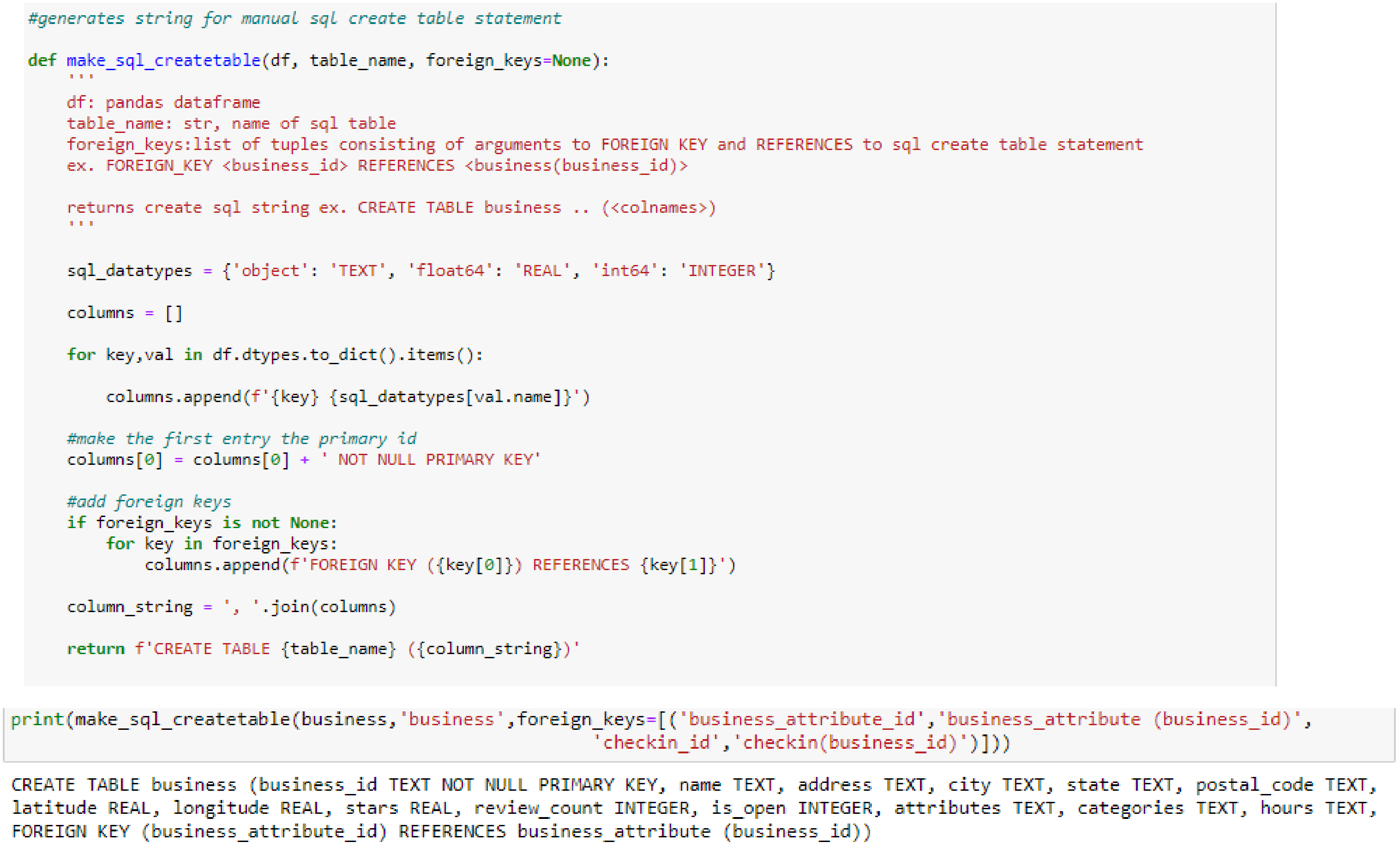
business_id	text
RestaurantsTableService	text
WiFi	text
BikeParking	text
BusinessParking	text
BusinessAcceptsCreditCards	text
RestaurantsReservations	text
WheelchairAccessible	text
Caters	text
OutdoorSeating	text
RestaurantsGoodForGroups	text
HappyHour	text
BusinessAcceptsBitcoin	text
RestaurantsPriceRange2	text
Ambience	text
HasTV	text
Alcohol	text
GoodForMeal	text
DogsAllowed	text
RestaurantsTakeOut	text
NoiseLevel	text
RestaurantsAttire	text
RestaurantsDelivery	text
GoodForKids	text
ByAppointmentOnly	text
AcceptsInsurance	text
HairSpecializesIn	text
GoodForDancing	text
BestNights	text
Music	text
BYOB	text
CoatCheck	text
Smoking	text
DriveThru	text

Courtesy of DBSchema.com
(free trial version)

Part 1 – Working with JSON and Sqlite

Problem: Don’t want to manually type a SQL CREATE statement with 50+ columns

Solution 1: write a function to write the create statement



```python
#generates string for manual sql create table statement

def make_sql_createtable(df, table_name, foreign_keys=None):
    '''
    df: pandas dataframe
    table_name: str, name of sql table
    foreign_keys:list of tuples consisting of arguments to FOREIGN KEY and REFERENCES to sql create table statement
    ex. FOREIGN_KEY <business_id> REFERENCES <business(business_id)>

    returns create sql string ex. CREATE TABLE business .. (<colnames>)
    '''

    sql_datatypes = {'object': 'TEXT', 'float64': 'REAL', 'int64': 'INTEGER'}

    columns = []

    for key,val in df.dtypes.to_dict().items():

        columns.append(f'{key} {sql_datatypes[val.name]}')

    #make the first entry the primary id
    columns[0] = columns[0] + ' NOT NULL PRIMARY KEY'

    #add foreign keys
    if foreign_keys is not None:
        for key in foreign_keys:
            columns.append(f'FOREIGN KEY ({key[0]}) REFERENCES {key[1]}')

    column_string = ', '.join(columns)

    return f'CREATE TABLE {table_name} ({column_string})'
```

```python
print(make_sql_createtable(business,'business',foreign_keys=[('business_attribute_id','business_attribute (business_id)',
                                                            'checkin_id','checkin(business_id)')]))
```

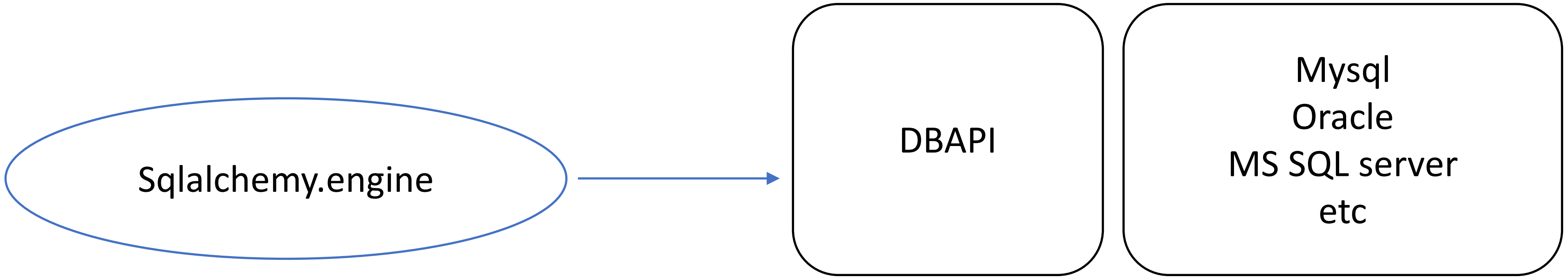
```
CREATE TABLE business (business_id TEXT NOT NULL PRIMARY KEY, name TEXT, address TEXT, city TEXT, state TEXT, postal_code TEXT,
latitude REAL, longitude REAL, stars REAL, review_count INTEGER, is_open INTEGER, attributes TEXT, categories TEXT, hours TEXT,
FOREIGN KEY (business_attribute_id) REFERENCES business_attribute (business_id))
```

Part 1 – Working with JSON and Sqlite

Problem: What if I want to migrate out of Sqlite eventually?
Approach tried : use Sqlalchemy and Pandas

```
business = pd.read_json(business_path,lines=True) #returns a dataframe
business_table.to_sql('business', con=con, if_exists='replace', index=False)
```

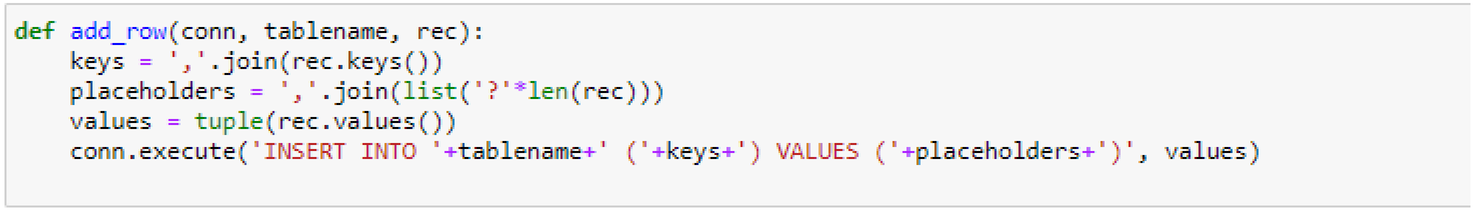
con parameter refers to the sqlalchemy engine object, this objects connects to the DBAPI



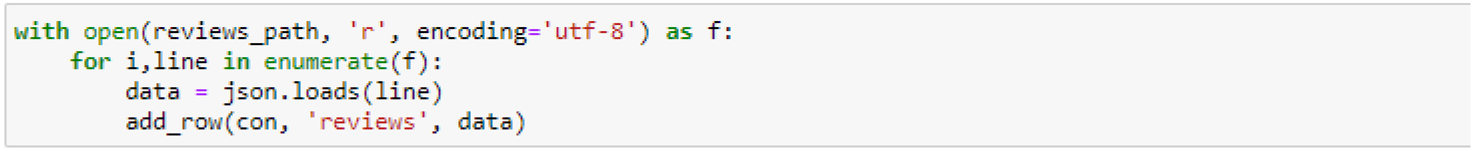
Part 1 – Working with JSON and Sqlite

Problem: The Pandas approach can be slow if the input is large. Don't
Want to commit to memory.

Approach tried : Not use pandas/sqlalchemy but try to stream the data naturally so that we
read the rows sequentially

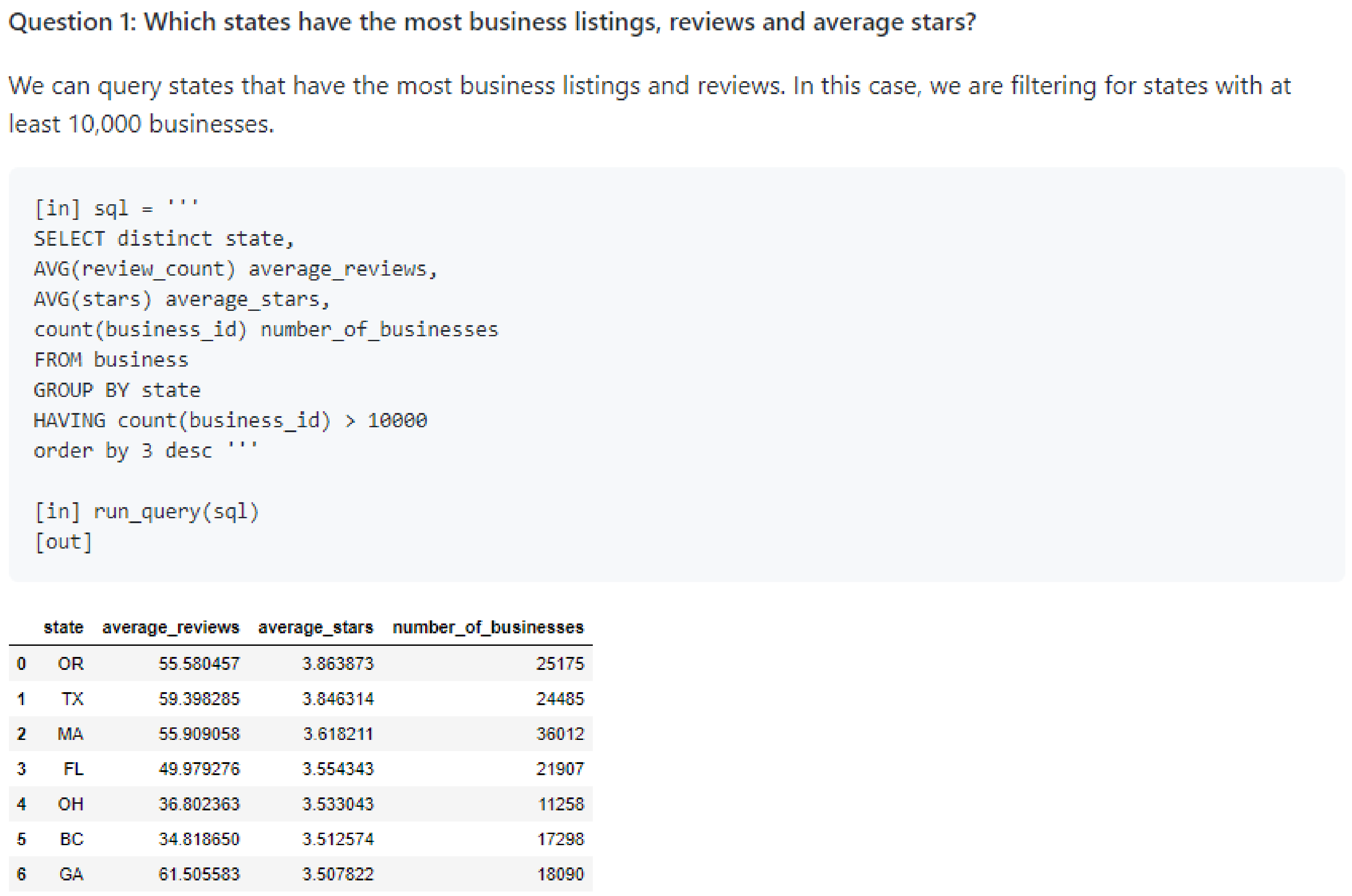


```python
def add_row(conn, tablename, rec):
    keys = ','.join(rec.keys())
    placeholders = ','.join(list('?'*len(rec)))
    values = tuple(rec.values())
    conn.execute('INSERT INTO '+tablename+' ('+keys+') VALUES ('+placeholders+')', values)
```



```python
with open(reviews_path, 'r', encoding='utf-8') as f:
    for i,line in enumerate(f):
        data = json.loads(line)
        add_row(con, 'reviews', data)
```

Part 2 – Yelp EDA using pandas and SQL



Question 1: Which states have the most business listings, reviews and average stars?

We can query states that have the most business listings and reviews. In this case, we are filtering for states with at least 10,000 businesses.

```
[in] sql = '''
SELECT distinct state,
AVG(review_count) average_reviews,
AVG(stars) average_stars,
count(business_id) number_of_businesses
FROM business
GROUP BY state
HAVING count(business_id) > 10000
order by 3 desc '''

[in] run_query(sql)
[out]
```

	state	average_reviews	average_stars	number_of_businesses
0	OR	55.580457	3.863873	25175
1	TX	59.398285	3.846314	24485
2	MA	55.909058	3.618211	36012
3	FL	49.979276	3.554343	21907
4	OH	36.802363	3.533043	11258
5	BC	34.818650	3.512574	17298
6	GA	61.505583	3.507822	18090

Part 2 – Yelp EDA using pandas and SQL

Question 2: In the state of OR, which city has the highest concentration of businesses and reviews?

```
[in] sql = '''
SELECT distinct(city),
state,
count(business_id) num_businesses
from business
where state = 'OR'
GROUP BY city
ORDER BY 3 DESC
LIMIT 1 '''
[in] run_query(sql)
[out]
```

	city	state	num_businesses
0	Portland	OR	18196

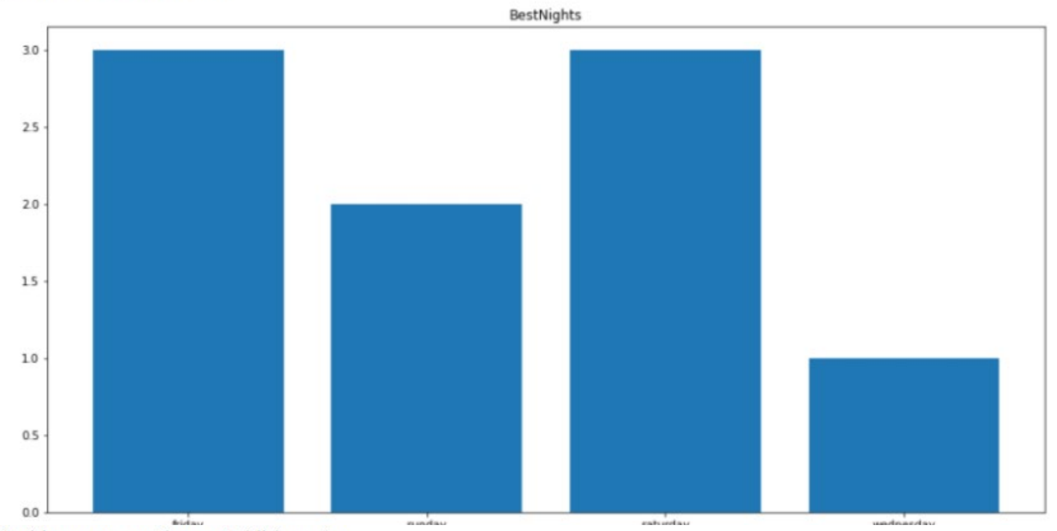
Part 2 – Yelp EDA using pandas and SQL

Question 3: Within this city, which are the top 10 most reviewed establishments and what are some of their attributes?

```
[in] sql = '''
SELECT business_id, name, city,
state, review_count,
avg(stars) average_stars
FROM business
where state = 'OR' and city = 'Portland'
GROUP BY business_id
ORDER BY 5 DESC
LIMIT 10'''

[in] run_query(sql)
[out]
```

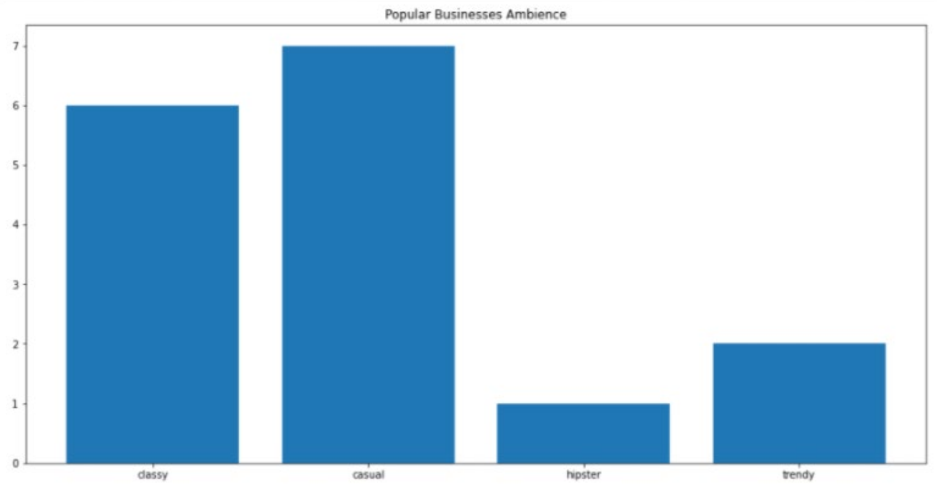
	business_id	name	city	state	review_count	average_stars
0	4CxF8c3MB7VAdY8zFb2cZQ	Voodoo Doughnut - Old Town	Portland	OR	9185	3.5
1	OQ2oHkcWA8KNC1Lsvj1SBA	Screen Door	Portland	OR	7219	4.5
2	zioLxtBc9THNS2TOn9xW1w	Pok Pok	Portland	OR	6221	4.0
3	n73rxa6e6-fTIxQzfv4BuA	Mother's Bistro & Bar	Portland	OR	4466	4.5
4	OPfgKOm_n-ajUo3qjSEgRg	Powell's City of Books	Portland	OR	4295	5.0
5	H_RM2u1WWGU1HkKZrYq2Ow	Blue Star Donuts	Portland	OR	4011	4.0
6	FBQp3R_PQIM9eGnkqzmlOw	Tasty n Alder	Portland	OR	3875	4.5
7	o_L9Ss4boqq6ZEF9xeSH6Q	Salt & Straw	Portland	OR	3672	4.5
8	5oed6H5F8qZxNzELq_1e1w	Pine State Biscuits	Portland	OR	3670	4.5
9	Ys42wLKqrflqmtqkgqOXgA	Luc Lac	Portland	OR	3199	4.0

```
plt.bar(count_nested_attributes('BestNights').keys(), count_nested_attributes('BestNights').values())
plt.title('BestNights');
```



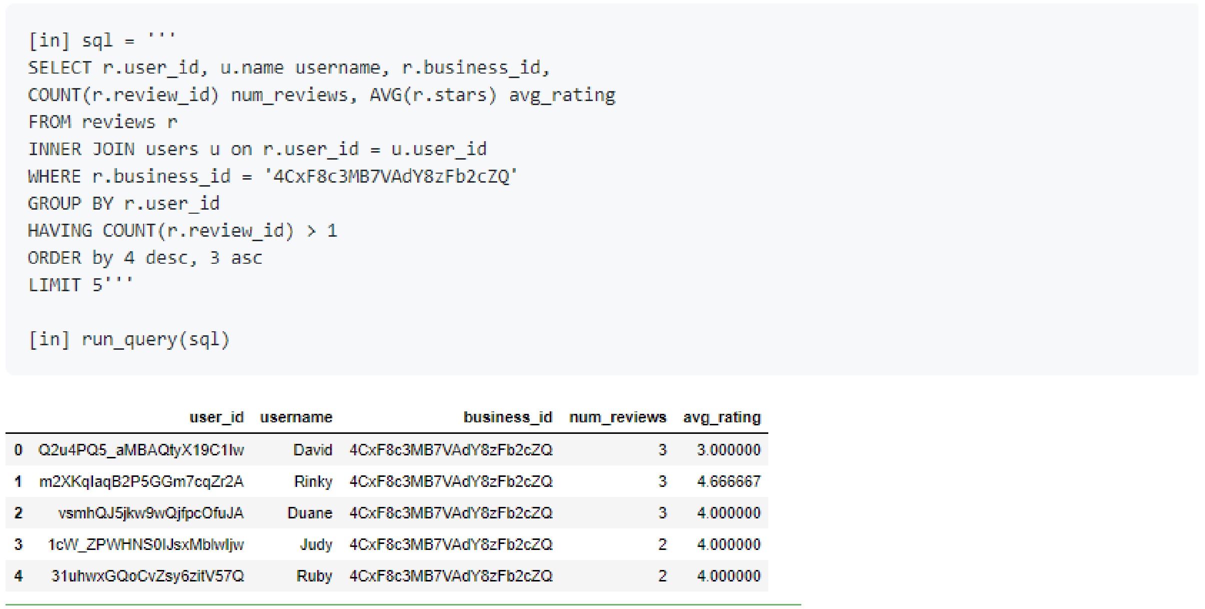
Ambience among these establishments:

```
In [193]: plt.bar(count_nested_attributes('Ambience').keys(), count_nested_attributes('Ambience').values())
          plt.title('Popular Businesses Ambience');
```



Part 2 – Yelp EDA using pandas and SQL

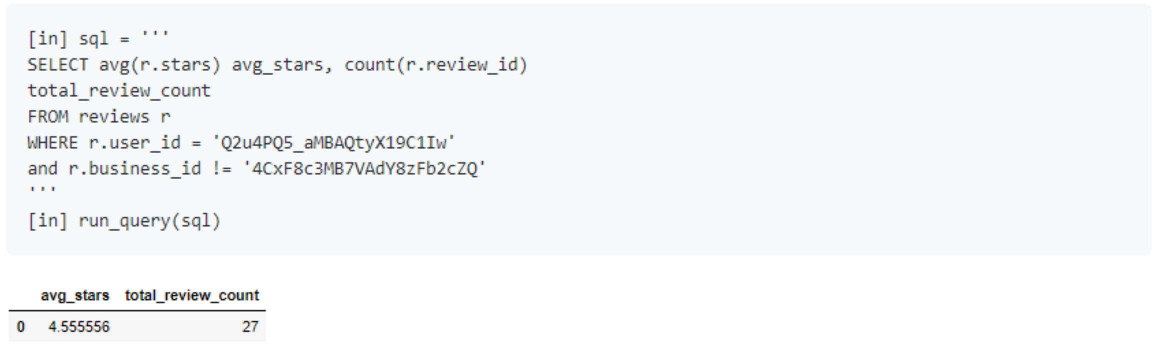
Question 4: For the most popular restaurant in this city, who is a repeat user reviewer that is the most negative?



```
[in] sql = '''
SELECT r.user_id, u.name username, r.business_id,
COUNT(r.review_id) num_reviews, AVG(r.stars) avg_rating
FROM reviews r
INNER JOIN users u on r.user_id = u.user_id
WHERE r.business_id = '4CxF8c3MB7VAdY8zFb2cZQ'
GROUP BY r.user_id
HAVING COUNT(r.review_id) > 1
ORDER by 4 desc, 3 asc
LIMIT 5'''

[in] run_query(sql)
```

	user_id	username	business_id	num_reviews	avg_rating
0	Q2u4PQ5_aMBAQtyX19C1Iw	David	4CxF8c3MB7VAdY8zFb2cZQ	3	3.000000
1	m2XKqIaqB2P5GGm7cqZr2A	Rinky	4CxF8c3MB7VAdY8zFb2cZQ	3	4.666667
2	vsmhQJ5jkw9wQjfpcOfuJA	Duane	4CxF8c3MB7VAdY8zFb2cZQ	3	4.000000
3	1cW_ZPWHNS0lJsxMblwljw	Judy	4CxF8c3MB7VAdY8zFb2cZQ	2	4.000000
4	31uhwxGQoCvZsy6zitV57Q	Ruby	4CxF8c3MB7VAdY8zFb2cZQ	2	4.000000

Question 5: For this most negative reviewer in portland, does David spread his negativity elsewhere?

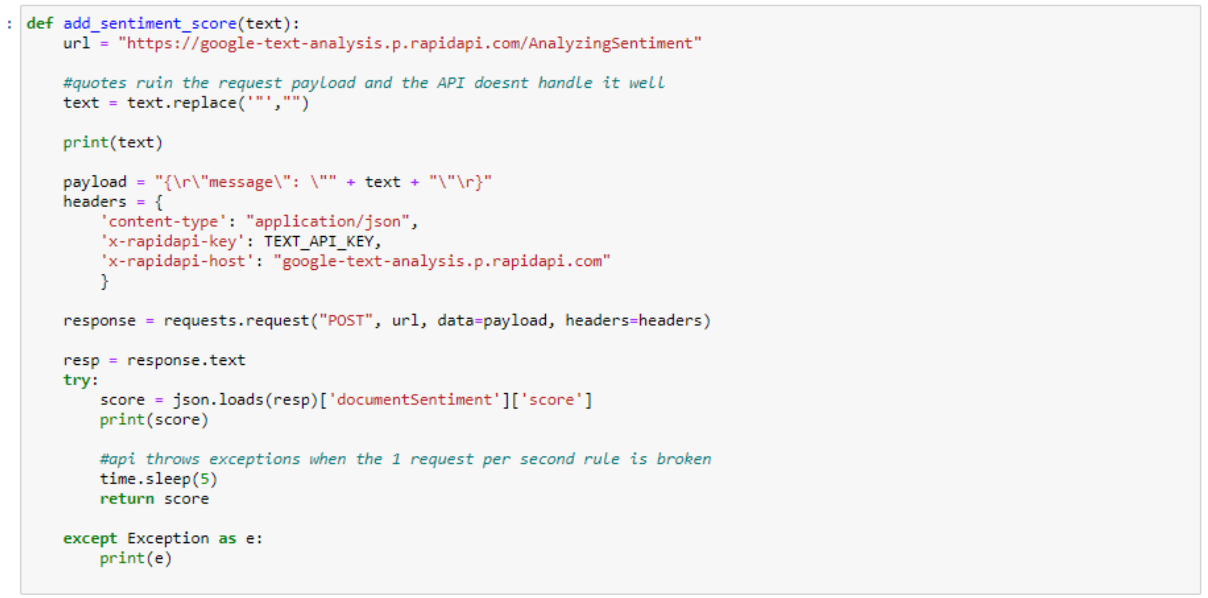


```
[in] sql = '''
SELECT avg(r.stars) avg_stars, count(r.review_id)
total_review_count
FROM reviews r
WHERE r.user_id = 'Q2u4PQ5_aMBAQtyX19C1Iw'
and r.business_id != '4CxF8c3MB7VAdY8zFb2cZQ'
'''
[in] run_query(sql)
```

	avg_stars	total_review_count
0	4.555556	27

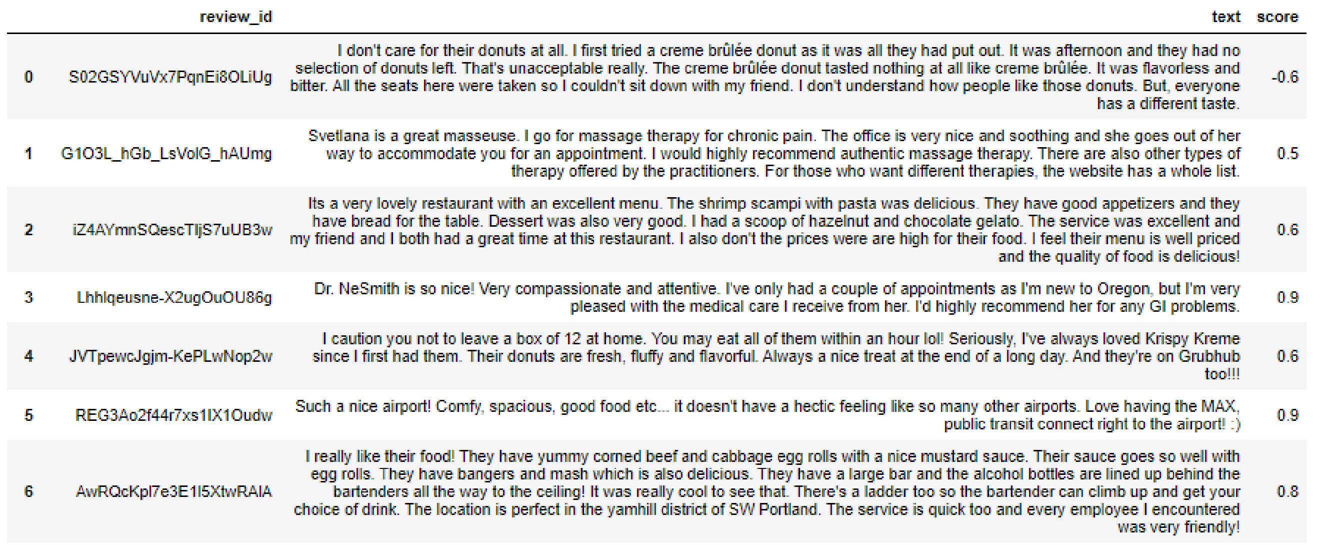
Part 3 – Quick Sentiment Analysis to validate

RapidAPI

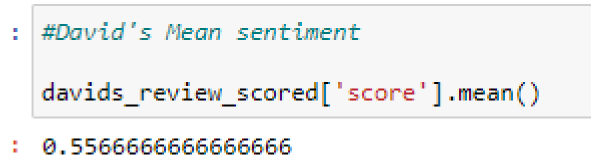


```
: def add_sentiment_score(text):
      url = "https://google-text-analysis.p.rapidapi.com/AnalyzingSentiment"

      #quotes ruin the request payload and the API doesnt handle it well
      text = text.replace('"',"")

      print(text)

      payload = "{\r\"message\": \"" + text + "\"\r}"
      headers = {
          'content-type': "application/json",
          'x-rapidapi-key': TEXT_API_KEY,
          'x-rapidapi-host': "google-text-analysis.p.rapidapi.com"
          }

      response = requests.request("POST", url, data=payload, headers=headers)

      resp = response.text
      try:
          score = json.loads(resp)['documentSentiment']['score']
          print(score)

          #api throws exceptions when the 1 request per second rule is broken
          time.sleep(5)
          return score

      except Exception as e:
          print(e)
```

```
: davids_review_scored['score'] = davids_review_scored['text'].map(add_sentiment_score)
```



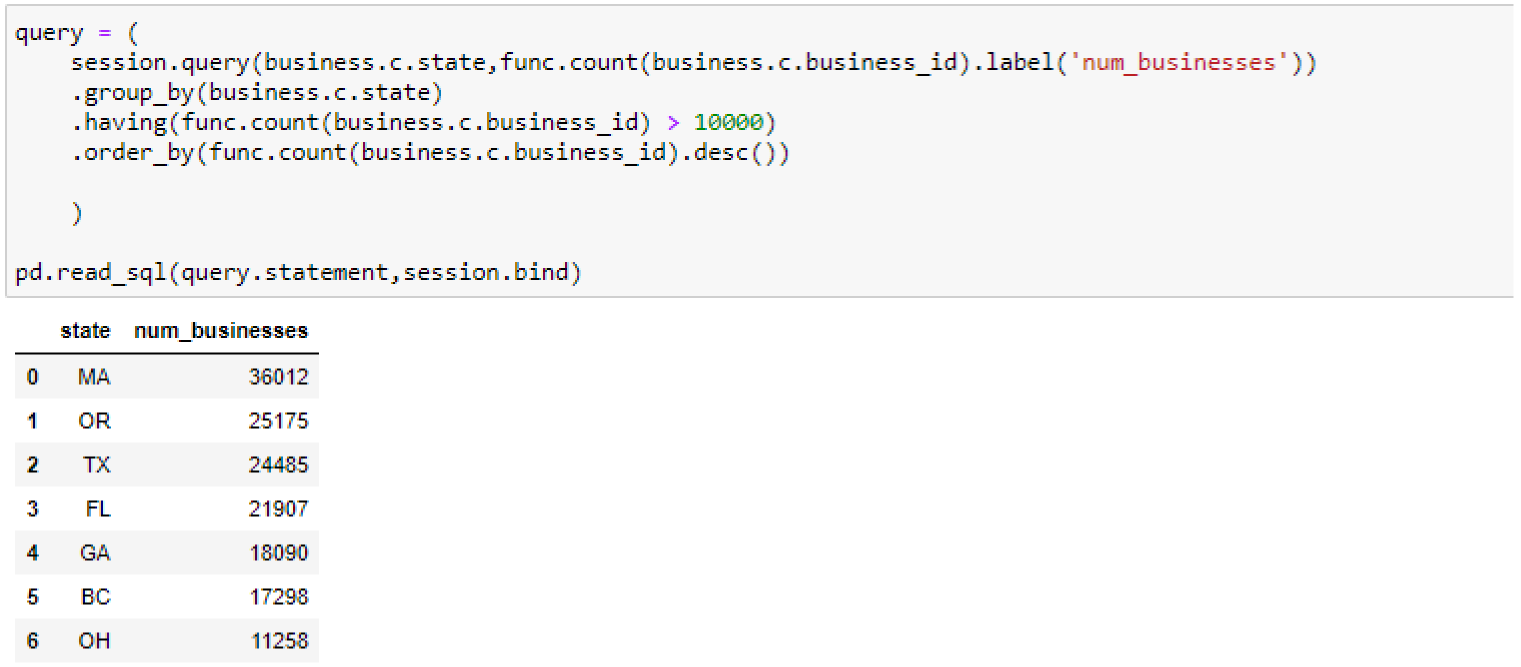
	review_id	text	score
0	S02GSYVuVx7PqnEi8OLiUg	I don't care for their donuts at all. I first tried a creme brûlée donut as it was all they had put out. It was afternoon and they had no selection of donuts left. That's unacceptable really. The creme brûlée donut tasted nothing at all like creme brûlée. It was flavorless and bitter. All the seats here were taken so I couldn't sit down with my friend. I don't understand how people like those donuts. But, everyone has a different taste.	-0.6
1	G1O3L_hGb_LsVolG_hAUmg	Svetlana is a great masseuse. I go for massage therapy for chronic pain. The office is very nice and soothing and she goes out of her way to accommodate you for an appointment. I would highly recommend authentic massage therapy. There are also other types of therapy offered by the practitioners. For those who want different therapies, the website has a whole list.	0.5
2	iZ4AYmnSQescTljS7uUB3w	Its a very lovely restaurant with an excellent menu. The shrimp scampi with pasta was delicious. They have good appetizers and they have bread for the table. Dessert was also very good. I had a scoop of hazelnut and chocolate gelato. The service was excellent and my friend and I both had a great time at this restaurant. I also don't the prices were are high for their food. I feel their menu is well priced and the quality of food is delicious!	0.6
3	Lhhlqeusne-X2ugOuOU86g	Dr. NeSmith is so nice! Very compassionate and attentive. I've only had a couple of appointments as I'm new to Oregon, but I'm very pleased with the medical care I receive from her. I'd highly recommend her for any GI problems.	0.9
4	JVTpewcJgjm-KePLwNop2w	I caution you not to leave a box of 12 at home. You may eat all of them within an hour lol! Seriously, I've always loved Krispy Kreme since I first had them. Their donuts are fresh, fluffy and flavorful. Always a nice treat at the end of a long day. And they're on Grubhub too!!!	0.6
5	REG3Ao2f44r7xs1IX1Oudw	Such a nice airport! Comfy, spacious, good food etc... it doesn't have a hectic feeling like so many other airports. Love having the MAX, public transit connect right to the airport! :)	0.9
6	AwRQcKpl7e3E1I5XtwRAIA	I really like their food! They have yummy corned beef and cabbage egg rolls with a nice mustard sauce. Their sauce goes so well with egg rolls. They have bangers and mash which is also delicious. They have a large bar and the alcohol bottles are lined up behind the bartenders all the way to the ceiling! It was really cool to see that. There's a ladder too so the bartender can climb up and get your choice of drink. The location is perfect in the yamhill district of SW Portland. The service is quick too and every employee I encountered was very friendly!	0.8



```
: #David's Mean sentiment

  davids_review_scored['score'].mean()

: 0.5566666666666666
```

Part 4 – using an ORM

Query Sample finding states with most business listings.
** can chain multiple function calls

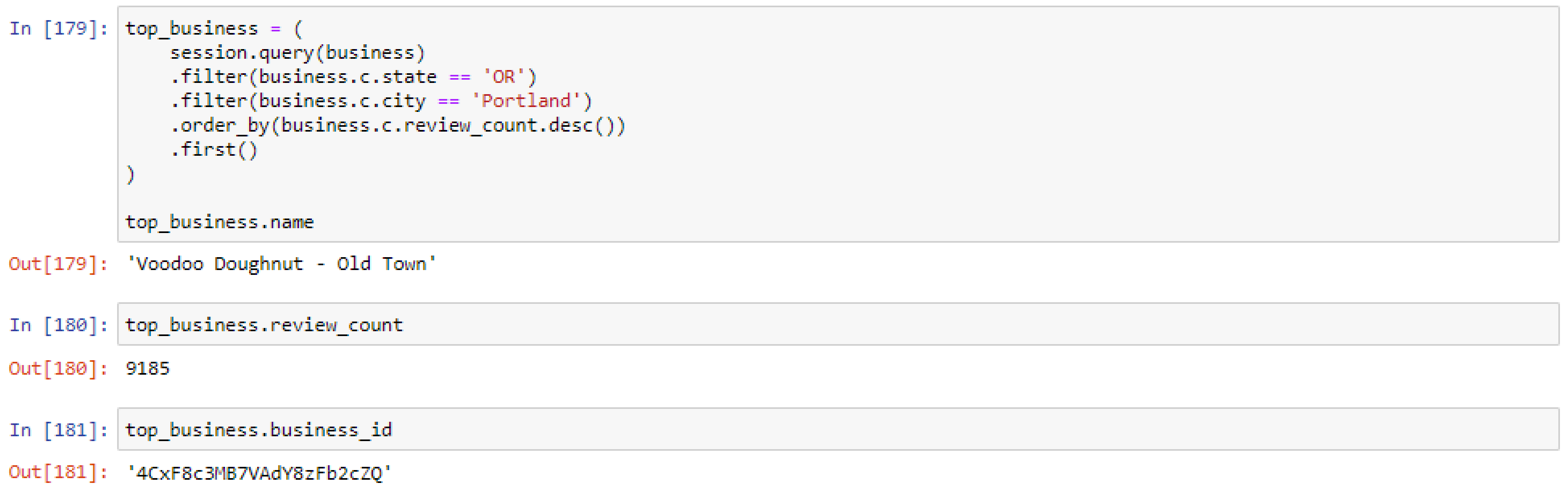


```
query = (
    session.query(business.c.state,func.count(business.c.business_id).label('num_businesses'))
    .group_by(business.c.state)
    .having(func.count(business.c.business_id) > 10000)
    .order_by(func.count(business.c.business_id).desc())

    )

pd.read_sql(query.statement,session.bind)
```

	state	num_businesses
0	MA	36012
1	OR	25175
2	TX	24485
3	FL	21907
4	GA	18090
5	BC	17298
6	OH	11258

Part 4 – using an ORM

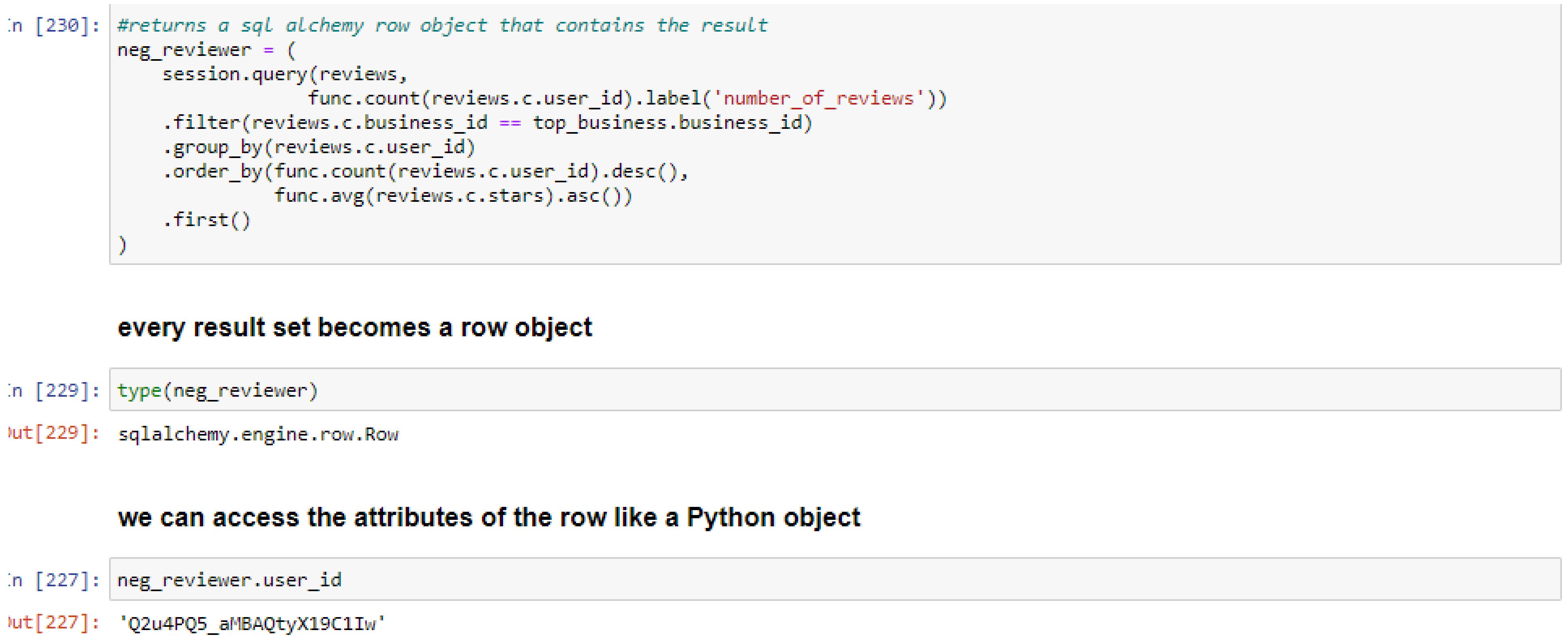
We can represent results as python objects



```
In [179]: top_business = (
              session.query(business)
              .filter(business.c.state == 'OR')
              .filter(business.c.city == 'Portland')
              .order_by(business.c.review_count.desc())
              .first()
          )

          top_business.name

Out[179]: 'Voodoo Doughnut - Old Town'

In [180]: top_business.review_count

Out[180]: 9185

In [181]: top_business.business_id

Out[181]: '4CxF8c3MB7VAdY8zFb2cZQ'
```

Part 4 – using an ORM

We can represent result rows as row objects



```
n [230]: #returns a sql alchemy row object that contains the result
         neg_reviewer = (
             session.query(reviews,
                           func.count(reviews.c.user_id).label('number_of_reviews'))
             .filter(reviews.c.business_id == top_business.business_id)
             .group_by(reviews.c.user_id)
             .order_by(func.count(reviews.c.user_id).desc(),
                       func.avg(reviews.c.stars).asc())
             .first()
         )
```

every result set becomes a row object

```
n [229]: type(neg_reviewer)

ut[229]: sqlalchemy.engine.row.Row
```

we can access the attributes of the row like a Python object

```
n [227]: neg_reviewer.user_id

ut[227]: 'Q2u4PQ5_aMBAQtyX19C1Iw'
```

Part 4 – using an ORM

Pro’s and Cons of an ORM like SQL Alchemy

Pro’s:

1. Can represent my data model as objects
2. Not bound to a flavor of SQL/database management system.
3. Scalability: Integrates with application logic and can scale along with it

Cons:

1. There are many ways to do things
2. Can be slow
3. Can be complicated to follow when using large nested function calls