Big Data Project:

Analysis of Yelp reviews

created within the scope of the Master Program
Business Intelligence and Process Management
at the HWR Berlin

submitted by
Anna Anisienia (425601)
Irina Korchagina (609320)
Francis Liu (609498)
Emre Övey (464138)

Supervisor: Prof. Dr. Markus Schaal
Summer Semester 2018

Table of Content

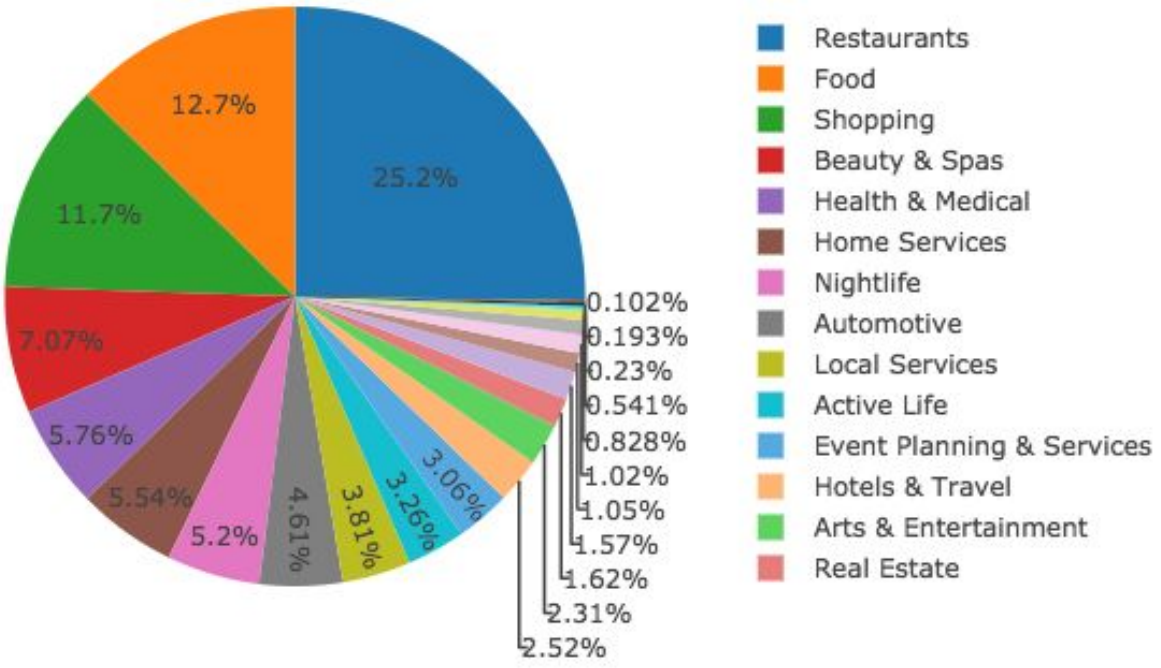
About the project

This project aims at analyzing Yelp reviews: Big Dataset consisting of 9 GB of data spread across nine tables. This dataset initially comes from the Yelp Dataset Challenge[1], but we used a CSV version of that from Kaggle[2]. In particular, we intended to apply machine learning to generate insights for businesses and to visualize our findings.

Our target group

The scope of this project would be too broad if we would try to satisfy the needs of all possible Yelp businesses. That is why we decided to limit our target group to address only one particular type of business.

The following diagram[3] visualizes the top business categories that are currently reviewed on Yelp. Based on the two first largest items of the pie chart, it is clear that the food service industry constitutes ⅓ of all reviews. After researching about those businesses, we came across a study conducted by Harvard Business School[4], which implied that Yelp reviews have no impact on chain restaurants and their revenues. In contrast, independent restaurants can benefit from Yelp substantially. That is why we narrowed down our final target group to independent restaurants.



[1] https://www.yelp.com/dataset/challenge
[2] https://www.kaggle.com/yelp-dataset/yelp-dataset
[3] https://zhiyangzeng.github.io/
[4] https://www.hbs.edu/faculty/Pages/item.aspx?num=41233

Impact of Yelp reviews on businesses

In this section, we try to answer the question: how independent restaurants can benefit from using Yelp? The same study from Harvard Business School we referred to earlier compared the Yelp reviews with the revenue data from the Washington State Department. They found that a 1-star increase in Yelp rating leads to a 5 to 9 percent increase in revenue for independent restaurants. Of course, the study used the ceteris paribus principle: it assumes that we keep all other factors constant. But overall, this effect is so substantial that we decided to focus our attention solely on this target group.

The additional impact of Yelp reviews is that more and more consumers prefer to read online reviews rather than relying on traditional sources of reputation, such as word-of-mouth, which highlights the importance of review websites such as Yelp.com.

Furthermore, the study concludes that most businesses use and analyze only a fraction of data available in the reviews, which was the ultimate reason for our project. We want to change this trend by offering more insights for Yelp businesses.

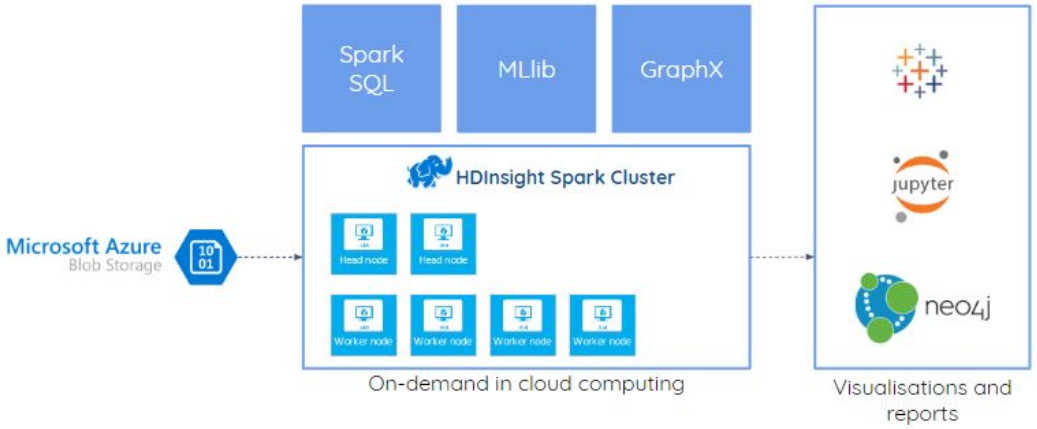
The scope of the project

We can assume that all what independent restaurants care about is offering good food and service to their customers to maximize the profits. Naturally, businesses can only maximize their profits, if they can maintain a positive image to the public. That is why many independent restaurants use Yelp to improve their company image by showing other customers positive feedback from previous customers, which ideally should lead to more potential clients and a better company image. However, no matter how hard one would try, it is impossible to obtain only positive feedback, since we cannot prevent fake reviews from happening. But what we can do is to identify them and filter them out for the internal business analysis so that we can improve our products and services based on real customer feedback. This analysis would also allow finding the competitors on the map, along with the sentiment of their customers' reviews. What is more, by using the results from our project, independent restaurants can also find the most significant users of their business.

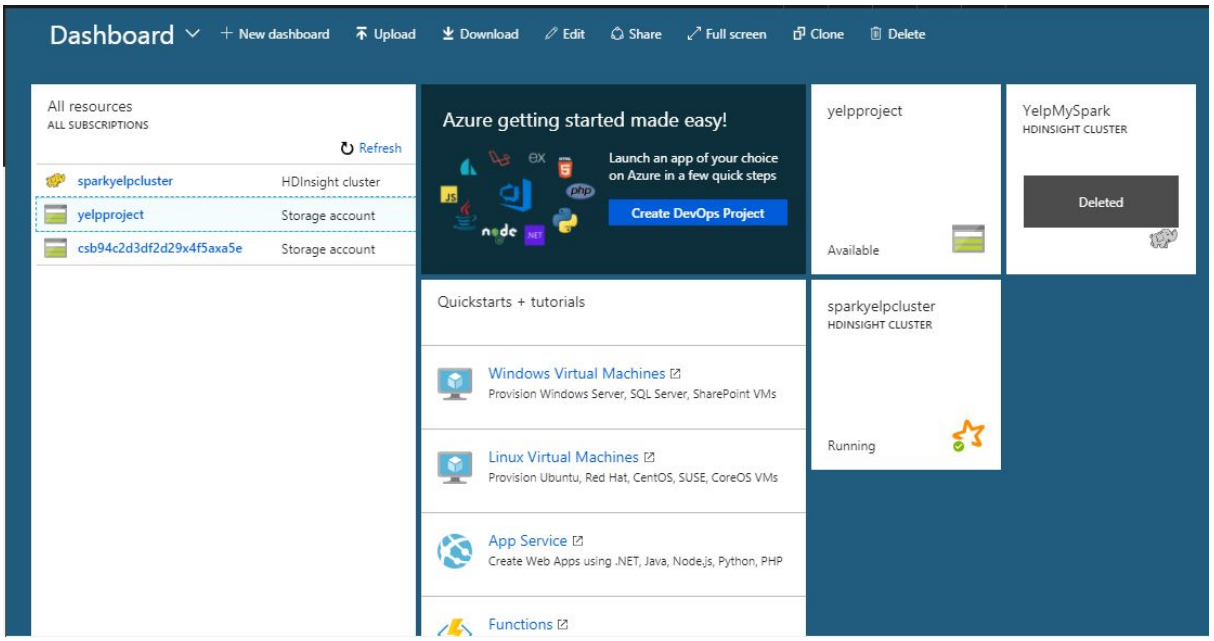
Furthermore, we did in this project something that most businesses fail to do - namely, we analyzed the textual review data in more depth by using sentiment and graph analysis, topic modeling and review rating prediction.

Environment

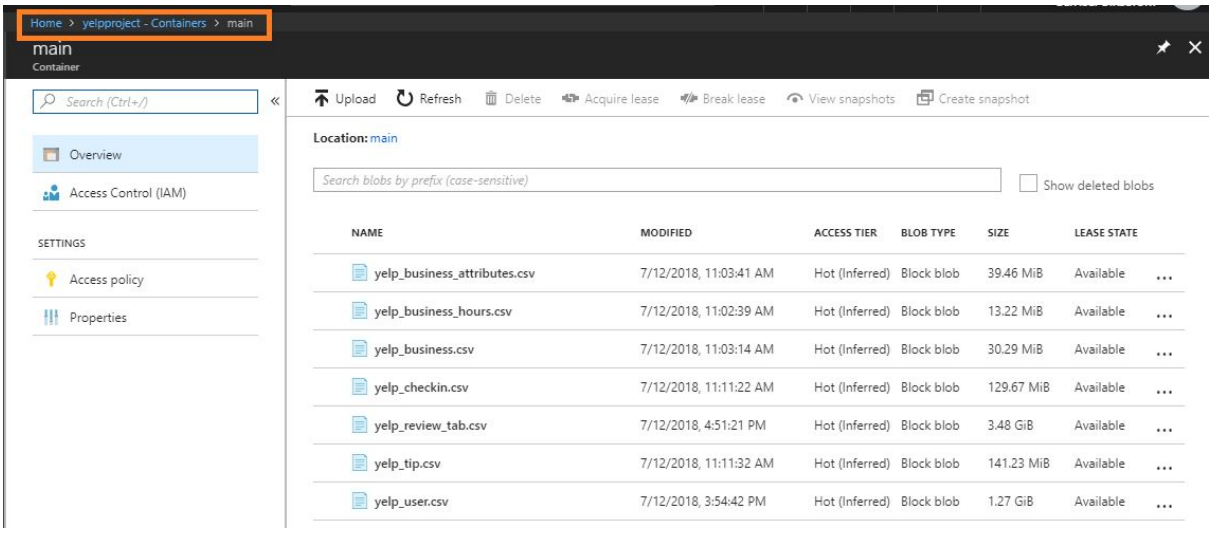
Our main working environment for this project is Microsoft Azure.



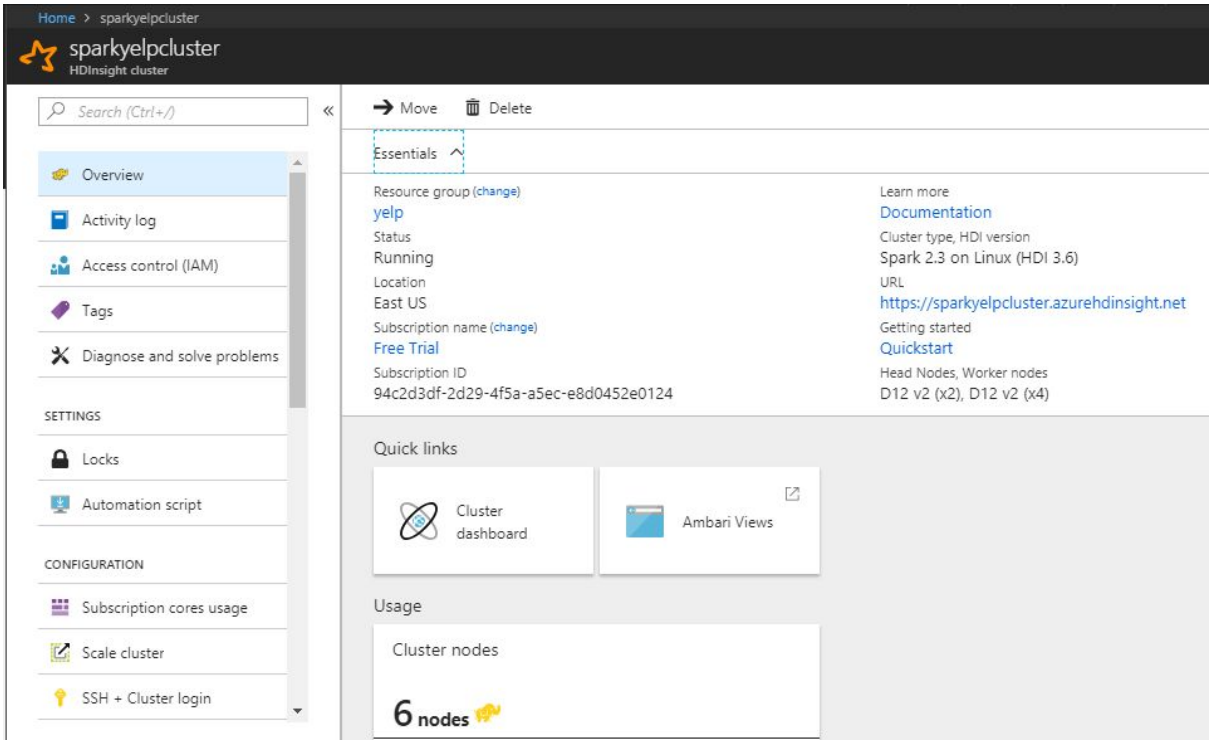
The results are visualized either in Tableau, Jupyter notebooks or Neo4j.
Microsoft Azure dashboard:



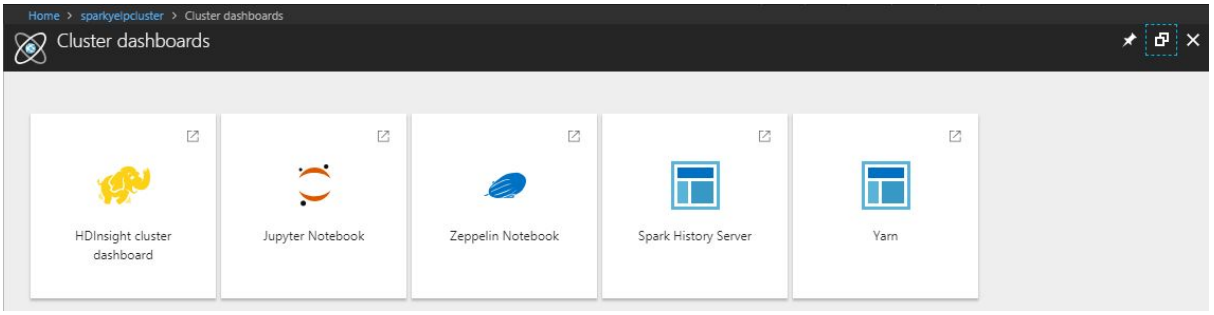
Firstly, the Yelp dataset was uploaded into Microsoft Azure Blob Storage:



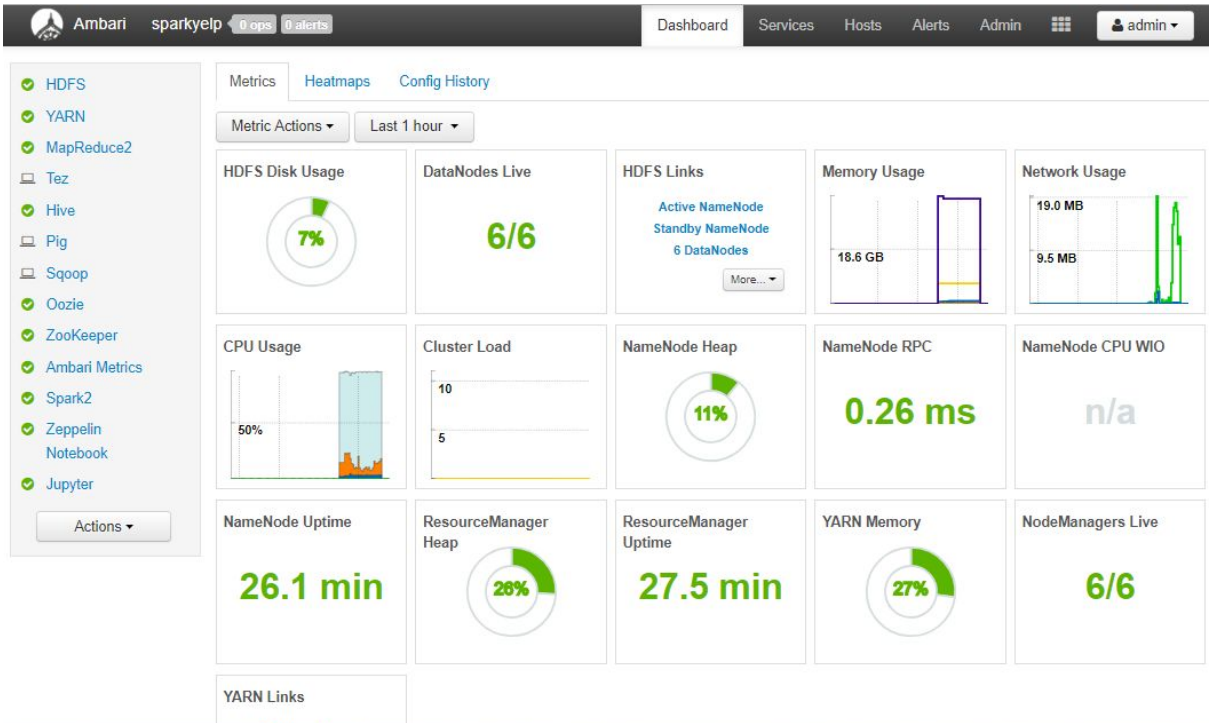
After the deployment of Spark cluster, next cluster overview is available:



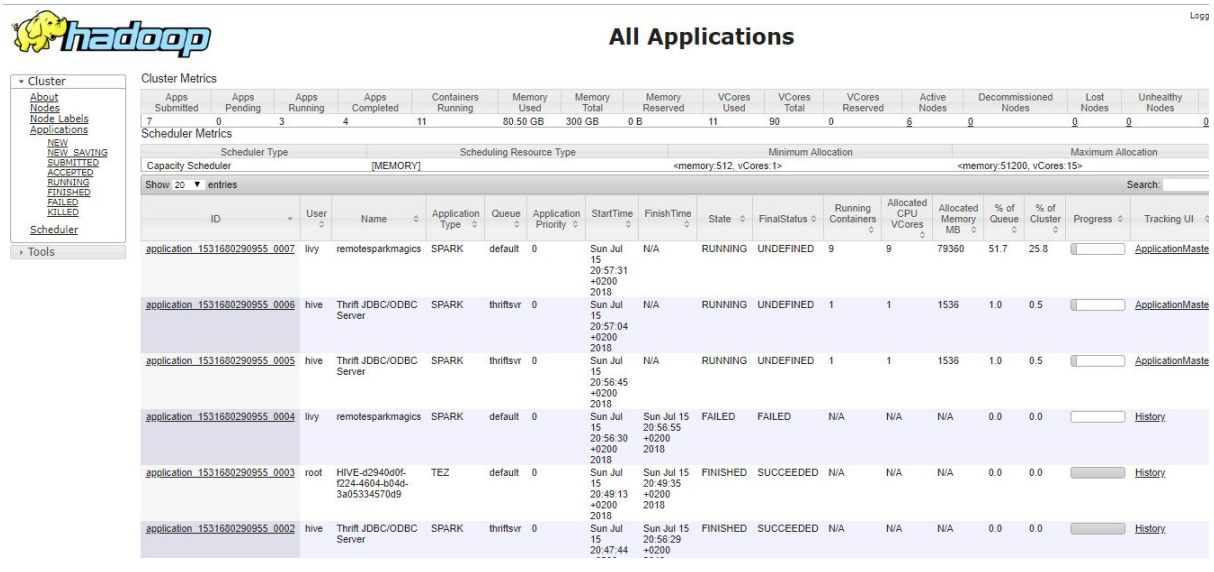
Available cluster dashboards:



Ambari view:



Yarn view:



Yelp Data Exploration

In this section we explore the Yelp businesses and summarize our findings through visualizations and summary statistics.

Location of businesses & their average ratings

In following, we plot the average star rating distribution on the map for all businesses.

```
# remove col address and save to csv
import pandas as pd
df = pd.read_csv('yelp_business.csv')
df.head()
```

	business_id	name	neighborhood	address	city	state	postal_code	latitude	longitude	stars	review_count	is_open
0	FYWN1wneV18bWNgQjJ2GNg	"Dental by Design"	NaN	"4855 E Warner Rd, Ste B9"	Ahwatukee	AZ	85044	33.330690	-111.978599	4.0	22	1
1	He-G7vWjzVUysIKrfNbPUQ	"Stephen Szabo Salon"	NaN	"3101 Washington Rd"	McMurray	PA	15317	40.291685	-80.104900	3.0	11	1

```
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True) # set up everything so that the figures show up in the notebook offline

data = [dict(type = 'scattergeo',
             #locations = ['AZ','CA','NY'],
             locationmode = 'USA-states',
             lon = df["longitude"], lat = df["latitude"],
             text = df["name"],
             marker = dict(size=4*df["stars"],
                           opacity=0.8, line = dict(width=1, color="black"),
                           autocolorscale=False, colorscale="custom-colorscale", color=df["stars"],
             colorbar=dict(title="Stars")))]

layout = dict(title="USA businesses by nr of stars",
                geo = dict(scope="usa", #=dict(type="usa"),
                        showland=True, landcolor="rgb(250,250,250)", #symbol = "pentagon",
                        subunitcolor="rgb(217,217,217)", countrycolor="rgb(0, 0, 0)",
                        countrywidth=0.5, subunitwidth=0.5))
choromap = go.Figure(data = data,layout = layout)
iplot(choromap)
```

In the plot above, you can see the location of businesses, colored by their average star rating. To gain a different point of view, we create the plot of states with their respective average rating.

```
topstates = df.groupby("state").mean().sort_values(by=['stars'], ascending = False)
topstates["state"] = topstates.index
topstates.head()
```

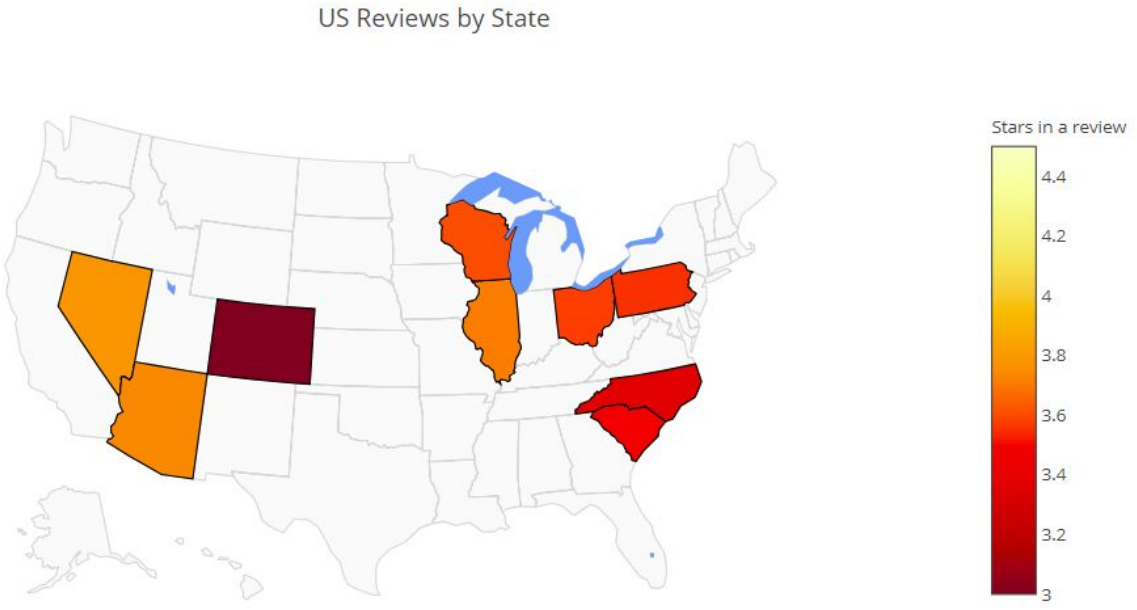
state	latitude	longitude	stars	review_count	is_open	state
MLN	55.962444	-3.197662	4.500000	16.000000	1.000000	MLN
CHE	53.187298	-2.892515	4.000000	4.000000	1.000000	CHE
EDH	55.949431	-3.200403	3.785714	12.000000	0.785714	EDH
NV	36.117819	-115.163201	3.780612	49.780612	0.852041	NV
AZ	33.501149	-111.989656	3.739510	33.811189	0.870629	AZ

```
state_names = [state_abbrev[i] for i in topstates.index]
print(state_names)
```

```
['Edinburgh', 'Chester', 'Edinburgh', 'Nevada', 'Arizona', 'Illinois', 'Baden-Württemberg', 'Quebec', 'Wisconsin', 'Ohio', 'Pennsylvania', 'Dunfermline', 'South Carolina', 'Ontario', 'North Carolina', 'Colorado']
```

```
topstates["state_names"] = state_names
```

```
#import colorlover as cl
data = dict(type='choropleth',
            colorscale = "YlOrRd",
            locations = topstates['state'],
            z = topstates['stars'],
            locationmode = 'USA-states',
            text = topstates['state_names'],
            marker = dict(line = dict(color = 'rgb(0,0,0)',width = 1)),
            colorbar = {'title':"Stars in a review"})

layout = dict(title = 'US Reviews by State',
              geo = dict(showland=True, landcolor = "rgb(250,250,250)", scope='usa',
                         subunitcolor="rgb(217,217,217)", countrycolor="rgb(0, 0, 0)",
                          showlakes = True, lakecolor = 'rgb(85,173,240)'))

choromap = go.Figure(data = [data],layout = layout)
iplot(choromap)
```



By hovering over the plot, one can see the states and the average star rating for businesses in this particular state.

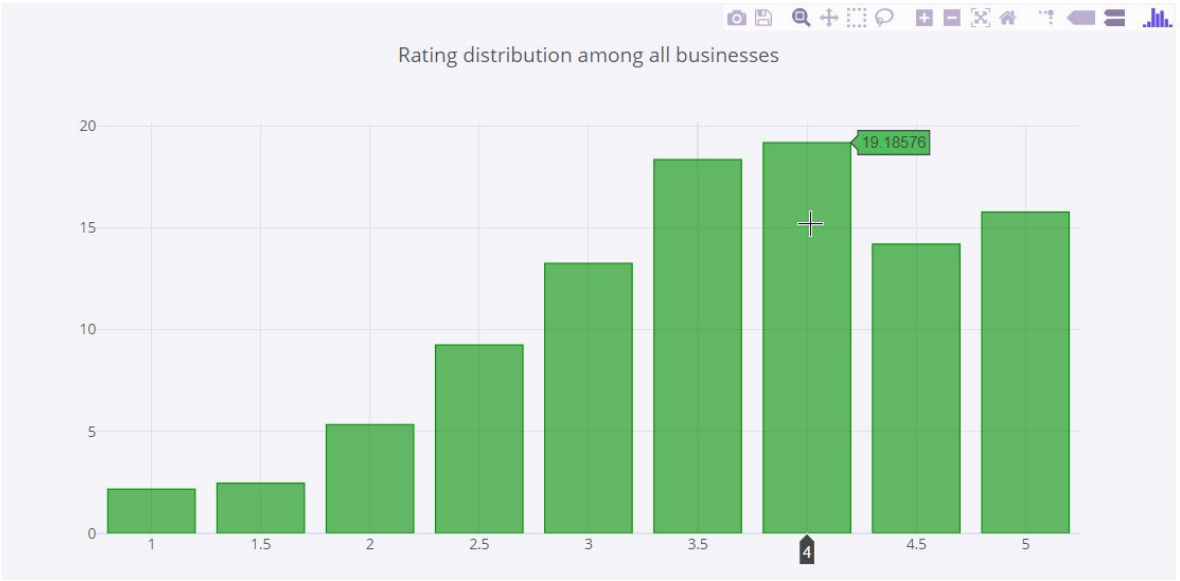
Average rating distribution among businesses

Each business has average rating assigned to them. The bar chart below shows the distribution of average rating of all businesses. What we can infer from this chart is that only around 20% of businesses has average star rating below 2.5. Furthermore, we can see that most businesses have average rating of 4.0 or 3.5 stars.

```
counts = df.groupby("stars").count() # only around 20% of businesses has overall nr of stars <= 2.5
counts = counts.apply(lambda x: 100 * x / float(x.sum())) # percentage
counts["stars"] = counts.index
counts
```

stars	business_id	name	neighborhood	address	city	state	postal_code	latitude	longitude	review_count	is_open	categories	stars
1.0	2.169940	2.169940	1.754025	2.169940	2.169953	2.169953	2.140919	2.169953	2.169953	2.169940	2.169940	2.169940	1.0
1.5	2.464956	2.464956	2.193634	2.464956	2.464970	2.464970	2.465736	2.464970	2.464970	2.464956	2.464956	2.464956	1.5
2.0	5.338924	5.338924	4.923914	5.338924	5.338955	5.338955	5.343674	5.338955	5.338955	5.338924	5.338924	5.338924	2.0
2.5	9.250316	9.250316	8.792178	9.250316	9.250369	9.250369	9.244929	9.250369	9.250369	9.250316	9.250316	9.250316	2.5
3.0	13.256801	13.256801	13.673454	13.256801	13.256877	13.256877	13.276687	13.256304	13.256304	13.256801	13.256801	13.256801	3.0
3.5	18.352839	18.352839	19.657429	18.352839	18.352944	18.352944	18.372580	18.352944	18.352944	18.352839	18.352839	18.352839	3.5
4.0	19.185757	19.185757	21.404102	19.185757	19.185294	19.185867	19.207906	19.185867	19.185867	19.185757	19.185757	19.185757	4.0
4.5	14.204288	14.204288	14.829082	14.204288	14.204370	14.204370	14.210895	14.204370	14.204370	14.204288	14.204288	14.204288	4.5
5.0	15.776178	15.776178	12.772183	15.776178	15.776268	15.775695	15.736674	15.776268	15.776268	15.776178	15.776178	15.776178	5.0

```
counts.iplot(kind='bar', x='stars', y='business_id', color="green", title = "Rating distribution among all businesses")
```



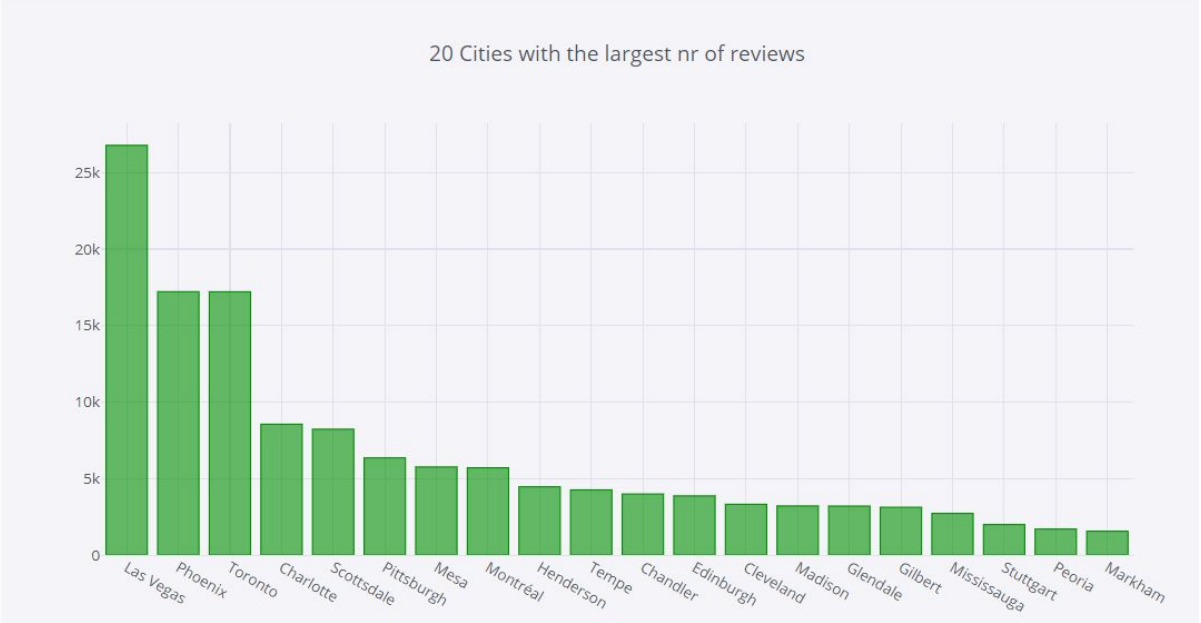
Where are the most businesses on Yelp located?

By looking at the plot below, we can find the most famous cities where Yelp restaurants reside.

```
top_cities = df.groupby("city").count().sort_values(by=['business_id'], ascending = False).head(20)
top_cities["city"] = top_cities.index
top_cities
# show 20 cities with the largest nr of reviews
```

city	business_id	name	neighborhood	address	state	postal_code	latitude	longitude	stars	review_count	is_open	categories	city
Las Vegas	26775	26775	21887	26775	26775	26655	26775	26775	26775	26775	26775	26775	Las Vegas
Phoenix	17213	17213	0	17213	17213	17121	17213	17213	17213	17213	17213	17213	Phoenix
Toronto	17206	17206	14064	17206	17206	17102	17205	17205	17206	17206	17206	17206	Toronto

```
top_cities.iplot(kind='bar', x='city', y='review_count', color="green", title = "20 Cities with the largest nr of reviews")
```



Top businesses

Which businesses have the highest number of reviews among all businesses that have average rating of 5.0 stars? The answer is: *Little Miss BBQ* wins! *Brew Tea Bar* also shows a high number of reviews, among which almost all are 5 stars!

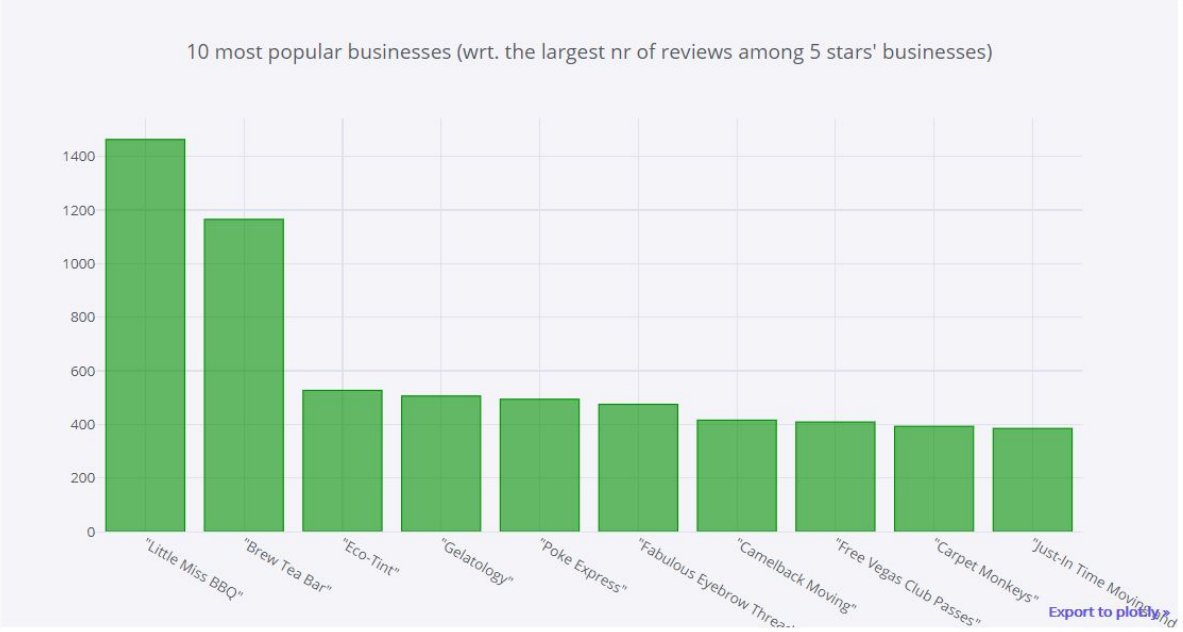
```
best = df.query('stars == 5.0')
len(best) # nrow
```

```
27540
```

```
best_popular = best.sort_values(by=['review_count'], ascending = False)
# show 10 most popular businesses, i.e. those with the largest nr of reviews among 5 stars' businesses
top10 = best_popular.head(10)
top10
```

	business_id	name	neighborhood	address	city	state	postal_code	latitude	longitude	stars	review_count	is_open
153363	Xg5qEQiB-7L6kGJ5F4K3bQ	"Little Miss BBQ"	NaN	"4301 E University Dr"	Phoenix	AZ	85034	33.421808	-111.989174	5.0	1463	1
13577	IhNASEZ3XnBHmuuVnWdIwA	"Brew Tea Bar"	Southwest	"7380 S Rainbow Blvd, Ste 101"	Las Vegas	NV	89139	36.054195	-115.242443	5.0	1165	1

```
top10.iplot(kind='bar', x='name', y='review_count', color="green", text="categories",
            title = "10 most popular businesses (wrt. the largest nr of reviews among 5 stars' businesses)")
```



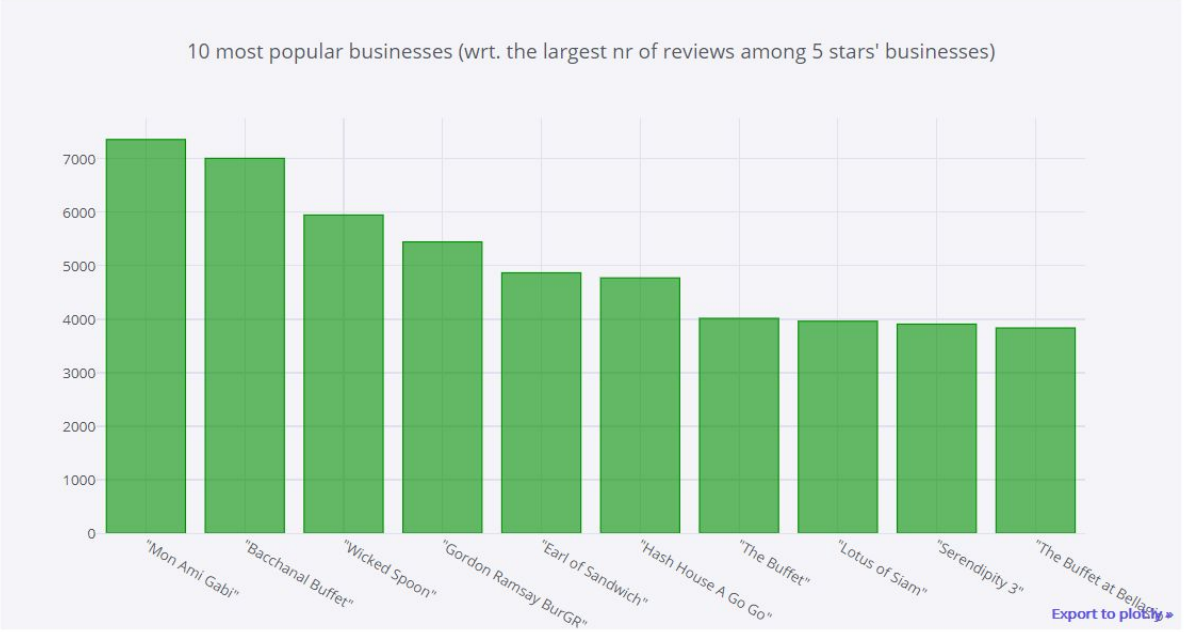
Most popular businesses

The popularity can be also measured in the number of reviews. The following plot shows businesses that have the largest number of reviews on Yelp.

```
top10_all = df.sort_values(by=['review_count'], ascending = False).head(10)
top10_all
# show 10 most popular businesses overall (i.e. not only among 5.0 stars businesses) wrt. the review count
```

	business_id	name	neighborhood	address	city	state	postal_code	latitude	longitude	stars	review_count	is_open
97944	4JNXUYY8wbaaDmk3BPzlWw	"Mon Ami Gabi"	The Strip	"3655 Las Vegas Blvd S"	Las Vegas	NV	89109	36.112827	-115.172581	4.0	7361	1
119907	RESDUcs7fIiihp38-d6_6g	"Bacchanal Buffet"	The Strip	"3570 S Las Vegas Blvd"	Las Vegas	NV	89109	36.116113	-115.176222	4.0	7009	1
69993	K7lWdNUhCbcnEvI0NhGewg	"Wicked Spoon"	The Strip	"3708 Las Vegas Blvd S"	Las Vegas	NV	89109	36.109538	-115.176170	3.5	5950	1

```
top10_all.iplot(kind='bar', x='name', y='review_count', color="green", text="stars",
           title = "10 most popular businesses (wrt. the largest nr of reviews among 5 stars' businesses)")
```



States with the largest number of Yelp reviews

We want to find out, which states have a highest number of Yelp reviews. For improved understandability of the plot, we convert the state abbreviations into full names by using a dictionary.

```
topstates = df.groupby("state").count().sort_values(by=['business_id'], ascending = False).head(10)
```

```
topstates.index
```

```
Index(['AZ', 'NV', 'ON', 'NC', 'OH', 'PA', 'QC', 'WI', 'EDH', 'BW'], dtype='object', name='state')
```

```
ON = df[df["state"] == "BW"]
```

At this point, we create a dictionary, where state abbreviations are the keys and values are the full names of the state. For readability purposes, we show only 6 key-value pairs:

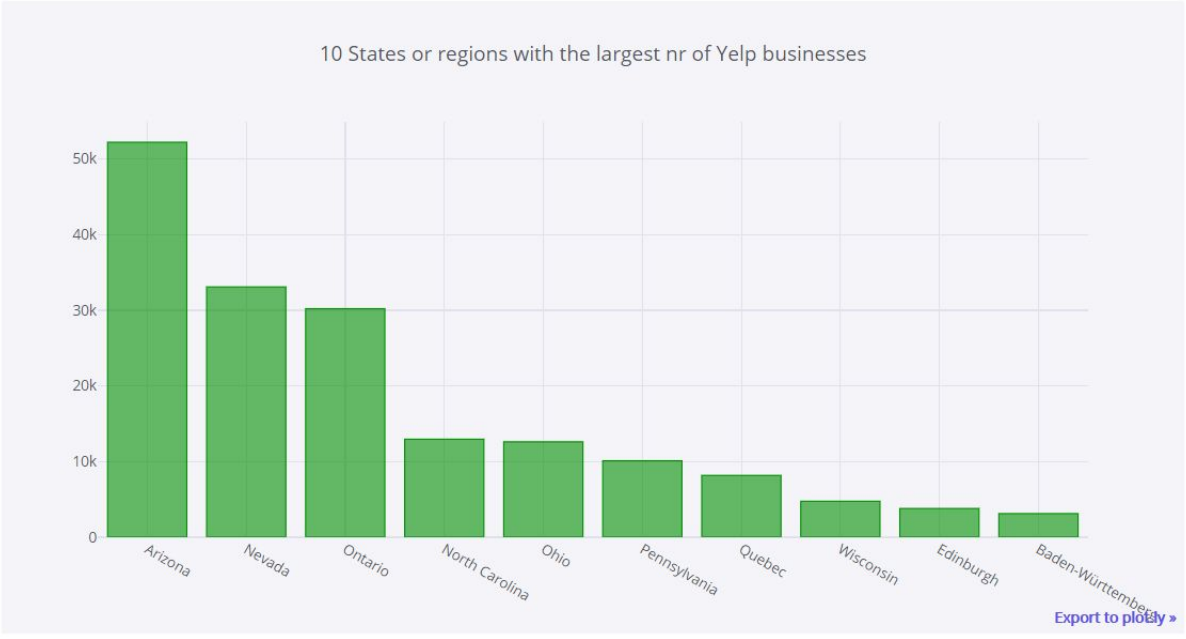
```
state_abbrev = {
    'AL': 'Alabama',
    'AK': 'Alaska',
    'AZ': 'Arizona',
    'QC': 'Quebec',
    'SK': 'Saskatchewan',
    'YT': 'Yukon'
}
```

```
state_names = [state_abbrev[i] for i in topstates.index]
print(state_names)
```

```
['Arizona', 'Nevada', 'Ontario', 'North Carolina', 'Ohio', 'Pennsylvania', 'Quebec', 'Wisconsin', 'Edinburgh', 'Baden-Württemberg']
```

```
topstates["state_names"] = state_names
```

```
topstates.iplot(kind='bar', x='state_names', y='review_count', color="green",
                title = "10 States or regions with the largest nr of Yelp businesses")
```



In this simple way, we can quickly see that most Yelp businesses are located in Arizona, Nevada and Ontario. Beside the USA and Canada, we can also see some businesses from Europe: Edinburgh and Baden-Württemberg.

Yelp on Microsoft Azure Cloud

Microsoft Azure is a cloud service, which is designed for companies that do not have a distributed hardware system that would be available for running distributed processing on them. Instead, Azure offers a full service, including hosting of computational power, predefined software configurations that can be conveniently installed and used on demand. On-demand means that as a user, we can create a cluster, for which we would be charged money according to our usage. After finishing all jobs, the cluster can be deleted, since all created objects, scripts and notebooks, as well as imported data sets, are stored on the Azure Cloud platform as BLOB objects.

Since Microsoft does not provide any students discounts, one of our project members found a smart workaround solution: every group member had to create an account and register for a trial version for a period of 1 month. After the trial of team member A expired, the team member B registered an account on Azure, and so on. This way we managed to avoid any payment and were still able to use all benefits of a distributed cluster, without committing any fraud.

One of the main advantages of Azure was the fact that we could perform a real distributed processing on a cluster of 2 master nodes and around 6-8 slaves and leverage the computational power of at least 50 cores of CPU and about 60-80 GB of RAM.

By using Jupyter Notebooks on Azure, we created several HTML files that include reproducible statistical learning code along with Machine Learning Pipeline objects, data preprocessing and even interactive visualizations by using built-in functionalities of Spark SQL, if one chooses to run the code only on a master node.

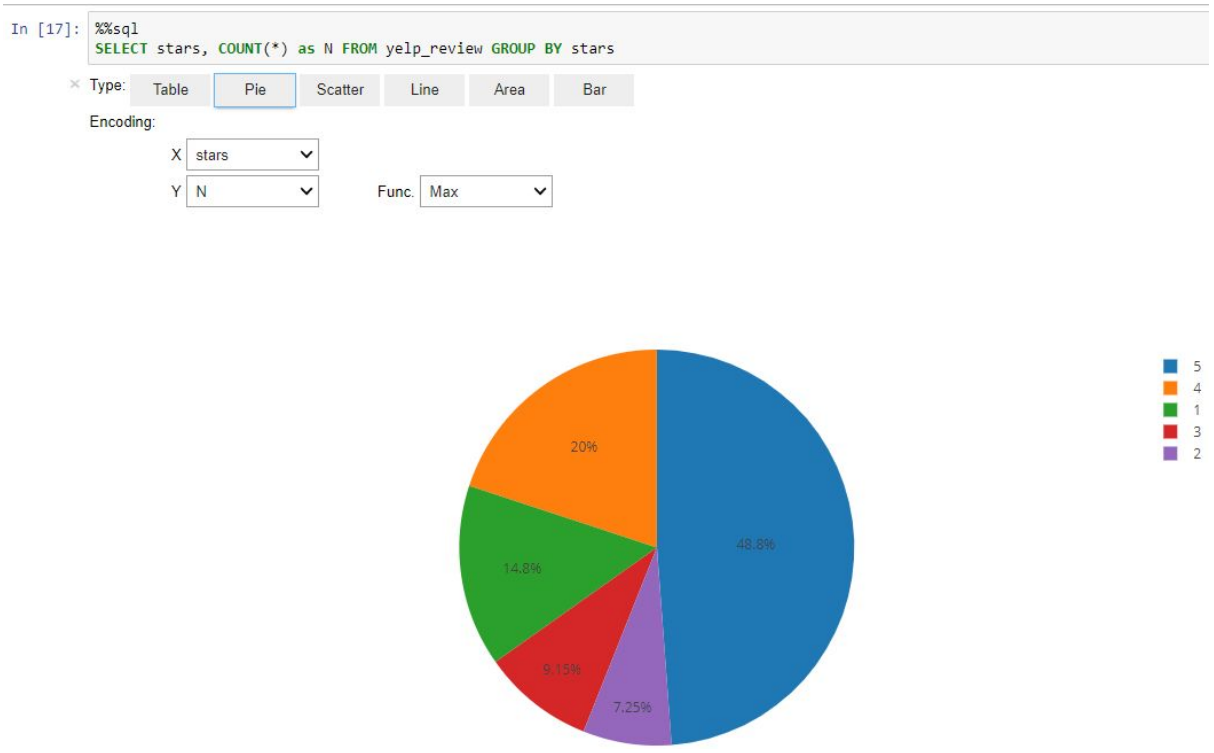
In the following, we present the links to the full HTML files (or Jupyter Notebooks), including Python code, utilizing the capabilities of Pyspark MLlib library. Furthermore, we provide sample code as screenshots across the entire documentation.

1. Initial Review Rating prediction: https://drive.google.com/open?id=1DX0svaBw36IhDmaVriKtWVfDOeP8ecD_
2. Attempt to improve current Review Rating prediction by introducing new features: https://drive.google.com/open?id=1Wnl3Xw9ajYC5F7SmlpxksCS7MxPouxzs
3. Attempt of training neural network on Big Data from Yelp: https://drive.google.com/open?id=1X8TD0tGv_Ze8y6AHaVua9bTR3-sgui6P
4. Python Code performing analysis of Yelp Businesses, including interactive Plotly maps: https://drive.google.com/open?id=1UZ5lYeWtyghpRtUnbThZqtSBpCIVwtfa, https://drive.google.com/open?id=1ibRmzfowYCRjhU_11cz4W1nb6ZIHq4rN
5. Jupyter notebook and html file for LDA clustering can be found here: https://drive.google.com/open?id=1-DN_bOf_9-uMM6DTj2BlGW9w0HcHPdG5 https://drive.google.com/open?id=1OyrQr07czZJkS8B_EsU9k-VbodXvNa5x

PDF version of visualization in Tableau and respective twb workbook can be found here:
https://drive.google.com/open?id=1lEaySS7yHgNpThKIAM7LoWsv9l_8o_U-
https://drive.google.com/open?id=1PHoUZ6uIs2GqcIpuhZ9U81G3aWF4wXBY

Star rating prediction

One of the most crucial elements on Yelp is that users rate businesses and that they assign a star rating to each of them. In the following pie chart, we demonstrate the star rating distribution of all reviews. By looking at it, we can infer that most users tend to give either very favorable rating, such as 5 or 4 stars, or a very negative one, such as 1 star.



In the following section, we are going to create a classification model on our Spark cluster, by using Pyspark, in order to predict one of the five classes of star rating for each review, given the textual data from Yelp.

Business value

Overall, it may not be obvious what are the benefits of such a model: Why would it be useful? What is the business value of that? Many websites like Amazon or Yelp have millions of reviews, and it's impossible for the user to read them all. Generally, readers prefer to look only at the star rating and ignore the text. However, there are many sources of customer feedback that do not have a star rating assigned to them, for instance, the company's website or feedback cards in the coffee shop or the restaurant. In those cases, Review Rating Prediction is extremely useful, because the independent restaurants can incorporate the customer’s feedback from all sources and use our model to predict the star rating of each of them, which allows to aggregate data and use it to guide the business decisions for future improvement and increase in revenues.

Implementation

Traditional machine learning environments that run on a local machine, such as Python or R, would struggle to process 9 GB of data from Yelp. It could take hours or even days to fit the model. For more efficient processing, we used Pyspark MLlib library on a distributed Spark cluster. This way, we could train many different classifiers and compare their performance. It turned out that a properly tuned Naive Bayes model gives the best performance.

Naive Bayes

Naive Bayes is a very "naive" classifier because it assumes that given a specific class, the features are independent of each other, which is not true as words in the review create meaning only through combinations with nearby words. However, this assumption works well in many situations, among others when we want to assign a star rating to each review! There is one pitfall in using Naive Bayes: since we assume that variables are independent of each other, we calculate the posterior probability by multiplying all constituents. Therefore, if only one element in this multiplication equals precisely zero, we would obtain an inaccurate result. To avoid this issue, we set the smoothing parameter to 2, which means that we use Beta distribution as our prior to prevent from obtaining a posterior probability of zero.

In the following, we present and explain the code of this model.

1. We read the data from Blob storage on Azure and create a Spark Session.

Predict stars rating based on the text of the review and its length

```
review = spark.read.csv('wasb://yelp-main@yelpdata.blob.core.windows.net/yelp_review_tab.csv', sep='\t', header=True, inferSchema
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1527884924519_0004	pyspark3	idle	Link	Link	✓

```
SparkSession available as 'spark'.
```

```
review.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- review_id: string (nullable = true)
 |-- user_id: string (nullable = true)
 |-- business_id: string (nullable = true)
 |-- stars: integer (nullable = true)
 |-- date: timestamp (nullable = true)
 |-- text: string (nullable = true)
 |-- useful: integer (nullable = true)
 |-- funny: integer (nullable = true)
 |-- cool: integer (nullable = true)
```

2. We create additional feature: text length.

```
import pyspark.sql.functions as F
review = review.withColumn('text_length', F.length('text')) # add text length column
```

3. We remove NA values simply by using WHERE … NOT NULL in our query.

```
review.createOrReplaceTempView("review")
spark.sql("SELECT stars, text, text_length FROM review LIMIT 5").show(truncate = 80)
```

```
+-----+--------------------------------------------------------------------------------+-----------+
|stars|                                                                            text|text_length|
+-----+--------------------------------------------------------------------------------+-----------+
|    5|Super simple place but amazing nonetheless. It's been around since the 30's a...|        175|
| null|                                                                            null|       null|
|    5|Small unassuming place that changes their menu every so often. Cool decor and...|        140|
| null|                                                                            null|       null|
| null|                                                                            null|       null|
+-----+--------------------------------------------------------------------------------+-----------+
```

```
review = review.filter("text is not NULL and stars is not NULL and text_length is not NULL and categories is not NULL \
                        and useful is not null and funny is not null and cool is not null \
                       and useful >=0 and funny >=0 and cool >=0")

review.createOrReplaceTempView("review")
spark.sql("SELECT stars, text, text_length, categories, useful, funny, cool FROM review LIMIT 5").show(truncate = 40)
```

```
+-----+----------------------------------------+-----------+----------------------------------------+------+-----+----+
|stars|                                    text|text_length|                              categories|useful|funny|cool|
+-----+----------------------------------------+-----------+----------------------------------------+------+-----+----+
|    4|I've always enjoyed my time at brick ...|        202|American (New);Nightlife;Bars;Sandwic...|     0|    0|   0|
|    2|1st time here. Came w my Unc bc Louie...|        302|American (New);Nightlife;Bars;Sandwic...|     0|    0|   0|
|    1|Worse service ever andI use to be a s...|        546|American (New);Nightlife;Bars;Sandwic...|     0|    0|   0|
|    4|I enjoyed this place.  I went the nig...|        314|American (New);Nightlife;Bars;Sandwic...|     0|    0|   0|
|    2|First time there tonight. My boyfrien...|        867|American (New);Nightlife;Bars;Sandwic...|     2|    0|   0|
+-----+----------------------------------------+-----------+----------------------------------------+------+-----+----+
```

4. We found that high-rated reviews are shorter: text_length turned out to be a useful feature.

```
stars = review.groupBy("stars").mean("text_length")
stars.show()
```

```
+-----+------------------+
|stars|  avg(text_length)|
+-----+------------------+
|    1| 453.4136249714574|
|    3| 321.6780794208107|
|    5|305.04888994016454|
|    4| 300.0155511740221|
|    2|  377.554903334619|
+-----+------------------+
```

5. First try fitting the model with smooting parameter set to 1, which means that we use Laplace smoothing to avoid posterior probability of zero. This model is later referenced as “Naive Bayes 1”.

```
tokenizer = Tokenizer(inputCol="text", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
count_vec = CountVectorizer(inputCol='stop_tokens',outputCol='c_vec')
idf = IDF(inputCol="c_vec", outputCol="tf_idf")

tokenizer2 = Tokenizer(inputCol="categories", outputCol="token_cat")
stopremove2 = StopWordsRemover(inputCol='token_cat',outputCol='stop_cat')
count_vec2 = CountVectorizer(inputCol='stop_cat',outputCol='cat_vec')
idf2 = IDF(inputCol="cat_vec", outputCol="tf_idf_cat")


stars_to_label = StringIndexer(inputCol='stars', outputCol='label')
feature_vector = VectorAssembler(inputCols=['tf_idf','tf_idf_cat', 'text_length', 'useful', 'funny', 'cool'],
                                 outputCol='features')

nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
# smooting = 2 is equivalent to using Beta distr. as conjugate prior for parameters of Binomial distribution
```

```
# Create a pipeline
data_prep_pipe = Pipeline(stages=[stars_to_label, tokenizer, stopremove, count_vec, idf,
                                  tokenizer2, stopremove2, count_vec2, idf2, feature_vector])
cleaner = data_prep_pipe.fit(review)
clean_data = cleaner.transform(review)
clean_data = clean_data.select(['label','features'])

(train, test) = clean_data.randomSplit([0.7,0.3], 1234)
stars_predictor = nb.fit(train)
predictions = stars_predictor.transform(test)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
acc_eval = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
acc_nb = acc_eval.evaluate(predictions)
print("Accuracy of the model at predicting stars rating is: {}".format(acc_nb))
```

```
Accuracy of the model at predicting stars rating is: 0.5840904054952359
```

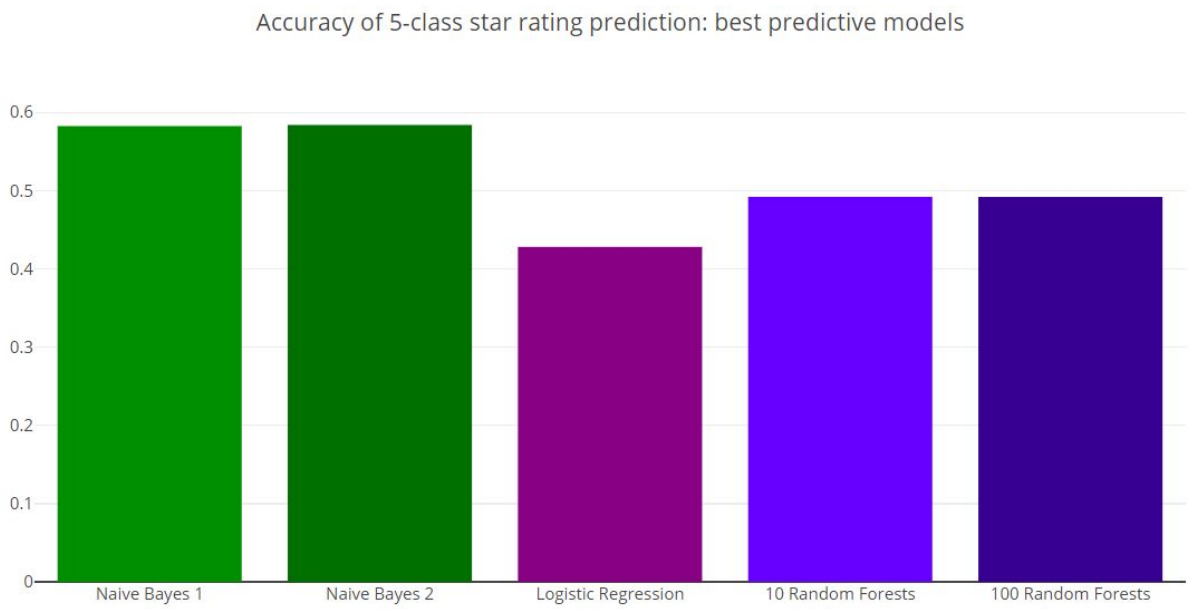
6. Second try fitting the model with smooting parameter set to 2, which means that we use Beta distribution as our prior to prevent from posterior probability of 0. This model is later referenced as “Naive Bayes 2”.

```
tokenizer = Tokenizer(inputCol="text", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
count_vec = CountVectorizer(inputCol='stop_tokens',outputCol='c_vec')
idf = IDF(inputCol="c_vec", outputCol="tf_idf")

tokenizer2 = Tokenizer(inputCol="categories", outputCol="token_cat")
stopremove2 = StopWordsRemover(inputCol='token_cat',outputCol='stop_cat')
count_vec2 = CountVectorizer(inputCol='stop_cat',outputCol='cat_vec')
idf2 = IDF(inputCol="cat_vec", outputCol="tf_idf_cat")


stars_to_label = StringIndexer(inputCol='stars', outputCol='label')
feature_vector = VectorAssembler(inputCols=['tf_idf','tf_idf_cat', 'text_length', 'useful', 'funny', 'cool'],
                                 outputCol='features')

nb = NaiveBayes(smoothing=2.0, modelType="multinomial")
# smooting = 2 is equivalent to using Beta distr. as conjugate prior for parameters of Binomial distribution

# Create a pipeline
data_prep_pipe = Pipeline(stages=[stars_to_label, tokenizer, stopremove, count_vec, idf,
                                  tokenizer2, stopremove2, count_vec2, idf2, feature_vector])
cleaner = data_prep_pipe.fit(review)
clean_data = cleaner.transform(review)
clean_data = clean_data.select(['label','features'])

(train, test) = clean_data.randomSplit([0.7,0.3], 1234)
stars_predictor = nb.fit(train)
predictions = stars_predictor.transform(test)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
acc_eval = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
acc_nb = acc_eval.evaluate(predictions)
print("Accuracy of the model at predicting stars rating is: {}".format(acc_nb))
```

```
Accuracy of the model at predicting stars rating is: 0.5879305661779948
```

The above model is simple, though it outperformed logistic regression and random forest. In the bar chart below, we contrast the performance of:

1) Naive Bayes 1 - Naive Bayes with smoothing parameter set to 1,
2) Naive Bayes 2 - Naive Bayes with smoothing parameter set to 2,
3) Logistic Regression,
4) Random Forest with 10 trees,
5) Random Forest with 100 trees.



Other classifiers

In the following, we briefly present the code of other classifiers, mentioned in the bar chart. All of them require the following preprocessing steps:

```
tokenizer = Tokenizer(inputCol="text", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
count_vec = CountVectorizer(inputCol='stop_tokens',outputCol='c_vec')
idf = IDF(inputCol="c_vec", outputCol="tf_idf")
stars_to_label = StringIndexer(inputCol='stars', outputCol='label')
feature_vector = VectorAssembler(inputCols=['tf_idf','text_length', 'useful', 'funny', 'cool'], outputCol='features')
```

1. Logistic Regression

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Create a pipeline
data_prep_pipe = Pipeline(stages=[stars_to_label, tokenizer, stopremove, count_vec, idf, clean_up])
cleaner = data_prep_pipe.fit(review) # error
clean_data = cleaner.transform(review)
clean_data = clean_data.select(['label','features'])

(train, test) = clean_data.randomSplit([0.7,0.3], 1234)
stars_predictor = lr.fit(train)
predictions = stars_predictor.transform(test)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
acc_eval = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
acc_logistic = acc_eval.evaluate(predictions)
print("Accuracy of the model at predicting stars rating is: {}".format(acc_logistic))
```

```
Accuracy of the model at predicting stars rating is: 0.4279770940355125
```

2. Random Forest with 10 trees

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=10)

(train, test) = clean_data.randomSplit([0.7,0.3], 1234)
stars_predictor = rf.fit(train)
predictions = stars_predictor.transform(test)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
acc_eval = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
acc = acc_eval.evaluate(predictions)
print("Accuracy of the model at predicting stars rating is: {}".format(acc))
```

```
Accuracy of the model at predicting stars rating is: 0.4919464609800363
```

3. Random Forest with 100 trees

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=100)

(train, test) = clean_data.randomSplit([0.7,0.3], 1234)
stars_predictor = rf.fit(train)
predictions = stars_predictor.transform(test)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
acc_eval = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
acc = acc_eval.evaluate(predictions)
print("Accuracy of the model at predicting stars rating is: {}".format(acc))
```

```
Accuracy of the model at predicting stars rating is: 0.49183303085299457
```

Further improvements & limitations

We did not stop after fitting just 5 models. We tried the same models by using different parameters, for instance:

- Random Forest with a different number of trees → it turned out that the model improves by fitting an ensemble of trees, instead of fitting a single large classification tree. However, as we can see in the prediction score, more trees do not guarantee better splits, and hence, better performance. The changes in prediction accuracy were insignificant.
- Logistic regression with different parameters than *maxIter=10, regParam=0.3, elasticNetParam=0.8* → also here, the changes in performance were tiny.

Moreover, we trained further models that make use of different features from other tables, for instance, by merging the table *review* with the table *business* in order to append the column categories:

```
# keep only relevant columns:
final_df = final[["stars", "categories", "text", "useful", "funny", "cool"]]
final_df.head(1)
```

	stars	categories	text	useful	funny	cool
0	4	American (New);Nightlife;Bars;Sandwiches;American (Traditional);Burgers;Restaurants	I've always enjoyed my time at brick house food and cocktails never disappoint! Its perfect for after work in the summer time. The staff is always polite and the decore is beautiful, a must place to be!	0	0	0

```
final_df.to_csv("final_df.csv", sep="\t")
```

Nevertheless, it turned out that additional features did not improve our model.

Furthermore, we trained a Spark-specific version of a neural network, which is called a multilayer perceptron classifier:

```
tokenizer = Tokenizer(inputCol="text", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
count_vec = CountVectorizer(inputCol='stop_tokens',outputCol='c_vec')
idf = IDF(inputCol="c_vec", outputCol="tf_idf")
stars_to_label = StringIndexer(inputCol='stars', outputCol='label')
feature_vector = VectorAssembler(inputCols=['tf_idf','text_length', 'useful', 'funny', 'cool'], outputCol='features')
```

```
# specify layers for the neural network:
# input layer of size 5 (features), two intermediate (=hidden) layers of size 5 and 4
# and output of size 5 (classes)
layers = [5, 5, 4, 5]
# create the trainer and set its parameters
FNN = MultilayerPerceptronClassifier(labelCol="label", featuresCol="features",
                                     maxIter=100, layers=layers, blockSize=128, seed=1234)
```

However, it turned out that it is computationally difficult to train this model. We encounter following errors:

```
# Create a pipeline
data_prep_pipe = Pipeline(stages=[stars_to_label, tokenizer, stopremove, count_vec, idf, feature_vector])
cleaner = data_prep_pipe.fit(review)
clean_data = cleaner.transform(review)
clean_data = clean_data.select(['label','features'])
```

```
(train, test) = clean_data.randomSplit([0.7,0.3], 1234)
stars_predictor = FNN.fit(train)
predictions = stars_predictor.transform(test)
```

```
# compute accuracy on the test set
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
predictionAndLabels = predictions.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))
```

```
An error occurred while calling o594.evaluate.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 15 in stage 688.0 failed 4 times, most recent failur
e: Lost task 15.3 in stage 688.0 (TID 7001, wn0-anna-s.u41e5xgde5cuzenjchz0ltftrg.bx.internal.cloudapp.net, executor 5): org.ap
ache.spark.SparkException: Failed to execute user defined function($anonfun$1: (vector) => double)
        at org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIterator.processNext(Unknown Source)
        at org.apache.spark.sql.execution.BufferedRowIterator.hasNext(BufferedRowIterator.java:43)
        at org.apache.spark.sql.execution.WholeStageCodegenExec$$anonfun$8$$anon$1.hasNext(WholeStageCodegenExec.scala:395)
        at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:408)
        at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:408)
        at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:408)
        at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:408)
        at org.apache.spark.util.collection.ExternalSorter.insertAll(ExternalSorter.scala:193)
        at org.apache.spark.shuffle.sort.SortShuffleWriter.write(SortShuffleWriter.scala:63)
        at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:96)
        at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:53)
        at org.apache.spark.scheduler.Task.run(Task.scala:108)
        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:338)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.IllegalArgumentException: requirement failed: A & B Dimension mismatch!
```

```
An error occurred while calling o567.fit.
: java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOf(Arrays.java:3236)
        at java.io.ByteArrayOutputStream.grow(ByteArrayOutputStream.java:118)
        at java.io.By
ava:93)
        at java.io.By
        at org.apache
tream.scala:41)
        at java.io.ObjectOutputStream$BlockDataOutputStream.drain(ObjectOutputS
tream.java:1877)
        at java.io.ObjectOutputStream$BlockDataOutputStream.setBlockDataMode(Ob
jectOutputStream.java:1786)
```

java.lang.OutOfMemoryError

The above errors indicate that we ran out of memory and one possible reason is that the dataset is large (3.5 GB). Given that we perform TF-IDF and boolean one-hot encoding for words, it is possible that it was too computationally expensive for a neural network to calculate all matrix multiplications and split the work across the cluster (56 cores, 56 GB

RAM x 6 slaves). We tried a workaround solution by fitting the same model on a small subset of data:

```
import pandas as pd
df = pd.read_csv('yelp_review_tab.csv', sep = "\t", chunksize = 100000) # tab sep small subset, only 100.000 rows
review = next(df)
```

```
review[["stars", "text", "useful", "funny", "cool"]].head()
```

	stars	text	useful	funny	cool
0	5	Super simple place but amazing nonetheless. It...	0	0	0
1	5	Small unassuming place that changes their menu...	0	0	0
2	5	Lester's is located in a beautiful neighborhoo...	0	0	0
3	4	Love coming here. Yes the place always needs t...	0	0	0
4	4	Had their chocolate almond croissant and it wa...	0	0	0

```
review = review[["stars", "text", "useful", "funny", "cool"]]
```

```
review.to_csv('yelp_review_tab_subset.csv', sep = "\t")
```

```
from pyspark.ml.feature import Tokenizer, StopWordsRemover, CountVectorizer, IDF, StringIndexer, VectorAssembler
from pyspark.ml.linalg import Vector
from pyspark.ml.classification import MultilayerPerceptronClassifier # FNN = Feedforward Neural Network
from pyspark.ml import Pipeline
import pyspark.sql.functions as F

# Load and parse the data file, converting it to a DataFrame.
review = spark.read.csv('wasb://yelp@yelphwr.blob.core.windows.net/yelp_review_tab_subset.csv',
                        sep='\t', header=True, inferSchema=True)
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1528648663186_0004	pyspark3	idle	Link	Link	✓

```
SparkSession available as 'spark'.
```

```
review = review.withColumn('text_length', F.length('text')) # add text length column
review = review.filter("text is not NULL and stars is not NULL and text_length is not NULL \
                        and useful is not null and funny is not null and cool is not null \
                       and useful >=0 and funny >=0 and cool >=0")

review.createOrReplaceTempView("review")
```

```
tokenizer = Tokenizer(inputCol="text", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
count_vec = CountVectorizer(inputCol='stop_tokens',outputCol='c_vec')
idf = IDF(inputCol="c_vec", outputCol="tf_idf")
stars_to_label = StringIndexer(inputCol='stars', outputCol='label')
feature_vector = VectorAssembler(inputCols=['tf_idf','text_length', 'useful', 'funny', 'cool'], outputCol='features')
```

```
# specify layers for the neural network:
# input layer of size 5 (features), two intermediate (=hidden) layers of size 5 and 4
# and output of size 5 (classes)
layers = [5, 5, 4, 5]
# create the trainer and set its parameters
FNN = MultilayerPerceptronClassifier(labelCol="label", featuresCol="features",
                                     maxIter=100, layers=layers, blockSize=128, seed=1234)
```

```
data_prep_pipe = Pipeline(stages=[stars_to_label, tokenizer, stopremove, count_vec, idf, feature_vector])
cleaner = data_prep_pipe.fit(review)
clean_data = cleaner.transform(review)
clean_data = clean_data.select(['label','features'])
```

```
(train, test) = clean_data.randomSplit([0.7,0.3], 1234)
stars_predictor = FNN.fit(train)
predictions = stars_predictor.transform(test)
```

```
# compute accuracy on the test set
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
predictionAndLabels = predictions.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))
```

```
An error occurred while calling o330.evaluate.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 15 in stage 456.0 failed 4 times, most recent failu
re: Lost task 15.3 in stage 456.0 (TID 4611, wn6-anna-s.u41e5xgde5cuzenjchz0ltftrg.bx.internal.cloudapp.net, executor 3): or
g.apache.spark.SparkException: Failed to execute user defined function($anonfun$1: (vector) => double)
        at org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIterator.processNext(Unknown Source)
        at org.apache.spark.sql.execution.BufferedRowIterator.hasNext(BufferedRowIterator.java:43)
        at org.apache.spark.sql.execution.WholeStageCodegenExec$$anonfun$8$$anon$1.hasNext(WholeStageCodegenExec.scala:395)
        at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:408)
        at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:408)
        at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:408)
        at scala.collection.Iterator$$anon$11.hasNext(Iterator.scala:408)
        at org.apache.spark.util.collection.ExternalSorter.insertAll(ExternalSorter.scala:193)
        at org.apache.spark.shuffle.sort.SortShuffleWriter.write(SortShuffleWriter.scala:63)
        at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:96)
        at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:53)
        at org.apache.spark.scheduler.Task.run(Task.scala:108)
        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:338)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
```

Another error that we once got:

```
(train, test) = clean_data.randomSplit([0.7,0.3], 1234)
stars_predictor = FNN.fit(train)
predictions = stars_predictor.transform(test)
```

```
An error was encountered:
Error sending http request and maximum retry encountered.
```

We did not dig deeper into this problem becaue we benchmarked our simpler models, such as Naive Bayes 2 with very complex neural networks that some people trained on Yelp dataset. It turned out that it is hard to achieve better accuracy than 62 percent, even by using a highly complicated and computationally expensive neural network[5][6][7][8]. Therefore we decided to follow the Occam's razor principle and use the simpler model, Naive Bayes 2, which:

- can be trained very quickly on a new dataset,
- has a sufficient accuracy given that we have five classes of rating to predict,
- does not overfit and is very effective.

[5] https://github.com/i008/nyyelp
[6] https://github.com/thomasan95/Yelp-Review-Prediction
[7] https://arxiv.org/abs/1712.04350
[8] http://www.developintelligence.com/blog/2017/03/predicting-yelp-star-ratings-review-text-python/

Customer preferences via LDA clustering

What are the trending cuisine and food in different cities? What are the patterns in customer preferences? What are the top topics people in custores reviews across the cities?
It is another useful insight for restaurants. Answers to these questions will be given by applying LDA clustering on yelp reviews.

Loading data

For further analysis two tables are required: yelp_review and yelp_business.

```
yelp_review = spark.read.csv('wasb://yelp@yelphwr.blob.core.windows.net/yelp_review_tab.csv', sep ='\t', header=True, \
                             mode="DROPMALFORMED", inferSchema=True)
yelp_review.write.saveAsTable("yelp_review", mode="overwrite")
yelp_review.printSchema()
```

```
yelp_business = spark.read.csv('wasb://yelp@yelphwr.blob.core.windows.net/yelp_business.csv', sep =',', header=True, \
                               mode="DROPMALFORMED", inferSchema=True)
yelp_business.write.saveAsTable("yelp_business", mode="overwrite")
yelp_business.printSchema()
```

Subsetting data

All in all there are 127 210 businesses in yelp_business table. We need only restaurants for further analysis:

```
# Subset all restaurants
yelp_restaurants = yelp_business.filter(yelp_business.categories.rlike('Food|Restaurants|Bars|Bakeries'))
yelp_restaurants.registerTempTable('yelp_restaurants')
yelp_restaurants.count()
```

```
58588
```

For a proof of concept, we will run LDA on reviews from 3 cities.

```
%%sql
/* What are the top 3 cities with highest number of restourants? */
select city, count(*) as N from yelp_restaurants group by city order by N DESC  LIMIT 3
```

city	N
Toronto	8525
Las Vegas	5425
Montréal	3957

Subset the data from ‘yelp_restaraunts’ for Toronto, Las Vegas and Montreal:

```
subset_toronto = spark.sql("select text, review_id, date from yelp_review where business_id IN (select business_id \
from yelp_restaurants where city IN ('Toronto'))")
subset_toronto.registerTempTable('subset_toronto')
# Number of reviews for Toronto
subset_toronto.count()
```

```
136201
```

```
subset_las_vegas = spark.sql("select text, review_id, date from yelp_review where business_id IN (select business_id \
from yelp_restaurants where city IN ('Las Vegas'))")
subset_las_vegas.registerTempTable('subset_las_vegas')
# Number of reviews for Las Vegas
subset_las_vegas.count()
```

```
384761
```

```
subset_montreal = spark.sql("select text, review_id, date from yelp_review where business_id IN (select business_id \
from yelp_restaurants where city IN ('Montréal'))")
subset_montreal.registerTempTable('subset_montreal')
# Number of reviews for Montreal
subset_montreal.count()
```

```
53412
```

Data preparation & LDA clustering

In order to process data for LDA clustering the following steps are required:

1. Word tokenization
2. Removing stop words
3. CountVectorizer is used to convert a collection of text documents to vectors of token counts. During the fitting process, CountVectorizer will select the top 2000 vocabSize words ordered by term frequency across the corpus.
4. TFIDF. We use IDF to rescale the feature vectors. Afterwards, these feature vectors are passed to the LDA model.

```
import pandas as pd
cities_df = pd.DataFrame()

cities = ['Toronto', 'Las Vegas', 'Montréal']
datasets = [subset_toronto, subset_las_vegas, subset_montreal]
for c in range(len(cities)):
    city = cities[c]
    dataset = datasets[c]

    tokenizer = RegexTokenizer(inputCol='text', outputCol='tokenized', pattern='\s+|[,."';()]')
    featurizedData0 = tokenizer.transform(dataset)

    stopwords = StopWordsRemover(inputCol=tokenizer.getOutputCol(), outputCol='words')
    featurizedData1 = stopwords.transform(featurizedData0)

    # Term Frequency Vectorization  - Option 2 (CountVectorizer):
    cv = CountVectorizer(inputCol="words", outputCol="rawFeatures", vocabSize = 2000)
    cvmodel = cv.fit(featurizedData1)
    featurizedData = cvmodel.transform(featurizedData1)

    vocab = cvmodel.vocabulary
    vocab_broadcast = sc.broadcast(vocab)

    # TFIDF
    idf = IDF(inputCol="rawFeatures", outputCol="features")
    idfModel = idf.fit(featurizedData)
    rescaledData = idfModel.transform(featurizedData)
```

Next, we fit LDA model with 10 topics and optimizer online variational inference. Afterwards we extract the topics with corresponding terms and probabilities and transform the results into Spark DataFrame.

```
# Generate 10 topics for each city
lda = LDA(k=10, seed=1234,  optimizer='online', featuresCol='features')
ldamodel = lda.fit(rescaledData)

# Load topics
topicIndices = ldamodel.describeTopics(maxTermsPerTopic=5)
vocablist = cvmodel.vocabulary

# Preprocess model resuts
topics_words = topicIndices.rdd\
   .map(lambda row: row['termIndices'])\
   .map(lambda idx_list: [vocab[idx] for idx in idx_list])\  # map tokens from vocabulary
   .collect()

weights_map = topicIndices.rdd\
       .map(lambda row: row['termWeights']).collect()
list(weights_map)

# Merge terms and weights into one dataframe
df_topics = pd.DataFrame(
    {'Terms': topics_words,
     'Weights': weights_map
    })

# Add additional inforamtion: City and Number of Topic
for i in range(len(df_topics)):
    for j in range(len(df_topics['Terms'][i])):
        row = [{'Topic': i+1, 'Term': df_topics['Terms'][i][j], 'Weight': df_topics['Weights'][i][j], 'City': city}]
        cities_df = cities_df.append(row, ignore_index=True)
```

```
# Create a new Spark DataFrame
schema = StructType([StructField('City', StringType(), True), StructField('Term', StringType(), True), \
                     StructField('Topic', IntegerType(), True), StructField('Weight', IntegerType(), True)])
cities_DF = spark.createDataFrame(sc.emptyRDD(), schema)
```

```
# Convert Pandas DataFrame into Spark DataFrame (in pandas version > 0.19 it is done in one line)
for i in range(len(cities_df)):
    newRow = spark.createDataFrame([Row(City=cities_df.iloc[i]['City'], Topic=cities_df.iloc[i]['Topic'].tolist(),\
                                        Term=cities_df.iloc[i]['Term'], Weight=cities_df.iloc[i]['Weight'].tolist() )])
    cities_DF = cities_DF.union(newRow)
```

The resulting table is represented below:

```
In [10]: pd.options.display.max_rows=300
         print(cities_df)
```

	City	Term	Topic	Weight
0	Toronto	pizza	1	0.016716
1	Toronto	ramen	1	0.010416
2	Toronto	eggs	1	0.008036
3	Toronto	bacon	1	0.006985
4	Toronto	fries	1	0.006122
5	Toronto	coffee	2	0.012492
6	Toronto	tea	2	0.011473
7	Toronto	store	2	0.006716
8	Toronto	get	2	0.005462
9	Toronto	place	2	0.005394
10	Toronto	food	3	0.006700

As the last step, we save the resulting table into the cluster. So, it will be possible to load it with Tableau.

```
cities_DF.write.saveAsTable("cities_DF", mode="overwrite")
cities_DF.count()
```

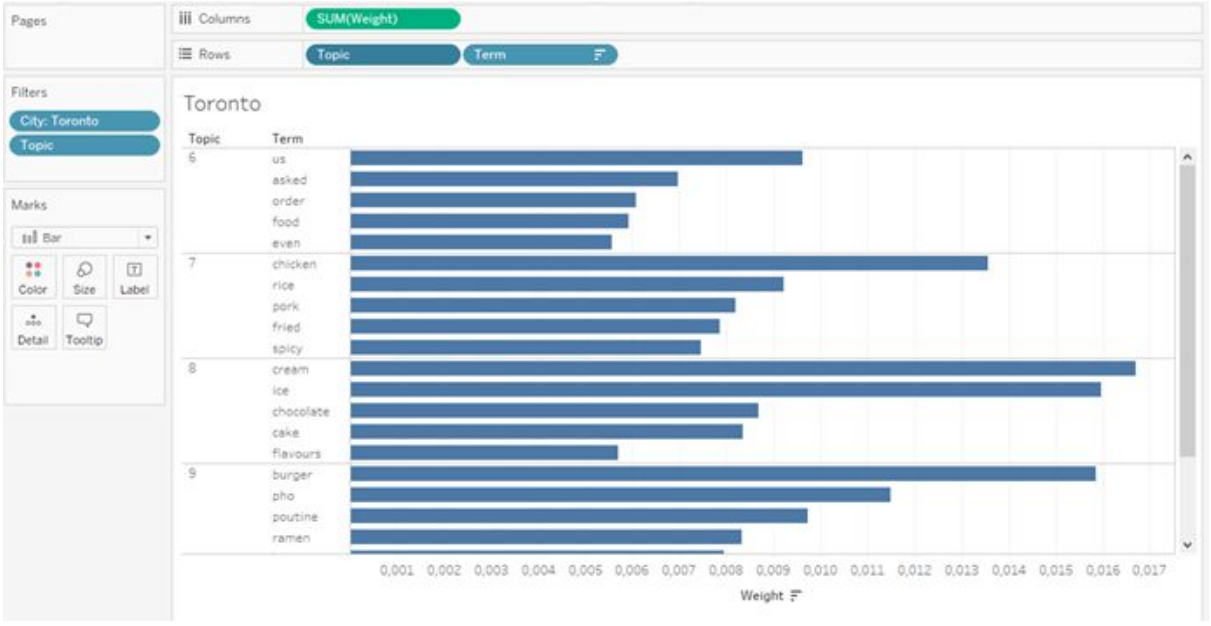
```
150
```

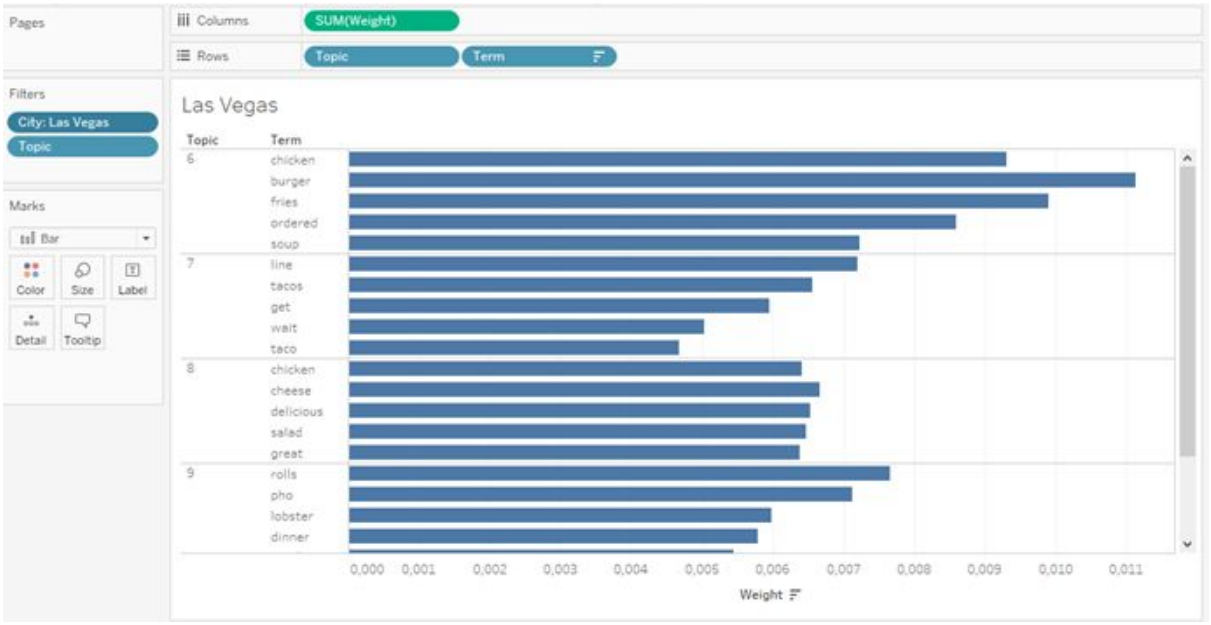
```
%%sql
show tables
```

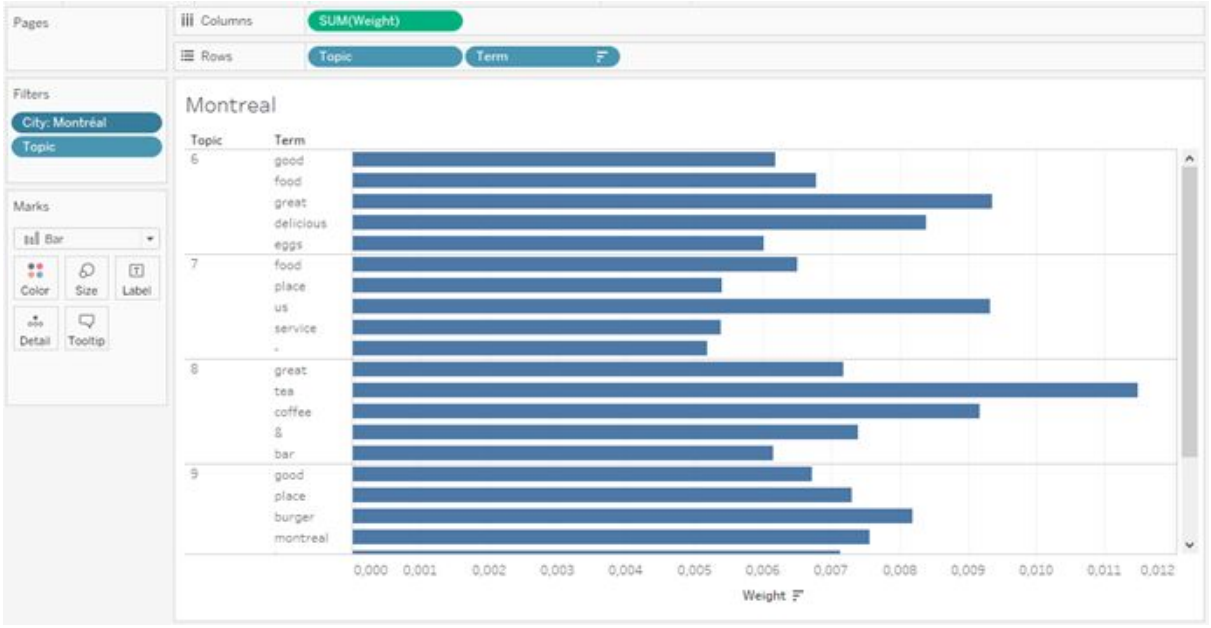
	database	tableName	isTemporary
0	default	cities_df	False
1	default	hivesampletable	False
2	default	yelp_business	False
3	default	yelp_review	False
4		subset_las_vegas	True
5		subset_montreal	True
6		subset_toronto	True
7		yelp_restaurants	True

Tableau visualisations

Here we can see the final visualizations in Tableau. Instruction how to connect Tableau and Spark cluster can be found in Appendix 2.







LDA clustering on business categories

The first LDA model in this project was applied on yelp business categories table in order to cluster them into groups.

1. Is there any null values in yelp_review?

```
In [4]: # Is there any null values in yelp_review?
yelp_business.select([count(when(isnull(c), c)).alias(c) for c in yelp_business.columns]).show()
```

```
+-----------+----+------------+-------+----+-----+-----------+--------+---------+-----+------------+-------+----------+
|business_id|name|neighborhood|address|city|state|postal_code|latitude|longitude|stars|review_count|is_open|categories|
+-----------+----+------------+-------+----+-----+-----------+--------+---------+-----+------------+-------+----------+
|          0|   0|       75939|      0|   1|    0|        590|       1|        1|    0|           0|      0|         0|
+-----------+----+------------+-------+----+-----+-----------+--------+---------+-----+------------+-------+----------+
```

2. Columns categories, business_id and name don't have NA values. Let's select these three columns and register a temporary table in spark cluster.

```
In [6]: # Subset the data for LDA
BusinessData = yelp_business.select(yelp_business.categories, yelp_business.business_id, yelp_business.name)
BusinessData.registerTempTable('BusinessData')
```

```
In [7]: %%sql
SHOW TABLES
```

Type: Table Pie Scatter Line Area Bar

database	tableName	isTemporary
default	hivesampletable	False
default	yelp_business	False
	businessdata	True

3. Data preprocessing steps:

```
In [8]: # First step - tokenize
tokenizer = RegexTokenizer(inputCol='categories', outputCol='tokenized', pattern='\s+|[,."();]')
featurizedData = tokenizer.transform(BusinessData)
featurizedData.head(5)
```

```
[Row(categories="Hair Stylists;Hair Salons;Men's Hair Salons;Blow Dry/Out Services;Hair Extensions;Beauty & Spas", business_id
='He-G7vWjzVUysIKrfNbPUQ', name='"""Stephen Szabo Salon"""', tokenized=['hair', 'stylists', 'hair', 'salons', "men's", 'hair',
'salons', 'blow', 'dry/out', 'services', 'hair', 'extensions', 'beauty', '&', 'spas']), Row(categories='American (New);Nightlif
e;Bars;Sandwiches;American (Traditional);Burgers;Restaurants', business_id='PfOCPjBrlQAnz__NXj9h_w', name='"""Brick House Tavern
+ Tap"""', tokenized=['american', 'new', 'nightlife', 'bars', 'sandwiches', 'american', 'traditional', 'burgers', 'restaurant
s']), Row(categories='Italian;Restaurants', business_id='o9eMRCWt5PkpLDE0gOPtcQ', name='"""Messina"""', tokenized=['italian', 'r
estaurants']), Row(categories='Coffee & Tea;Ice Cream & Frozen Yogurt;Food', business_id='EsMcGiZaQuG1OOvL9iUFug', name='"""Any
Given Sundae"""', tokenized=['coffee', '&', 'tea', 'ice', 'cream', '&', 'frozen', 'yogurt', 'food']), Row(categories='Automotiv
e;Auto Detailing', business_id='TGWhGNusxyMaA4kQVBNeew', name='"""Detailing Gone Mobile"""', tokenized=['automotive', 'auto', 'd
etailing'])]
```

```
In [9]: # Second step - remove stopwords
stopwords = StopWordsRemover(inputCol=tokenizer.getOutputCol(), outputCol='stop_removed')
featurizedData1 = stopwords.transform(featurizedData)
featurizedData1.head(1)
```

```
[Row(categories="Hair Stylists;Hair Salons;Men's Hair Salons;Blow Dry/Out Services;Hair Extensions;Beauty & Spas", business_id
='He-G7vWjzVUysIKrfNbPUQ', name='"""Stephen Szabo Salon"""', tokenized=['hair', 'stylists', 'hair', 'salons', "men's", 'hair',
'salons', 'blow', 'dry/out', 'services', 'hair', 'extensions', 'beauty', '&', 'spas'], stop_removed=['hair', 'stylists', 'hair',
'salons', "men's", 'hair', 'salons', 'blow', 'dry/out', 'services', 'hair', 'extensions', 'beauty', '&', 'spas'])]
```

```
# Term Frequency Vectorization  - Option 2 (CountVectorizer)    :
cv = CountVectorizer(inputCol="words", outputCol="rawFeatures", vocabSize = 1000)
cvmodel = cv.fit(featurizedData1)
featurizedData = cvmodel.transform(featurizedData1)

vocab = cvmodel.vocabulary
vocab_broadcast = sc.broadcast(vocab)

idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData) # TFIDF
```

4. Generate a LDA model with 25 topics

```
In [47]: # Generate 25 Topics:
lda = LDA(k=25, seed=1234,  optimizer='online', featuresCol="features")
ldamodel = lda.fit(rescaledData)
```

5. Compute logPerplexity and logLikelihood to asses LDA model

```
In [49]: ll = ldamodel.logLikelihood(rescaledData)
lp = ldamodel.logPerplexity(rescaledData)

print("The lower bound on the log likelihood of the entire corpus: " + str(ll))
print("The upper bound on perplexity: " + str(lp))
```

```
The lower bound on the log likelihood of the entire corpus: -11547023.792451285
The upper bound on perplexity: 4.181459433323344
```

6. Let's check the topics from LDA

```
In [52]: ldatopics = ldamodel.describeTopics()
         # Show the top 25 Topics
         ldatopics.show(25)
```

```
+-----+--------------------+--------------------+
|topic|         termIndices|         termWeights|
+-----+--------------------+--------------------+
|    0|[51, 58, 1, 248, ...|[0.16044835244988...|
|    1|[20, 37, 5, 4, 35...|[0.06655446972733...|
|    2|[120, 99, 129, 21...|[0.06658898135834...|
```

As spark is only working with numerical data, we need to match the id of the terms with the vocabulary generated in the preprocessing step:

```
topics_rdd = topicIndices.rdd
type(topics_rdd)
topics_words = topics_rdd\
       .map(lambda row: row['termIndices'])\
       .map(lambda idx_list: [vocab[idx] for idx in idx_list])\
       .collect()

for index, topic in enumerate(topics_words):
    print("topic: ", index)
    print(topic)
    print("------------------")
```

```
topic:  0
['chinese', 'bars', 'cafes', 'pubs', 'nightlife']
------------------
topic:  1
['arts', 'home', 'garden', 'entertainment', 'shopping']
------------------
topic:  2
['repair', 'gas', 'stations', 'convenience', 'automotive']
------------------
topic:  3
['pet', 'pets', 'delis', 'groomers', 'sandwiches']
```

Sentiment Analysis

Sentiment analysis is a method that is used very common in the area of text mining to develop a model in order to understand the content of the textual data and what people think about some certain entities (i.e. companies, celebrities, topics). We want to build a sentiment analysis model with the reviews data set. Then, we will apply this model with two datasets of Yelp which are Tip and Reviews.

Loading data

We use Pyspark to load the yelp csvs into our cluster, while loading we also transform csvs into Pyspark Sql dataframes so that we can work later on the tables using pyspark.

Cmd 3

```
#Reading in the csv file with pyspark
df = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load("/FileStore/tables/yelp_tip.csv",sep=',')
df.printSchema()
type(df)
```

▸ (1) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [text: string, date: string ... 3 more fields]

Data Preprocessing

As it is usual in text mining, we the tokenize the text, remove stopwords, and create a tf-idf so that we can perform logistic regression later in the sentiment analysis.

Cmd 14

```
from pyspark.ml import *
from pyspark.ml.feature import Tokenizer, RegexTokenizer, StopWordsRemover
#tokenizer and stop word remover
tok = Tokenizer(inputCol="text", outputCol="words")
#stop word remover
stopwordrm = StopWordsRemover(inputCol='words', outputCol='words_nsw')
# Build the pipeline
pipeline = Pipeline(stages=[tok, stopwordrm])
# Fit the pipeline
tokenized = pipeline.fit(train1).transform(train1)
```

Simple Sentiment Analysis

To quickly understand the sentiment of the textual data, first, we will perform a simple sentiment analysis using the csv file which contains negative and positive words and a sentiment score stating whether the word is positive or negative with the values 1,-1.

We first extract the words from the text using tokenization, then we remove any stop words and punctuation from the text. Later, we match the words with the words in our sentiment csv file. Then, we calculate the average sentiment per review. For example let's assume that there are two words in one review with the sentiment csv. One is positive (1) and the other

one is negative(-1) then this review is classified as neutral because the average of the values equals to 0. Finally, we apply this concept for all of the reviews.

Cmd 19

```
#calculating the average and predicted sentiment for each tip
sentiment_prediction_df = word_sentiment_df.\
    groupBy('id').\
    agg(fn.avg('sentiment').alias('avg_sentiment')).\
    withColumn('predicted', fn.when(fn.col('avg_sentiment') > 0, 1.0).otherwise(0.))
sentiment_prediction_df.sort('id').show(5)
```

Sentiment analysis using Logistic Regression

Now, we will use a machine learning algorithm to implement sentiment analysis because the previous concept, which we implemented, is a primitive one. We follow the steps that we followed in simple sentiment analysis model for data preprocessing, but this time we will create a Term Frequency- Inverse Document Frequency(TF-IDF) in order to be able to apply logistic regression.

Cmd 24

```
#Reading in the csv file with pyspark(reviews)
review = sqlContext.read.format("com.databricks.spark.csv").option("multiLine", "true").load("/FileStore/tables/yelp_review_tab.csv",sep='\t',header=True)
review.printSchema()
type(review)

```

▸ (1) Spark Jobs

▸ review: pyspark.sql.dataframe.DataFrame = [_c0: string, review_id: string ... 8 more fields]

Cmd 48

```
from pyspark.ml.feature import *
#tokenizer and stop word remover
tokenizer = Tokenizer().setInputCol('text').setOutputCol('words')
#stop word remover
from pyspark.ml.feature import StopWordsRemover
sw_filter = StopWordsRemover()\
  .setCaseSensitive(False)\
  .setInputCol("words")\
  .setOutputCol("filtered")
# we will remove words that appear in 5 docs or less
from pyspark.ml.feature import CountVectorizer
cv = CountVectorizer(minTF=1., minDF=5., vocabSize=2**17)\
  .setInputCol("filtered")\
  .setOutputCol("tf")
# Build the pipeline
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[tokenizer, sw_filter, cv])

```

Cmd 49

```
#We create the idf to create the tf-idf
from pyspark.ml.feature import IDF
idf = IDF().\
    setInputCol('tf').\
    setOutputCol('tfidf')
```

Command took 0.04 seconds -- by s_ovey@stud.hwr-berlin.de at 15/07/2018, 16:22

Cmd 50

```
#creating a pipeline that takes the previous pipeline as an in
from pyspark.ml import Pipeline
idf_pipeline = Pipeline(stages=[pipeline, idf]).fit(resultDF)
```

We use a tab separated data set because with the comma separated data set we had a loading problem because when there was a comma in the data set, pyspark perceived that after the comma there was another column. Because of that, our data set got destroyed and we had many other column which are originally not existing in our dataset. Finally we divided this tab separated dataset into training,validation, and test datasets.

Cmd 52

```
#splitting the data as training validation and test
training_df, validation_df, testing_df = resultDF.randomSplit([0.8, 0.1, 0.1], seed=0)

```

First, we tried to do a logistic regression without elastic net regularization, the accuracies were quite good too, but when we digged deeper in positive and negative words, we found out that the model was overfitting. Then, we applied elastic net regularization to avoid the overfitting with certain alpha and lambda values.

Cmd 54

```
#creating a logistic regression model
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression().\
    setLabelCol('stars').\
    setFeaturesCol('tfidf').\
    setRegParam(0.0).\
    setMaxIter(100).\
    setElasticNetParam(0.)
```

Cmd 58

```
#the most negative words
coeffs_df.sort_values('weight').head(5)
#these words doesn't make sense our model may be overfitting
```

Out[113]:

	weight	word
117072	-5.107207	wayanyway
108202	-4.692166	dissertations
115978	-4.558976	allhere
107927	-4.533517	timestamped
80599	-4.400178	hertzs

Cmd 60

```
#elastic net regularization to avoid overfitting
lambda_par = 0.02
alpha_par = 0.3
en_lr = LogisticRegression().\
        setLabelCol('stars').\
        setFeaturesCol('tfidf').\
        setRegParam(lambda_par).\
        setMaxIter(100).\
        setElasticNetParam(alpha_par)
```

Cmd 64

```
#most negative words now the words make sense
en_coeffs_df.sort_values('weight').head(15)

```

```
Out[119]:
        weight            word
395  -0.368148           worst
434  -0.288376        horrible
396  -0.286900            rude
472  -0.267635        terrible
842  -0.266930        mediocre
198  -0.244333              ok
792  -0.227296  disappointing
```

This was not a good approach as well because we manually entered the alpha and lambda values at the first try. In order to overcome this issue, we created a grid of alpha and lambda values which had 9 combinations in total. We have run all of those models, calculated their accuracies and picked the model with the highest accuracy as our main model. Then we implemented this model both for reviews and tips dataset and you can find our visualizations in the following chapter.

Cmd 68

```
#We create a grid with different combinations of alpha and lambda values
grid = ParamGridBuilder().\
    addGrid(en_lr.regParam, [0., 0.01, 0.02]).\
    addGrid(en_lr.elasticNetParam, [0., 0.2, 0.4]).\
    build()
```

Cmd 70

```
#We will fit 9 models with the 9 parameter combinations in our grid.
all_models = []
for j in range(len(grid)):
    print("Fitting model {}".format(j+1))
    model = en_lr_estimator.fit(training_df, grid[j])
    all_models.append(model)
```

▸ (51) Spark Jobs

```
Fitting model 1
Fitting model 2
Fitting model 3
Fitting model 4
Fitting model 5
Fitting model 6
Fitting model 7
Fitting model 8
Fitting model 9
```

Cmd 71

```
# estimate the accuracy of each of models:
accuracies = [m.\
    transform(validation_df).\
    select(fn.avg(fn.expr('float(stars = prediction)')).alias('accuracy')).\
    first().\
    accuracy for m in all_models]
```

Cmd 74

```
#our best model has a accuracy of 88.9%
best_model = all_models[best_model_idx]
accuracies[best_model_idx]
```

```
Out[129]: 0.8894220196325396
```

Cmd 75

```
#We run the model that we created for all reviews using .transform
sentiment=best_model.transform(resultDF)
```

```
sentiment: pyspark.sql.dataframe.DataFrame = [review_id: string,
```

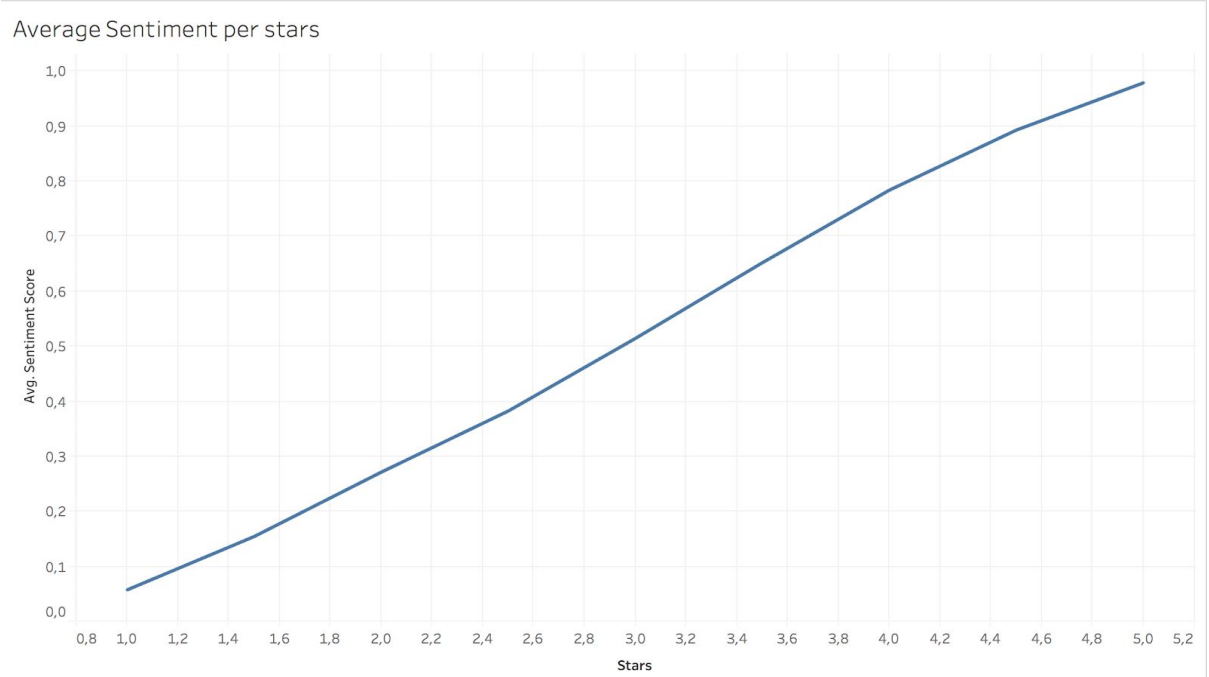
Cmd 78

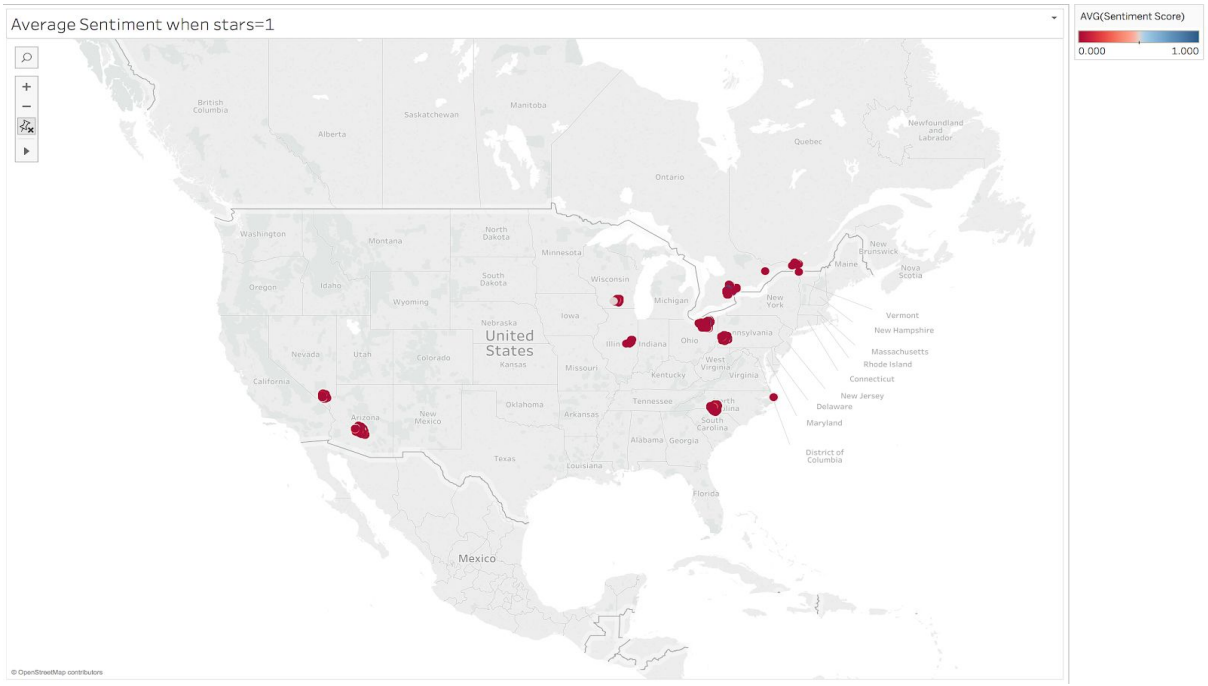
```
#Now we run the model, which we developed with reviews dataset, on tips data set.
sentiment_tip=best_model.transform(train1)
```

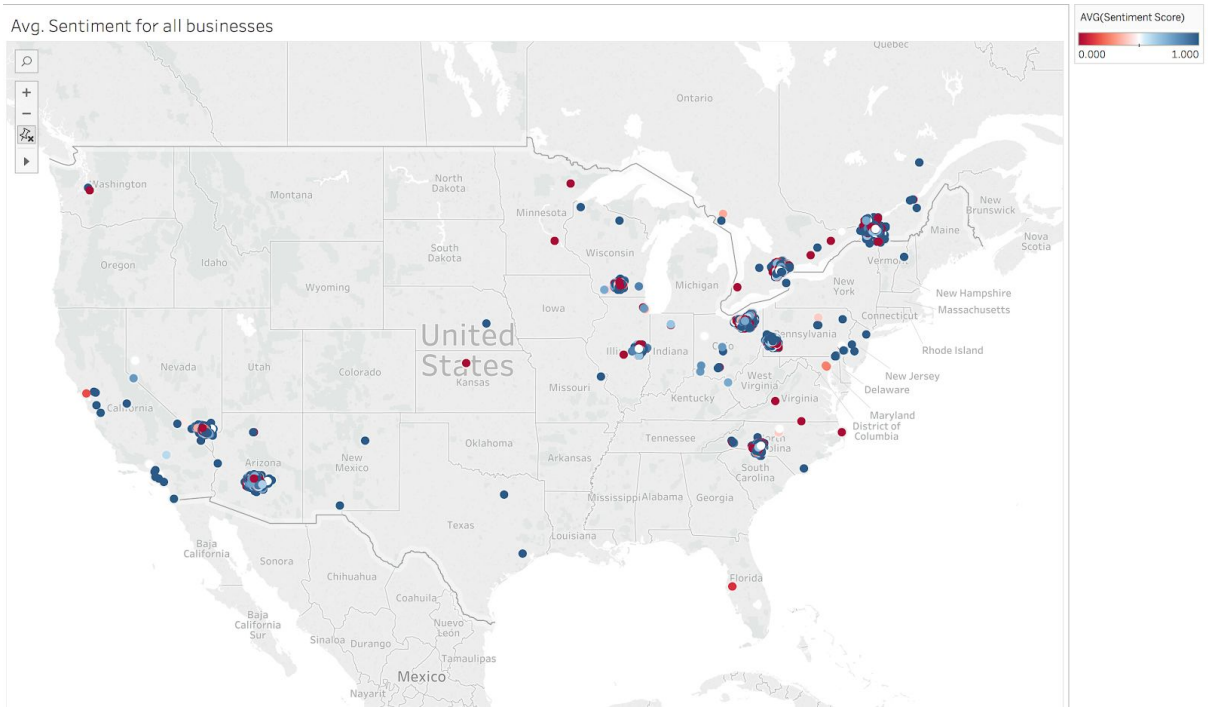
Finally, we created a use case to show how the sentiment scores can be used in an industry level. On the sixth figure below, you see how the sentiment is changing for all italian restaurants in the US. In the next figure, you see an example from Charlotte, NC. For example, if owner of one stores wants to become more competitive in their market, he can check the sentiment score of his/her own restaurant which shows how his customers are thinking about his/her store and he/she can see the sentiment of his competitors and he can figure out which competitor to benchmark to make people think better about his/her restaurant.

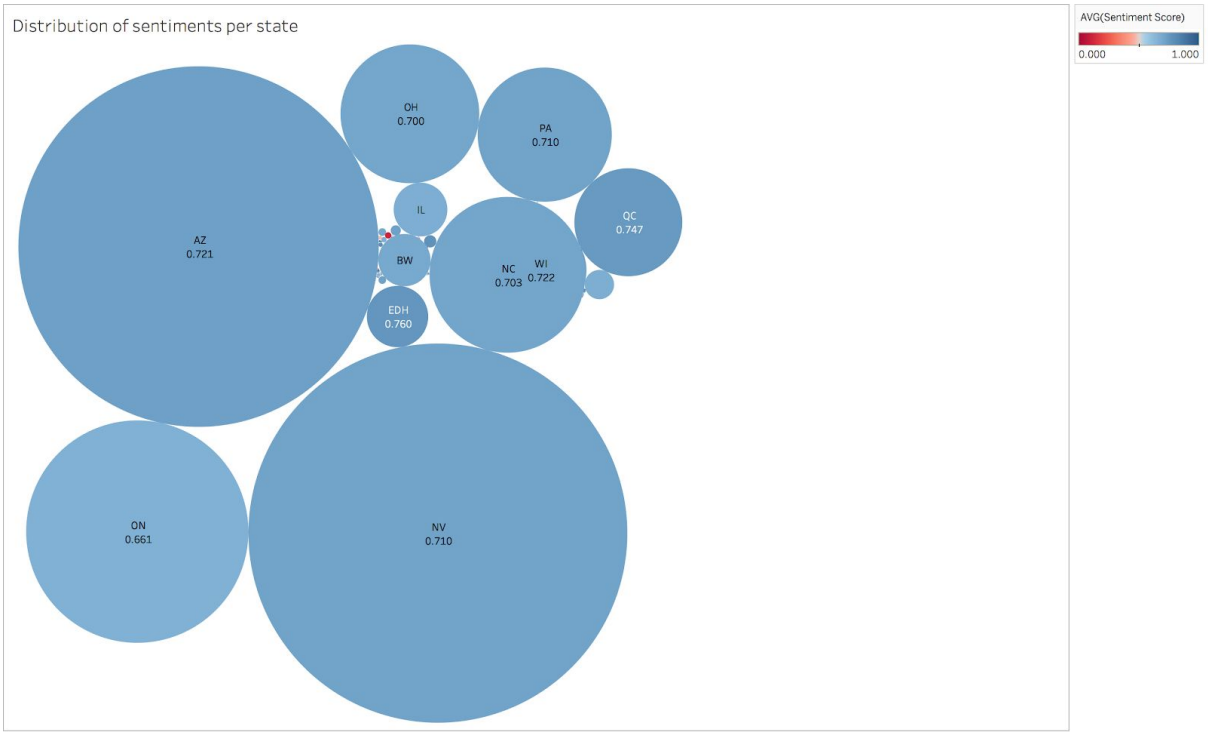
You can find the html file and Tableau files which are named “#5_Yelp_SentimentAnalysis.html” and “#7_Tableau_Sentiment_Analysis” in the folder which we provided you.

Tableau visualizations

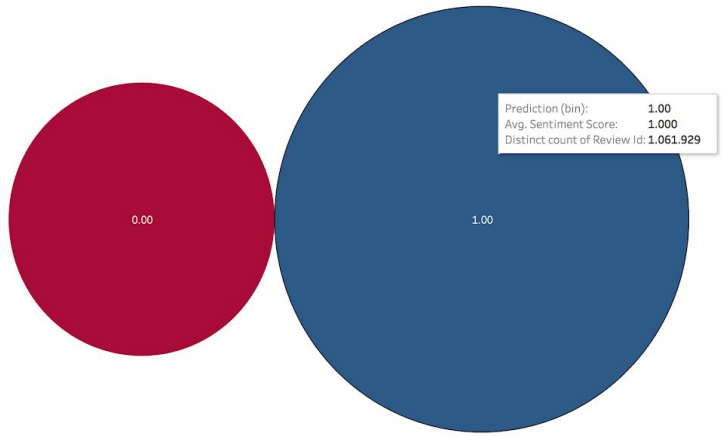




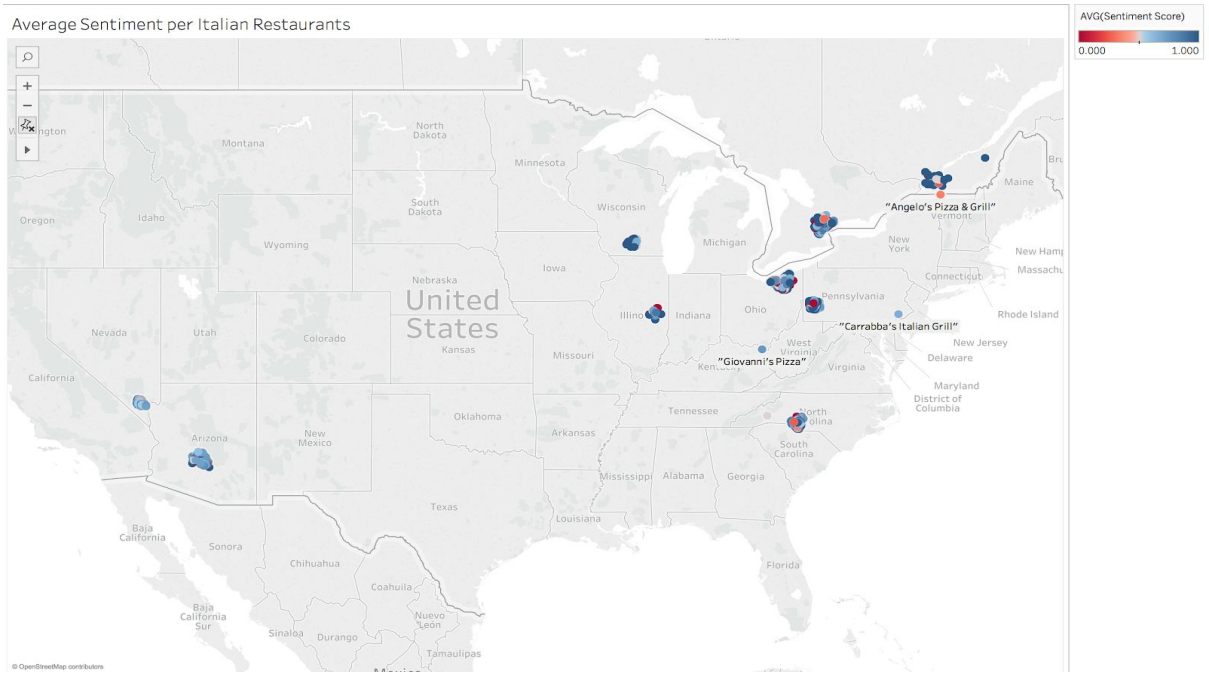

Avg. Sentiment for all businesses
AVG(Sentiment Score)
0.000
1.000
Ontario
Quebec
Washington
Montana
North Dakota
Minnesota
Oregon
Idaho
South Dakota
Wisconsin
Wyoming
Michigan
Iowa
New York
United States
Nevada
Utah
Colorado
Kansas
Missouri
Indiana
Pennsylvania
Connecticut
Rhode Island
New Jersey
Delaware
Maryland
District of Columbia
West Virginia
Kentucky
Virginia
Tennessee
Oklahoma
Arkansas
New Mexico
Arizona
Mississippi
Alabama
Georgia
South Carolina
Texas
Louisiana
Florida
Baja California
Sonora
Chihuahua
Coahuila
Nuevo Leon
Baja California Sur
Sinaloa
Durango
Tamaulipas
Mexico
Nayarit
New Brunswick
Maine
Nova Scotia
Vermont
New Hampshire
Massachusetts

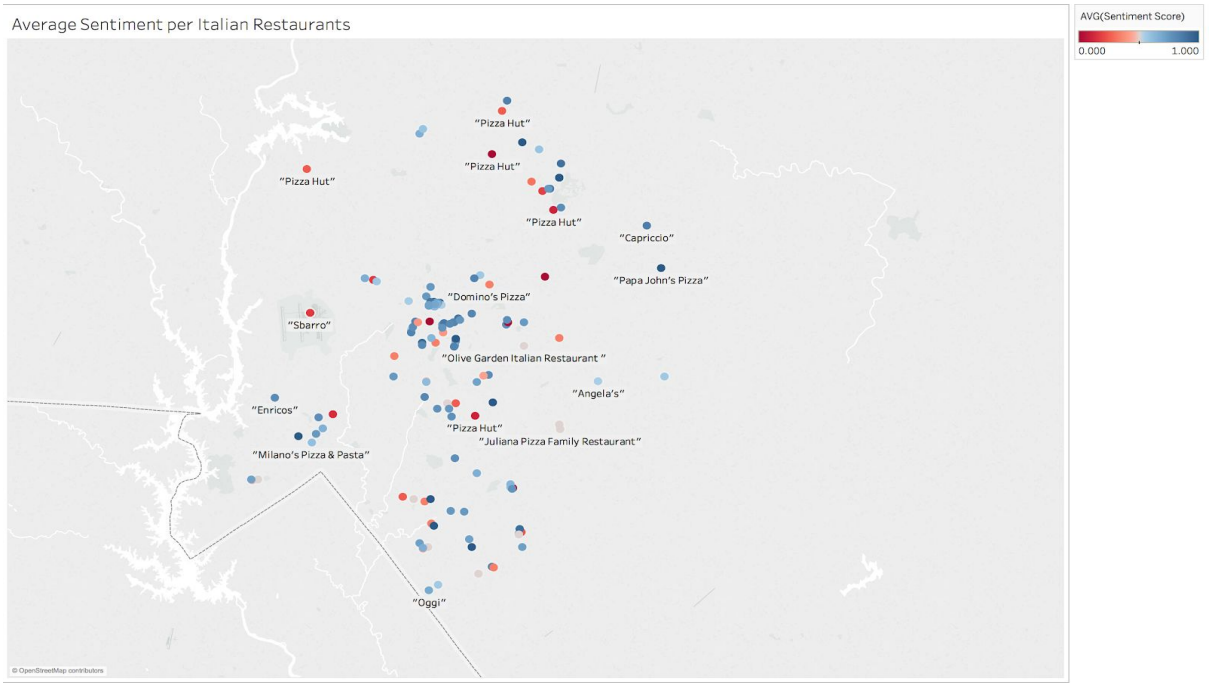

Distribution of sentiments per state
AVG(Sentiment Score)
0.000
1.000
OH
0.700
PA
0.710
IL
QC
0.747
AZ
0.721
BW
NC
0.703
WI
0.722
EDH
0.760
ON
0.661
NV
0.710

Overall Distribution of Sentiments



Average Sentiment per Italian Restaurants




Average Sentiment per Italian Restaurants
AVG(Sentiment Score)
0.000
1.000
"Pizza Hut"
"Pizza Hut"
"Pizza Hut"
"Pizza Hut"
"Capriccio"
"Papa John's Pizza"
"Domino's Pizza"
"Sbarro"
"Olive Garden Italian Restaurant "
"Angela's"
"Enricos"
"Pizza Hut"
"Juliana Pizza Family Restaurant"
"Milano's Pizza & Pasta"
"Oggi"
© OpenStreetMap contributors

Graph & network analysis

Graph is heavily used in mining important relationships in society, among businesses, and many other scenarios[9]. In this project, the team applied graph database technology, Neo4j, to identify potential fraud users and mining important users, as known as key opinion leaders.

Import data to Neo4j

Below are the steps to import data to Neo4j from Spark:

1. Use Spark to slice the data and convert data into CSVs
2. Preprocess the data by Python
3. Load data into Neo4j (codes can be found in https://drive.google.com/open?id=1aD9ccgDFMUyJcroIbWTOt522Bjc9EJ08)

Fraud detection

According to research conducted by Harvard Business School, around 20% of Yelp reviews have been potentially manipulated. Fraud users is threatening the trustworthiness and accuracy of analysis based on Yelp dataset. Comments from fraud users are noises. Businesses cannot make good decisions based on noisy data. Therefore, detecting fraud users and remove their comments are necessary.

The definition of a fraud user is strict forward: a user with low review count and gave extreme ratings on two shops in same category and same city. Users with low review count suggest that the accounts are used for gunner purpose. Giving extreme ratings to shops in same city and same category is not a common behaviour of a real user. Therefore, users with these two characteristics are potentially fraud users. Although this definition cannot cover all the fraud users, it can help us to screen out accounts that are obviously a gunner account.

[9]Graph Database Use Cases https://neo4j.com/use-cases/

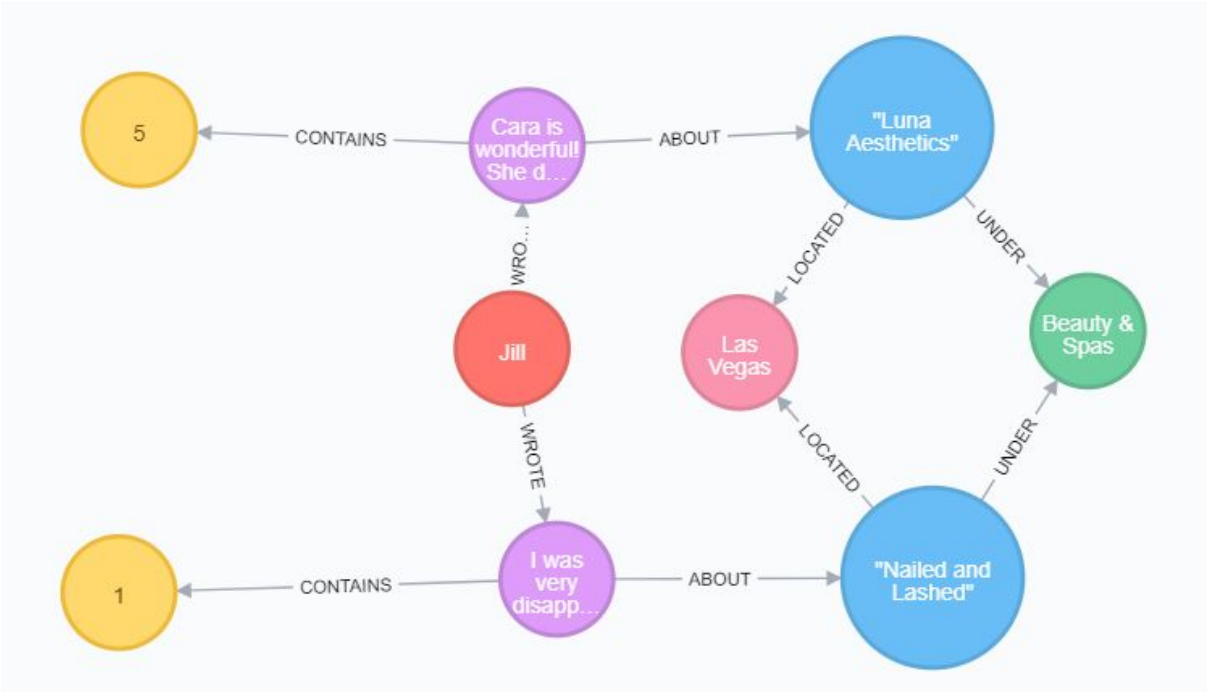


Fig. 1 Network structure of a potential fraud user, her comments, and the related businesses

Let's look at the only two comments made by user Jill.
Positive comments given to Luna Aesthetics: *Cara is wonderful! She does an excellent job and is a wonderful person. Highly recommend!!!*

Negative comments given to Nailed and Lashes: *I was very disappointed in this salon! Had a lash fill on Friday, by Monday most of my lashes were gone. I called to let them know and was basically told too bad! They said they would schedule me for another appointment but I would have to pay. I cant imagine that they would not stand behind their service!! Dont waste your time going here!!*

We suspect that Jill is a fraud account. First, from her comments we can see that the user gave extreme comments to two shops. Second, both comments gave suggestion of whether other customers should go to the shop or not. Furthermore, both comments are very specific to particular aspect of business, positive to certain employee, negative to lash quality.

After applying query of the definition of potential fraud user, we found out 1,082 reviews written by 196 users are potentially fraud and can be filtered.

Identifying Influencers by using Graph

Our team applied graph algorithms to identify influencers in the social network of Yelp. By comparing revenue data from WSD and ratings on Yelp, WSD found out that one star increase in Yelp rating leads to five to nine percent increase in revenue. Stars is the easiest attributes users can inspect of when they are deciding which shops to go. Our team suggest that getting more positive ratings from more users can lead to higher revenue. The problem is how can business allocate resources efficiently? This question is critical to independent small businesses because their resources are limited.

Our team applied two graph algorithms on Neo4j, Strongly Connected Component and PageRank. Strongly Connected Component helps us to cluster users into groups. It serves as a initial partition of users. When business need to research on their customers in order to make decision, businesses can refer to the whole partition, so the sample size is larger. PageRank is to assign rank according to the size of network a user connected to. Businesses can make use of this information to decide which users are more important and should respond to them first.

Strongly Connected Component

According to Wikipedia: “The strongly connected components or disconnected components of an arbitrary directed graph form a partition into subgraphs that are themselves strongly connected.” In this project, the team apply strongly connected component algorithm on Neo4j after importing all the data to the database. The Neo4j strongly connected component algorithm will annotate the node of its partition in one of the attribute. Our team applied such algorithm on users and friends relationship. In the figure below., one can see that users can be partitioned by their friends relationship with the others.

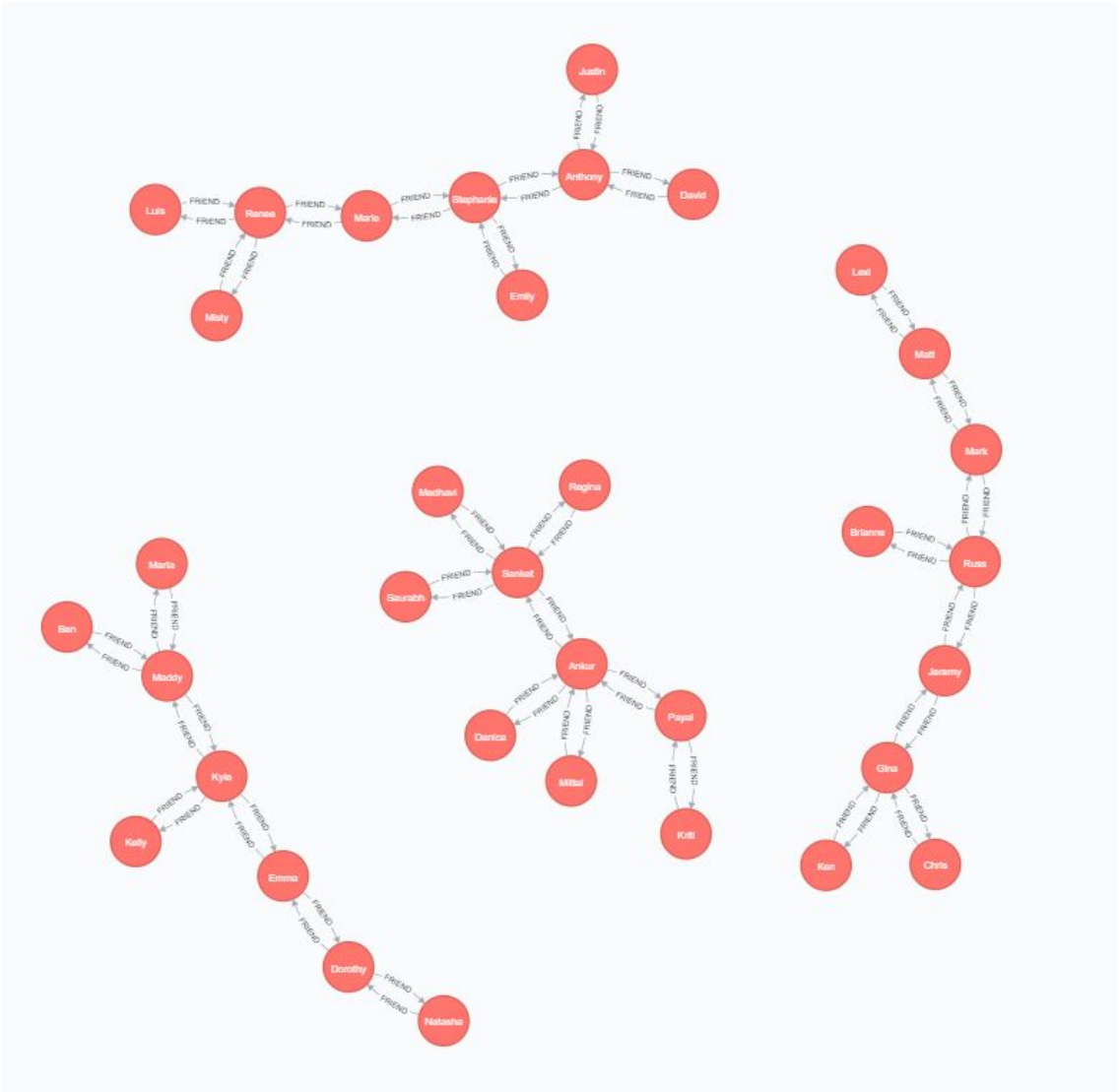


Figure: Users and how they are connected by friends relationship

Our team suggests two potential usage of Strongly Connected Components to small independent businesses. First, expand target respondents of market research. Small independent businesses may not have enough samples to study on. They can consider users' friends as target audience as well. Second, small independent businesses can group users who commented on the business and understand their users' behaviour.

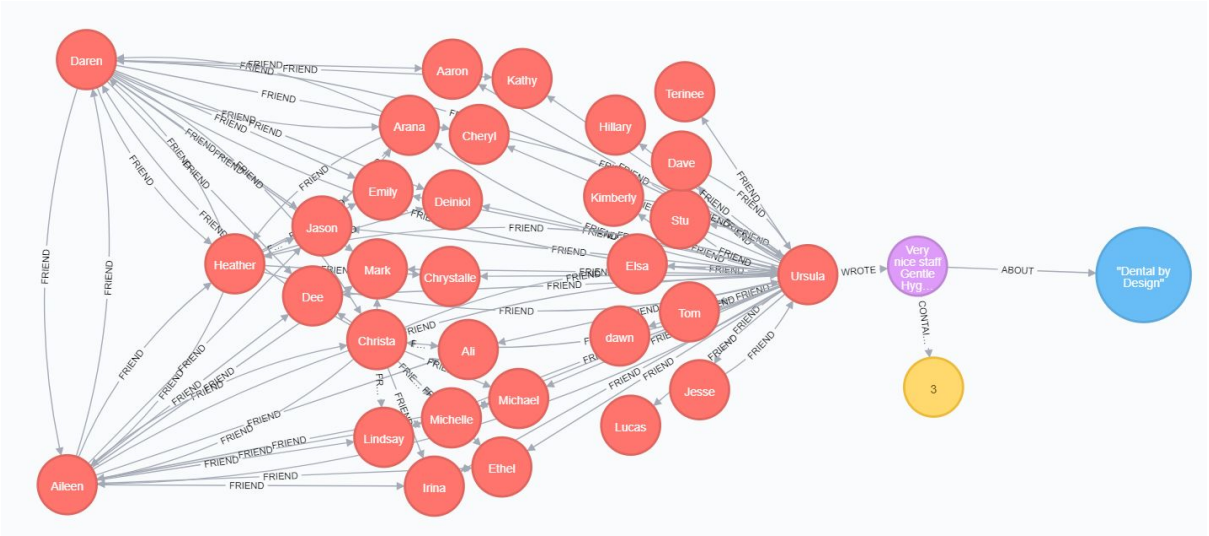


Figure: Ursula can help the business to directly reach 10 users as a group, and after these 10 users can reach more users in other groups.

PageRank

PageRank is originally a ranking system designed by Google to identify the best pages on the web. The algorithm assign score to each node according to how many connection they have, and how many connections do the connected nodes have.

$$PR(A) = 1 - d + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right).$$

Equation 1. PageRank Formula. PageRank of node A is determined by d the damping factor, PageRank of nodes that are connected to A, and number of connections those connected nodes have (e.g.L(B)).

This same concept can be applied to users as well. Small independent businesses can prioritise the target of customer service. Resources are always limited, but businesses can now respond to comments strategically by consider numbers of users with high PageRank. For example, if a business received two 1-star reviews from two users but resources are limited to respond to one user, it can response to the user with higher PageRank. The business can ease the negative comments by understanding the bad experience, giving coupons, or businesses can improve the business according to the content of the comment.

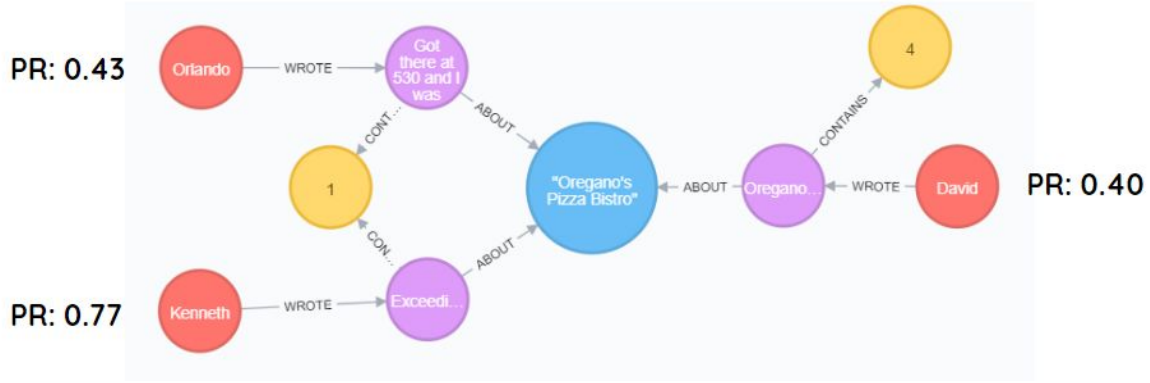
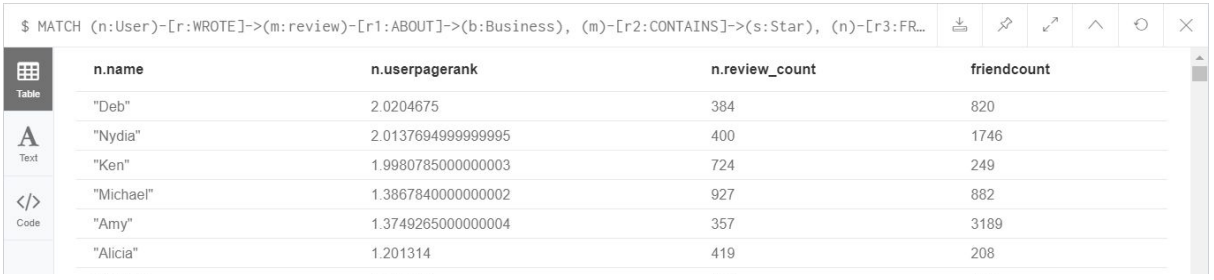


Figure: Users Orlando and Kenneth both gave 1 star comment.

PageRank can give a different point of view to importance of users. As an example in Fig 5, users with higher review count or friends count do not necessary to have higher PageRank.



```
$ MATCH (n:User)-[r:WROTE]->(m:review)-[r1:ABOUT]->(b:Business), (m)-[r2:CONTAINS]->(s:Star), (n)-[r3:FR…
```

Table | Text | Code

n.name	n.userpagerank	n.review_count	friendcount
"Deb"	2.0204675	384	820
"Nydia"	2.0137694999999995	400	1746
"Ken"	1.9980785000000003	724	249
"Michael"	1.3867840000000002	927	882
"Amy"	1.3749265000000004	357	3189
"Alicia"	1.201314	419	208

Figure: Comparing PageRank, review count and friends count.

Conclusion

In this report, we presented the analysis of mainly textual data by using machine learning to generate insights for independent restaurants. The project was based on a Big Dataset, which consists of over 9 GB of data from Yelp.com. After motivating the importance of analysis of customer reviews coming not only from Yelp but also from other sources of customer feedback, we highlighted different ways of analyzing this type of data, among others Review Rating Prediction, Sentiment Analysis, Latent Dirichlet Allocation and Graph Network Analysis. We conclude by presenting the difficulties encountered during the project and lessons learned.

Difficulties

Big Data is a field with dozens of fast-moving technologies. In the course of our project, we had to overcome several difficulties, both on technical as well as on organizational level. We present some of them in the table below.

Timeline	Milestone	Difficulties & Solutions
11.04.18	Ideas brainstorming	Yelp dataset selected
18.05.18	Kick-off presentation	Roadmap, scope & technology selection
29.05.18	First prototype	Local machine: Hadoop cluster & Spark on Ubuntu -> very slow -> moved to Spark cluster on Azure
13.06.18	Review presentation	Azure HDInsight Neo4j Databricks for processing small tables < 2Gb
09.07.18	Intermediate technical solution & Final visualization	Tableau connected to HDInsight Spark Cluster
16.07.18	Final presentation	Finalizing all results

The first challenge was finding a working environment that would satisfy the following criteria:

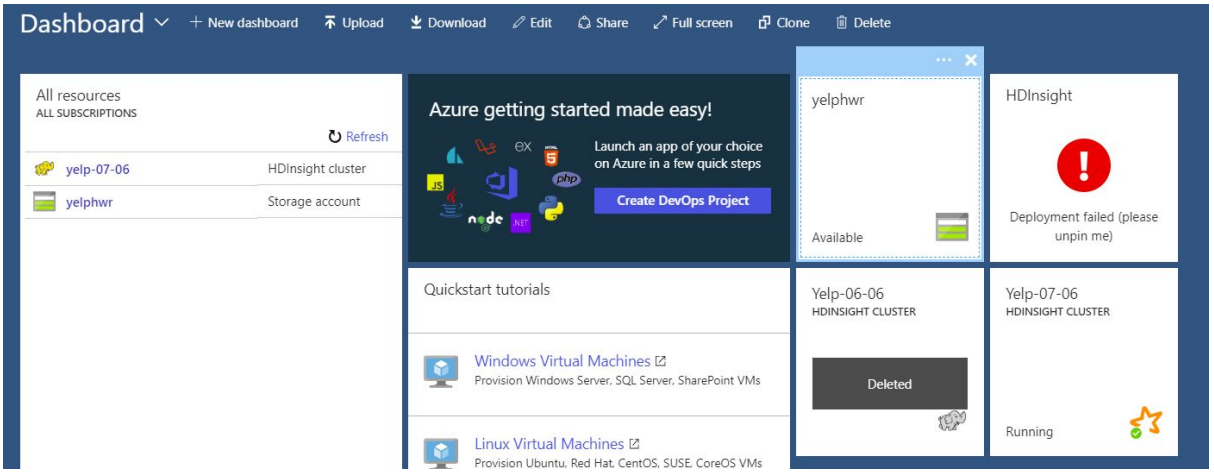
- It should enable collaborative work.
- It should handle Big Data by allowing distributed storage and parallel processing, preferably in-memory.
- Ideally, it should be for free. If not, the costs should be affordable for students.

After spending countless hours trying to establish a trouble-free working environment through a local installation of Hortonworks Sandbox or Spark on a Linux virtual machine, we came across Microsoft Azure HDInsight platform, which gave the opportunity of creating a distributed Spark cluster that we can use on-demand and for free, as long as we use a 30-day free trial version. This platform satisfied all our criteria because:

- we were able to work in the cloud and share our work in a team,

- we could create a distributed Spark cluster which not only spreads the work across multiple worker nodes but also works in memory,
- it did not cost us anything as we switched between free trial accounts.

A single drawback of this approach was that we could not work concurrently on the same account, as one of us got disconnected if another person tried to connect to the cluster simultaneously. The screenshot below represents the error that we encountered because of this issue: if one person was logged in, the other team member could not successfully deploy a second Spark cluster. However, we solved this issue simply by scheduling.



Lessons learned

This project was beneficial for us as we discovered many Big Data technologies and could see how they interact with each other. Not only could we put the knowledge gained in the course into practice, but also we learned a lot about distributed storage, parallel processing and performing machine learning on large datasets. Furthermore, we learned hot to choose the right tool for a given scenario: for us, Microsoft Azure and distributed Spark cluster worked best, but we also learned how to use Databricks, Neo4j and many more. Additionally, we learned our strengths and weaknesses while working in a very diverse team. We are grateful for giving us this opportunity!

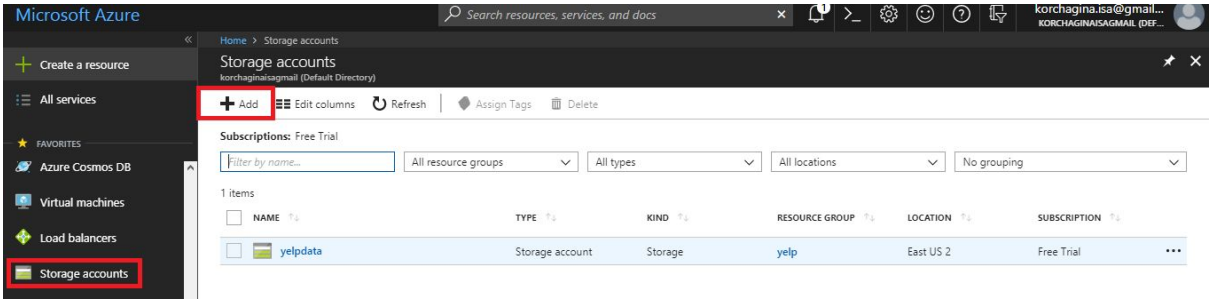
Appendix 1

How to set up a cluster on Microsoft Azure

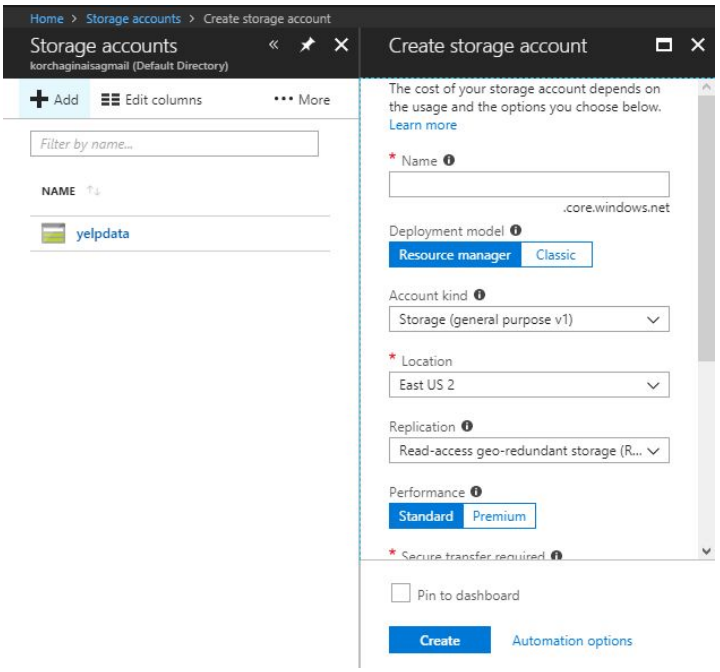
Step 1. Create free Azure account for 30 days with 170 euro credit. You will need to provide your credit card (you won't be charged until you choose to upgrade).

Step 2. Create storage account and upload Yelp dataset.

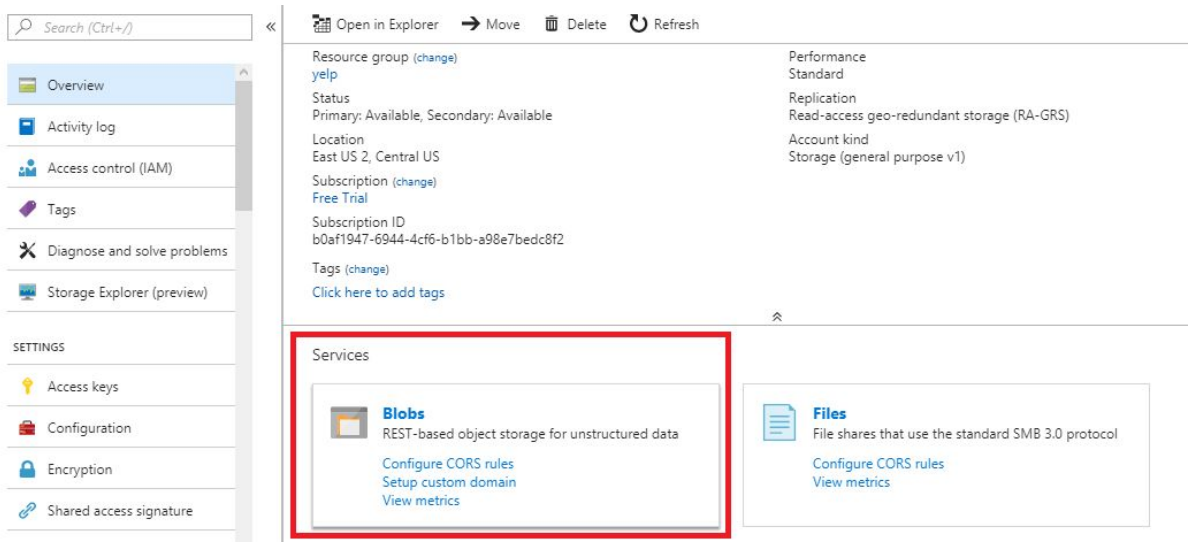
In the Azure portal, expand the menu on the left side to open the menu of services, and choose **More Services**. Then, scroll down to **Storage**, and choose **Storage accounts**. On the **Storage Accounts** window that appears, choose **Add**.



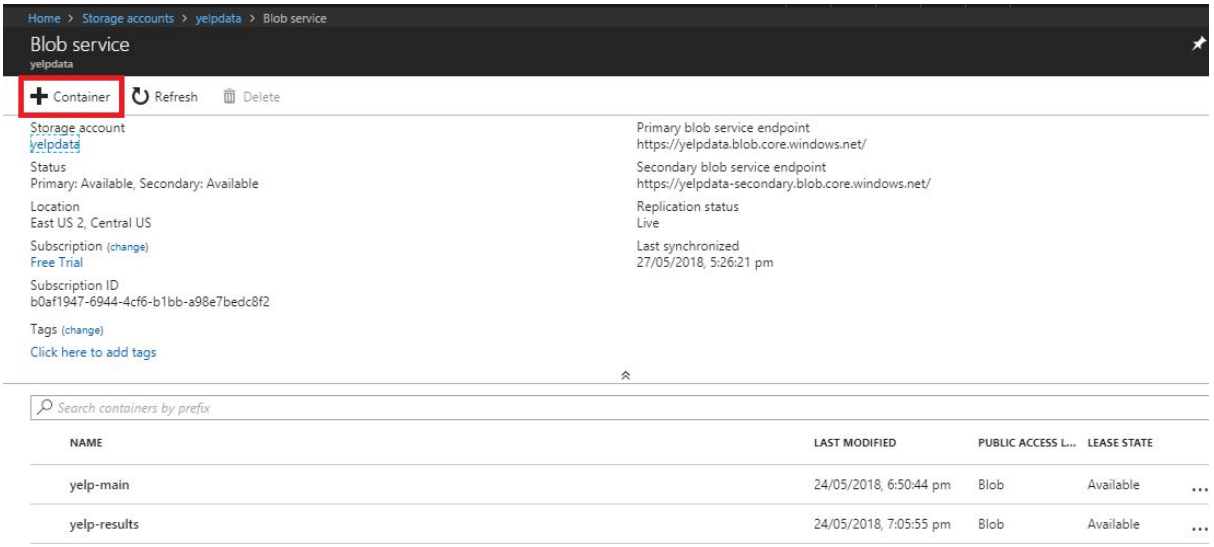
Leave all default settings, enter a name for your storage account and new resource group. Select **Pin to dashboard**. Click **Create** to create the storage account.



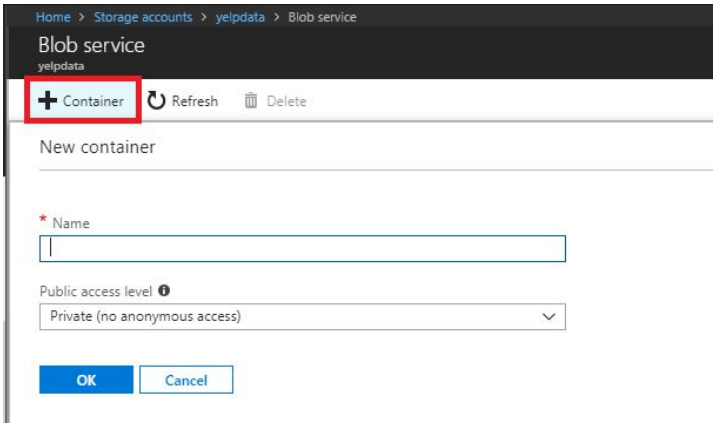
Afterwards, click on **Blobs** section.



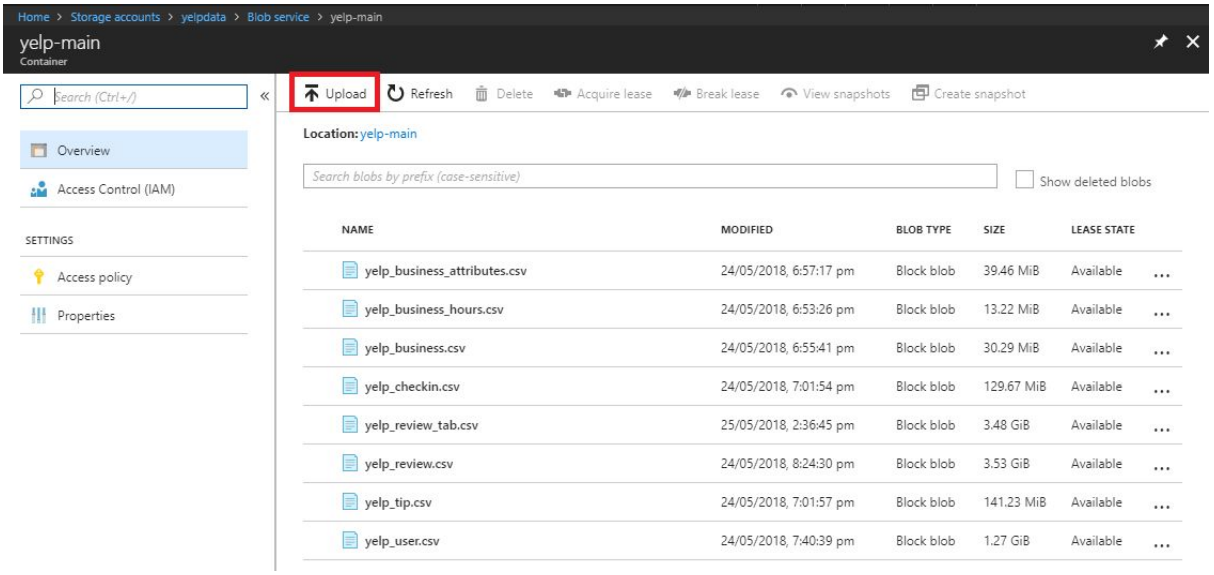
Now you can create your first **Container**:



Enter a name for new Container (for example yelp-main):

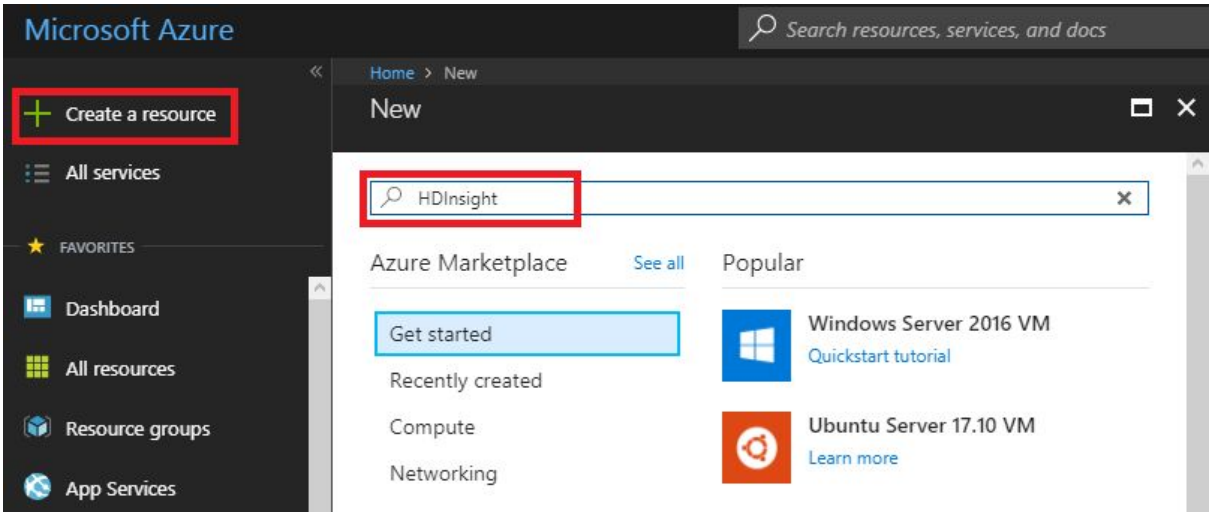


Upload csv files:

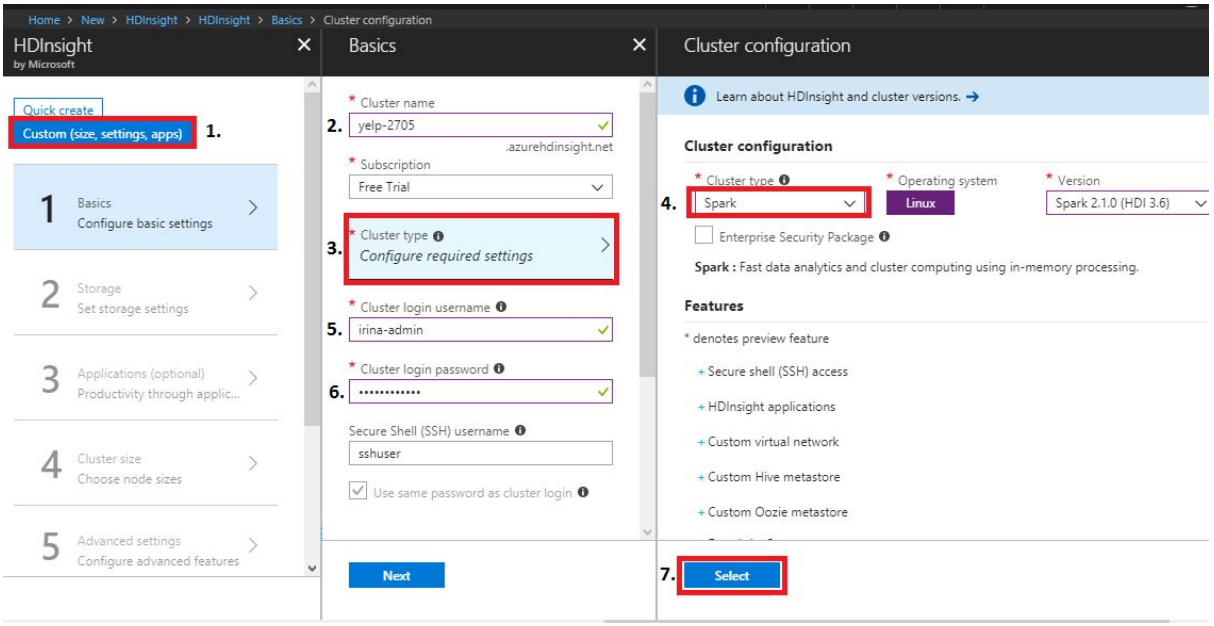


Step 3. Create an HDInsight **Spark cluster**

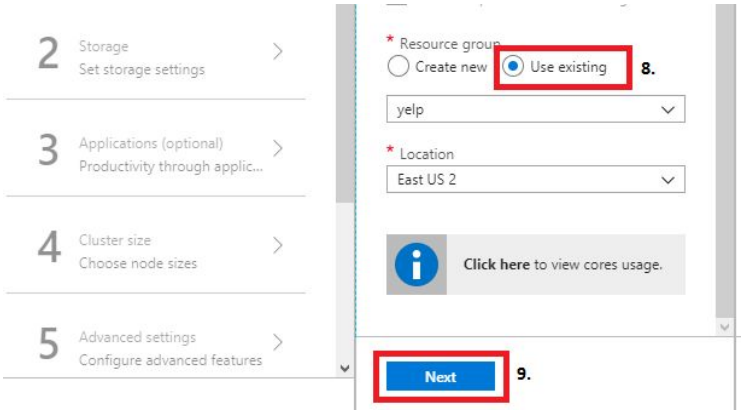
Go to the main menu, click **Create a resource** and type **HDInsight.** Click **Create.**



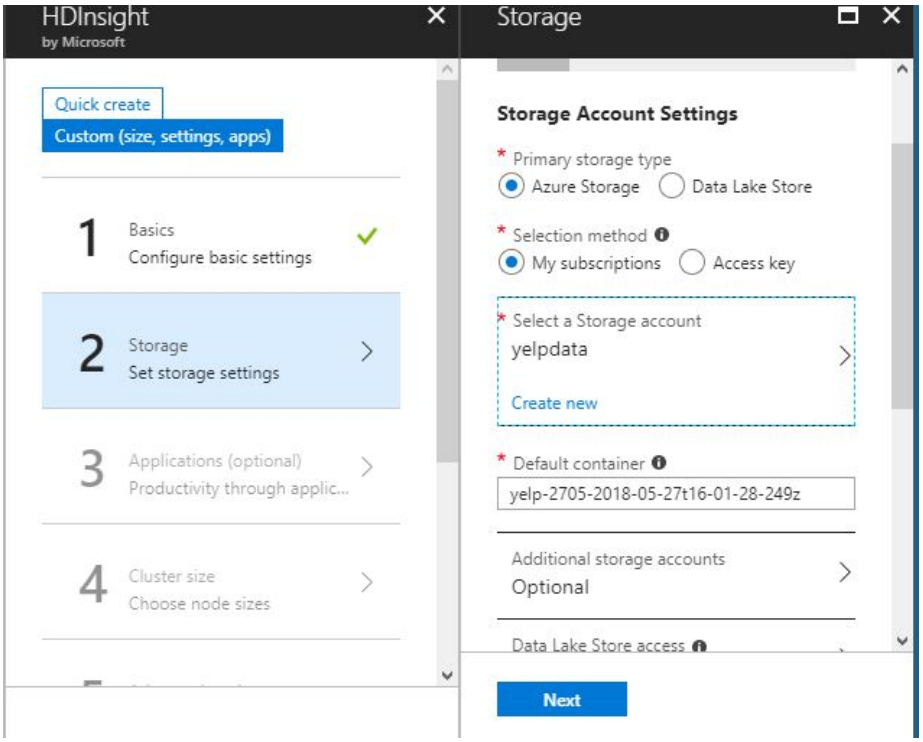
1. Choose **Custom** options.
2. Give a **name** to new Spark cluster.
3. Press **Configure required settings**.
4. Select cluster type **Spark**.
5. Enter the cluster login **username**.
6. Enter the cluster login **password**.
7. Press **Select**.



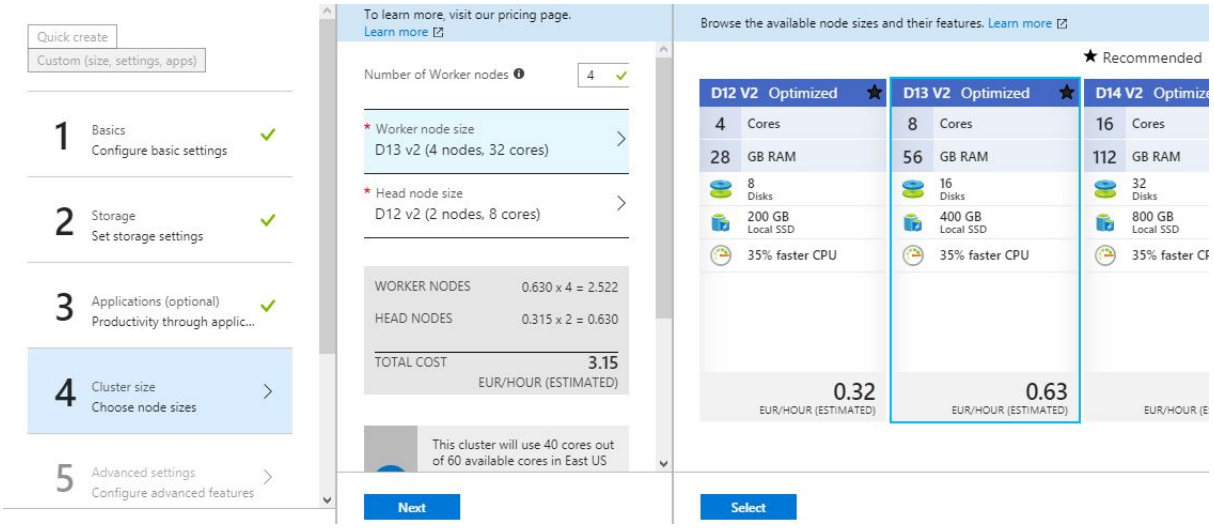
8. Click **use existing resource group** and choose the one where you uploaded Yelp data in Step 2.
9. Click **Next**.

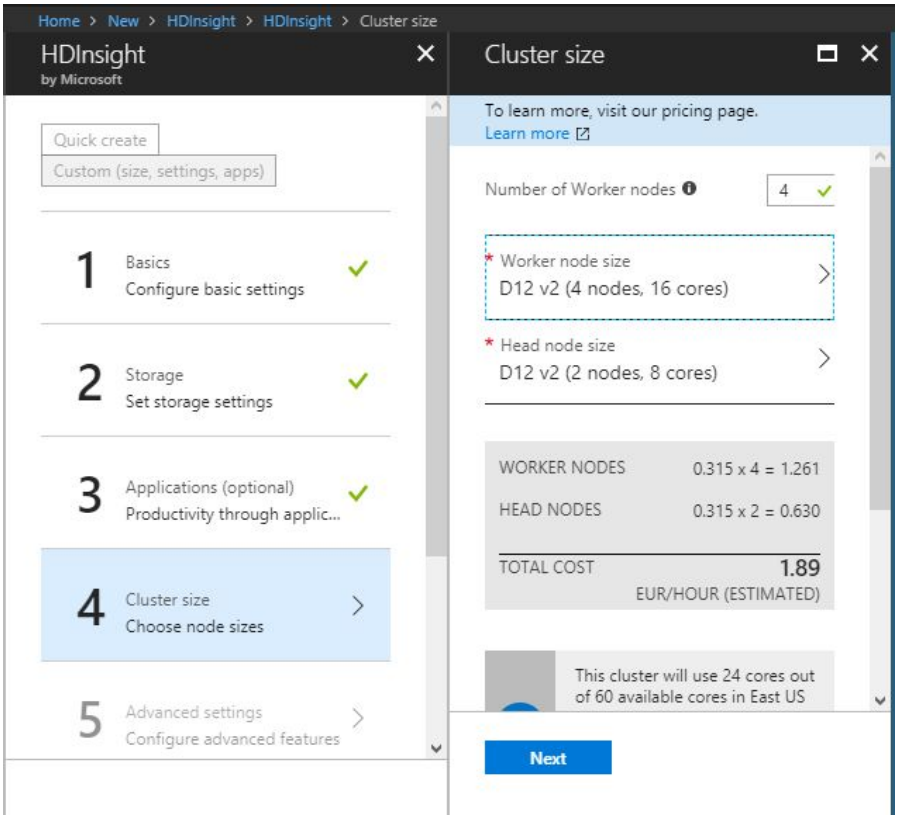


In the next dialog window leave all default options. When creating the cluster second time, we need to specify Default container from previous cluster.

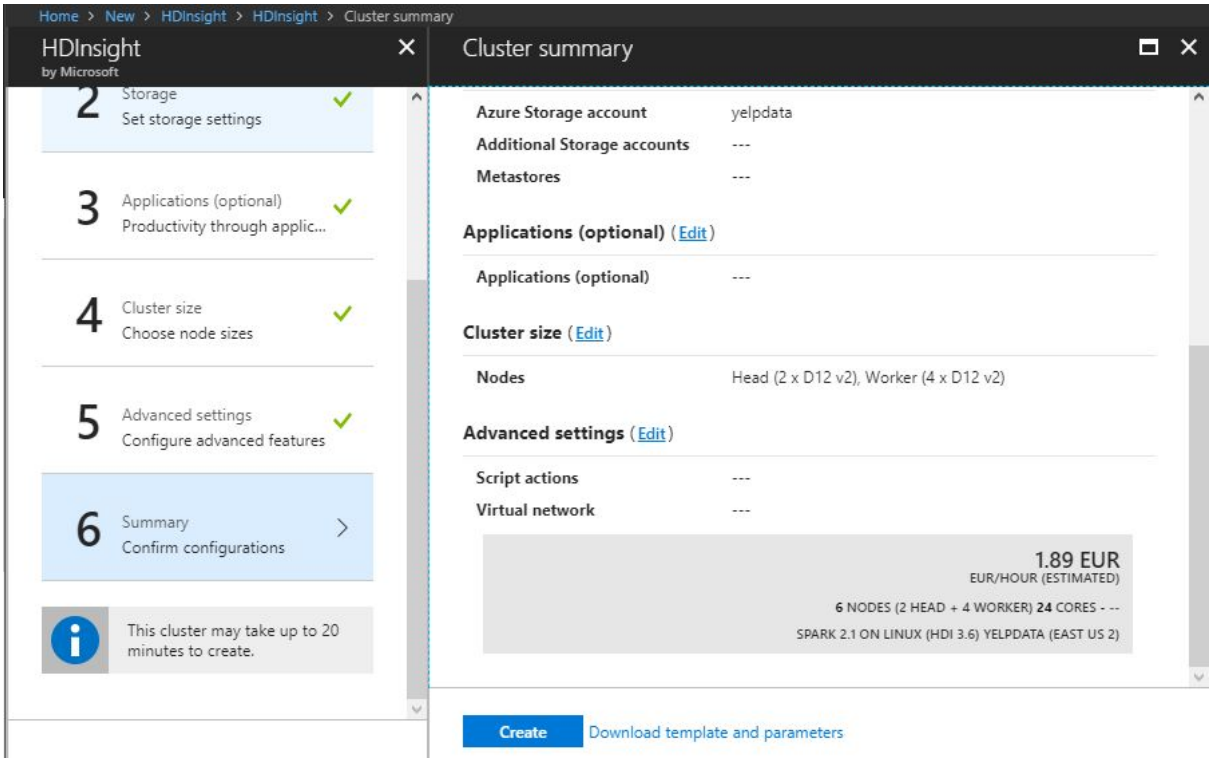


Here we can choose our cluster configuration:





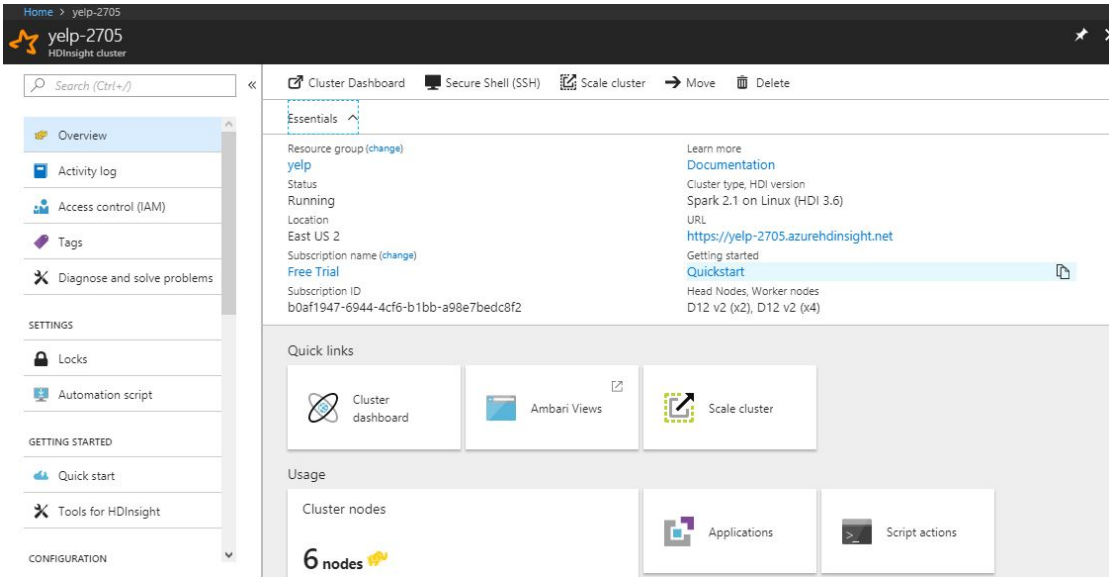
Click **'Create'** below to deploy your cluster. Once created you will be billed until your cluster is deleted.



After submitting a cluster deployment, we have to wait around 10-20 minutes.

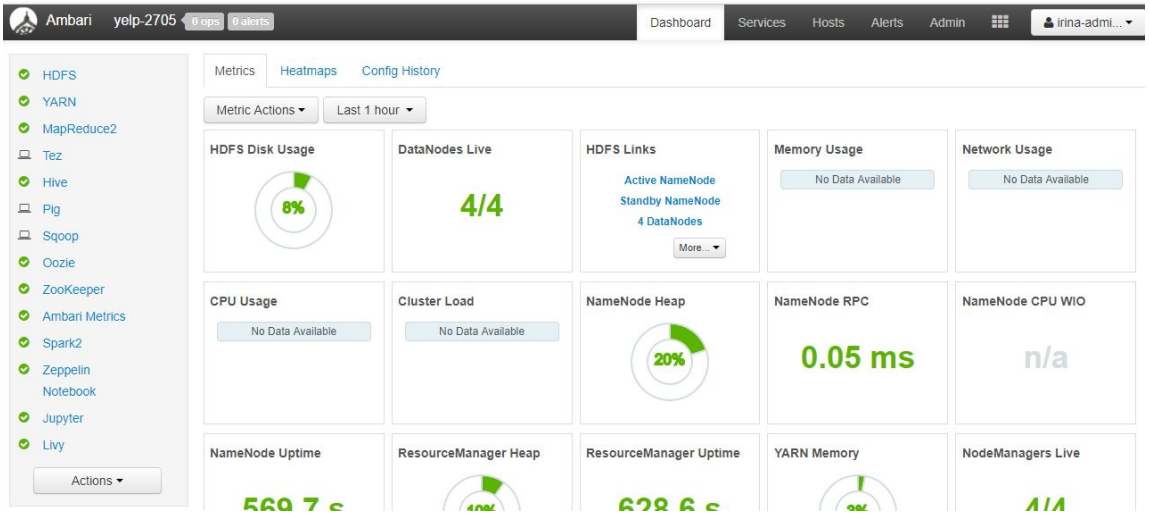
Step 4. Spark Cluster

After some waiting, we finally can start to work.



Click on **Cluster dashboard.** Here we have several options:

1. Ambari view



2. Jupyter notebook
3. Zeppelin notebook
4. Spark history server
5. Yarn

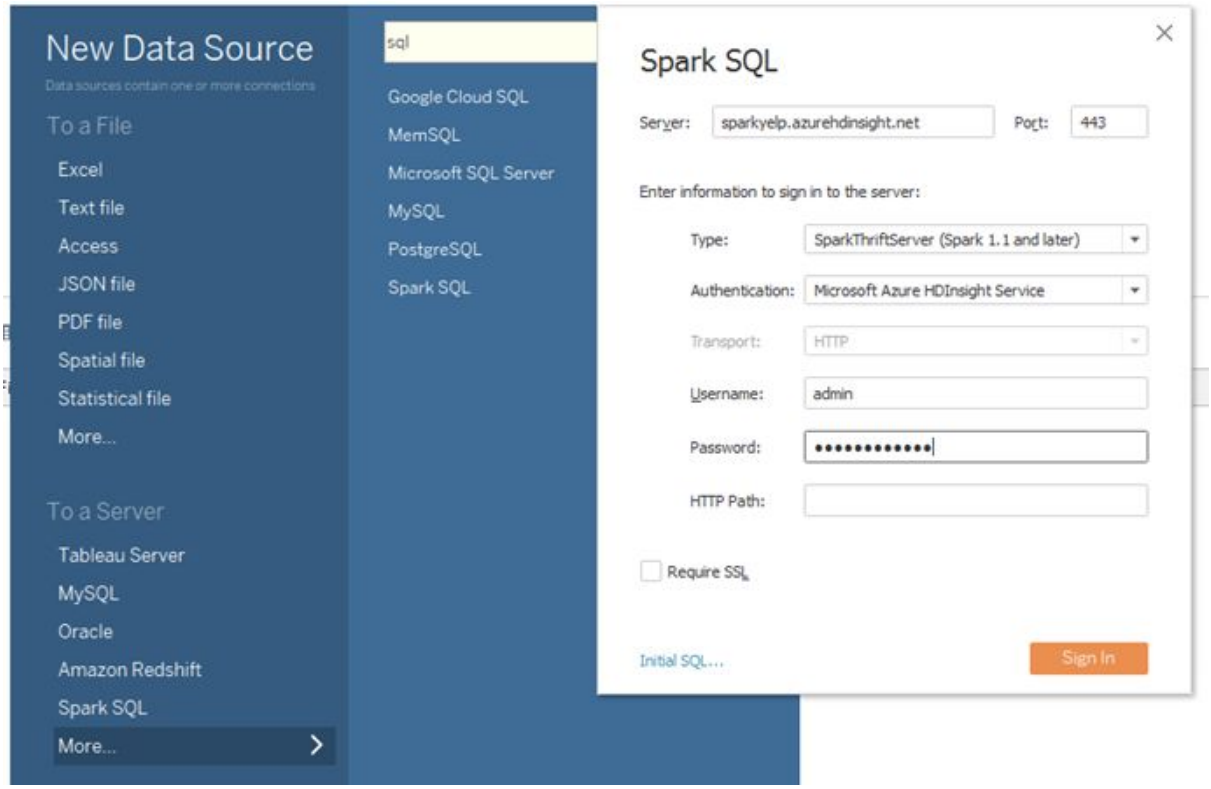
Step 5. Jupyter notebook

Appendix 2

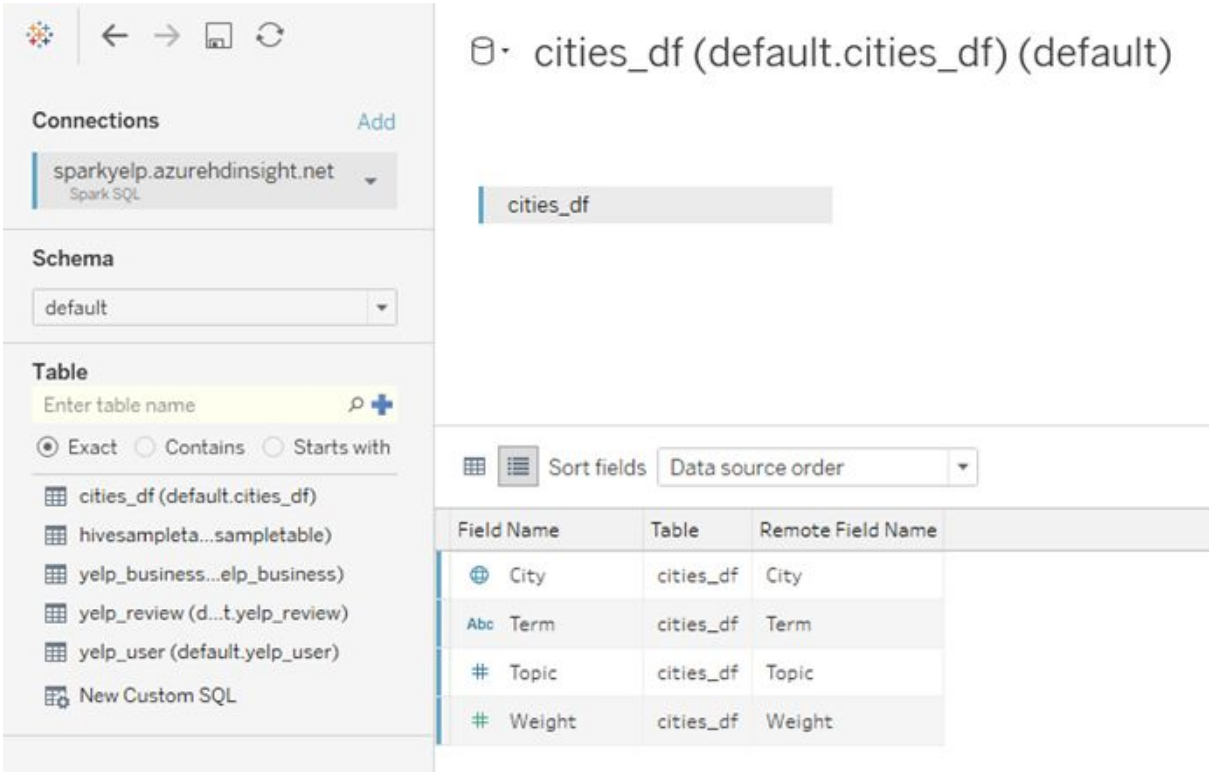
How to connect Tableau to HDInsight Spark cluster

Step1. Install Microsoft® Spark ODBC Driver from https://www.microsoft.com/en-us/download/details.aspx?id=49883

Step 2. Create a new connection to “Spark SQL”. Fill in the server’s name, port (443 is dedicated for connections to HDInsight Spark cluster) and credentials.



Step 3. Choose the schema (in this case it’s called “default”).


Connections
Add
sparkyelp.azurehdinsight.net
Spark SQL
Schema
default
Table
Enter table name
Exact
Contains
Starts with
cities_df (default.cities_df)
hivesampleta...sampletable)
yelp_business...elp_business)
yelp_review (d...t.yelp_review)
yelp_user (default.yelp_user)
New Custom SQL
cities_df (default.cities_df) (default)
cities_df
Sort fields
Data source order
Field Name
Table
Remote Field Name
City
cities_df
City
Term
cities_df
Term
Topic
cities_df
Topic
Weight
cities_df
Weight

Appendix 3.

Yelp Data Preprocessing & Ingestion into Hive

Business: Preprocessing

```
import pandas as pd
df = pd.read_csv('yelp_business_hours.csv')
df.head()
```

	business_id	monday	tuesday	wednesday	thursday	friday	saturday	sunday
0	FYWN1wneV18bWNgQjJ2GNg	7:30-17:0	7:30-17:0	7:30-17:0	7:30-17:0	7:30-17:0	None	None
1	He-G7vWjzVUysIKrfNbPUQ	9:0-20:0	9:0-20:0	9:0-20:0	9:0-20:0	9:0-16:0	8:0-16:0	None
2	KQPW8lFf1y5BT2MxiSZ3QA	None	None	None	None	None	None	None
3	8DShNS-LuFqpEWIp0HxijA	10:0-21:0	10:0-21:0	10:0-21:0	10:0-21:0	10:0-21:0	10:0-21:0	11:0-19:0
4	PfOCPjBrlQAnz__NXj9h_w	11:0-1:0	11:0-1:0	11:0-1:0	11:0-1:0	11:0-1:0	11:0-2:0	11:0-0:0

We wanted to use the table above to analyze businesses with regard to how long they are open. That is why we parsed all those column into opening time and closing time. Based on that, we derived a column that indicates how long the respective business is open on each day.

1 Replace all commas with semicolon

```
df.replace(to_replace=",", value=";", inplace=True, regex=True)
df.head()
```

	business_id	name	neighborhood	address	city	state	postal_code	latitude	longitude	stars	review_count	is_open
0	FYWN1wneV18bWNgQjJ2GNg	"Dental by Design"	NaN	"4855 E Warner Rd; Ste B9"	Ahwatukee	AZ	85044	33.330690	-111.978599	4.0	22	1
1	He-G7vWjzVUysIKrfNbPUQ	"Stephen Szabo Salon"	NaN	"3101 Washington Rd"	McMurray	PA	15317	40.291685	-80.104900	3.0	11	1
2	KQPW8lFf1y5BT2MxiSZ3QA	"Western Motor Vehicle"	NaN	"6025 N 27th Ave; Ste 1"	Phoenix	AZ	85017	33.524903	-112.115310	1.5	18	1
3	8DShNS-LuFqpEWIp0HxijA	"Sports Authority"	NaN	"5000 Arizona Mills Cr; Ste 435"	Tempe	AZ	85282	33.383147	-111.964725	3.0	9	0
4	PfOCPjBrlQAnz__NXj9h_w	"Brick House Tavern + Tap"	NaN	"581 Howe Ave"	Cuyahoga Falls	OH	44221	41.119535	-81.475690	3.5	116	1

As a preprocessing step, we had to replace commas with semicolon, so that later, when we load the CSV file [separated by commas], the columns will be correctly parsed and saved as a Hive table.

Create Business table in Hive

First, we load the CSV File “clean_yelp_business.csv” into HDFS:
https://drive.google.com/open?id=1OEkQJf2CYk1q8Q28Xx5MNJT2BBQa5a6g
Then, we open Hive 2.0 View on the Hadoop Cluster (under maria_dev) and run the following script:

```
DROP TABLE IF EXISTS temp_table;
CREATE TABLE temp_table (colname STRING)
tblproperties("skip.header.line.count"="1");
LOAD DATA INPATH '/user/maria_dev/yelp/clean_yelp_business.csv' OVERWRITE INTO
TABLE temp_table;
DROP TABLE IF EXISTS business;
CREATE TABLE business (row_index INT,
                       business_id STRING,
        name STRING,
        neighborhood STRING,
                       address STRING,
        city STRING,
        state STRING,
        postal_code STRING,
        latitude FLOAT,
        longitude FLOAT,
        stars FLOAT,
        review_count INT,
        is_open BOOLEAN,
        categories STRING);

INSERT OVERWRITE TABLE business
SELECT
  regexp_extract(colname, '^(?:([^,]*),?){1}', 1) row_index,
  regexp_extract(colname, '^(?:([^,]*),?){2}', 1) business_id,
  regexp_extract(colname, '^(?:([^,]*),?){3}', 1) name,
  regexp_extract(colname, '^(?:([^,]*),?){4}', 1) neighborhood,
  regexp_extract(colname, '^(?:([^,]*),?){5}', 1) address,
  regexp_extract(colname, '^(?:([^,]*),?){6}', 1) city,
  regexp_extract(colname, '^(?:([^,]*),?){7}', 1) state,
  regexp_extract(colname, '^(?:([^,]*),?){8}', 1) postal_code,
  regexp_extract(colname, '^(?:([^,]*),?){9}', 1) latitude,
  regexp_extract(colname, '^(?:([^,]*),?){10}', 1) longitude,
  regexp_extract(colname, '^(?:([^,]*),?){11}', 1) stars,
  regexp_extract(colname, '^(?:([^,]*),?){12}', 1) review_count,
  regexp_extract(colname, '^(?:([^,]*),?){13}', 1) is_open,
  regexp_extract(colname, '^(?:([^,]*),?){14}', 1) categories
from temp_table;

SELECT stars, count(*) as nr_of_businesses FROM business GROUP BY stars ORDER BY
nr_of_businesses DESC;
```

```
39 SELECT stars, count(*) as nr_of_businesses FROM business GROUP BY stars ORDER BY nr_of_businesses DESC;
40
```

Execute | Save As | Insert UDF | Visual Explain

RESULTS | LOG | VISUAL EXPLAIN | TEZ UI

Filter columns

stars	nr_of_businesses
4.0	33492
3.5	32038
5.0	27540
4.5	24796
3.0	23142
2.5	16148
2.0	9320
1.5	4303
1.0	3788

To set up interpreter in Zeppelin (to use Hive tables there for Data Querying):
https://zeppelin.apache.org/docs/0.5.6-incubating/manual/interpreters.html
https://zeppelin.apache.org/docs/0.7.0/interpreter/hive.html
https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.5.3/bk_zeppelin-component-guide/content/using-jdbc-interp.html

After this script, the CSV table from HDFS has been ingested by Hive and doesn't exist there any more.

Business attributes: Preprocessing

```
import pandas as pd
df = pd.read_csv('yelp_business_attributes.csv')
pd.set_option("max_columns", 100)
pd.set_option('display.max_colwidth', -1)
df.head() # recommendation
```

	business_id	AcceptsInsurance	ByAppointmentOnly	BusinessAcceptsCreditCards	BusinessParking_garage	BusinessParking_street
0	FYWN1wneV18bWNgQjJ2GNg	Na	Na	Na	True	Na
1	He-G7vWjzVUysIKrfNbPUQ	Na	Na	Na	Na	Na
2	8DShNS-LuFqpEWIp0HxijA	Na	Na	Na	Na	Na
3	PfOCPjBrlQAnz__NXj9h_w	Na	Na	Na	Na	Na
4	o9eMRCWt5PkpLDE0gOPtcQ	Na	Na	Na	Na	False

Since this table has 82 columns, we manually perform a feature selection: we keep only columns that are most interesting for our analysis.

```
import pandas as pd
df = pd.read_csv('yelp_business_attributes.csv')
pd.set_option("max_columns", 100)
pd.set_option('display.max_colwidth', -1)
df.head() # recommendation
```

	business_id	AcceptsInsurance	ByAppointmentOnly	BusinessAcceptsCreditCards	BusinessParking_garage	BusinessParking_street
0	FYWN1wneV18bWNgQjJ2GNg	Na	Na	Na	True	Na
1	He-G7vWjzVUysIKrfNbPUQ	Na	Na	Na	Na	Na
2	8DShNS-LuFqpEWIp0HxijA	Na	Na	Na	Na	Na
3	PfOCPjBrlQAnz__NXj9h_w	Na	Na	Na	Na	Na
4	o9eMRCWt5PkpLDE0gOPtcQ	Na	Na	Na	Na	False

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 152041 entries, 0 to 152040
Data columns (total 82 columns):
```

```
df2 = df[["business_id",
"ByAppointmentOnly",
"BusinessAcceptsCreditCards",
"GoodForKids",
"WheelchairAccessible",
"Alcohol",
"HasTV",
"RestaurantsGoodForGroups",
"WiFi",
"Smoking",
"DogsAllowed",
"BusinessAcceptsBitcoin",
"Open24Hours"]]
df2.head()
```

	business_id	ByAppointmentOnly	BusinessAcceptsCreditCards	GoodForKids	WheelchairAccessible	Alcohol	HasTV
0	FYWN1wneV18bWNgQjJ2GNg	Na	Na	Na	Na	Na	Na
1	He-G7vWjzVUysIKrfNbPUQ	Na	Na	Na	Na	Na	Na
2	8DShNS-LuFqpEWIp0HxijA	Na	Na	Na	Na	True	Na
3	PfOCPjBrlQAnz__NXj9h_w	Na	Na	Na	Na	Na	Na
4	o9eMRCWt5PkpLDE0gOPtcQ	Na	Na	Na	Na	Na	Na

```
# replace all "Na" values with HIVE standard value for Null, i.e. NULL
df2.replace(to_replace="Na", value="NULL", inplace=True, regex=True)
df2.head()
```

	business_id	ByAppointmentOnly	BusinessAcceptsCreditCards	GoodForKids	WheelchairAccessible	Alcohol	HasTV
0	FYWN1wneV18bWNgQjJ2GNg	NULL	NULL	NULL	NULL	NULL	NULL
1	He-G7vWjzVUysIKrfNbPUQ	NULL	NULL	NULL	NULL	NULL	NULL
2	8DShNS-LuFqpEWIp0HxijA	NULL	NULL	NULL	NULL	True	NULL
3	PfOCPjBrlQAnz__NXj9h_w	NULL	NULL	NULL	NULL	NULL	NULL
4	o9eMRCWt5PkpLDE0gOPtcQ	NULL	NULL	NULL	NULL	NULL	NULL

```
df2.to_csv("yelp_business_attributes_mini.csv", index = False)
```

Create Business attributes table in Hive

First, we load the CSV File “yelp_business_attributes_mini.csv” into HDFS: https://drive.google.com/file/d/1Xi_eBdB1FvA2mumOwnjc-6JNqaT4CgWP/view?usp=sharing
Then, we open Hive 2.0 View on the Hadoop Cluster (under maria_dev) and run the following script:

```
DROP TABLE IF EXISTS temp_table;
CREATE TABLE temp_table (colname STRING) tblproperties("skip.header.line.count"="1");
LOAD DATA INPATH '/user/maria_dev/yelp/yelp_business_attributes_mini.csv'
OVERWRITE INTO TABLE temp_table;
DROP TABLE IF EXISTS business_attributes;
CREATE TABLE business_attributes (
        business_id STRING,
        ByAppointmentOnly STRING,
        BusinessAcceptsCreditCards STRING,
        GoodForKids STRING,
        WheelchairAccessible STRING,
        Alcohol STRING,
        HasTV STRING,
        RestaurantsGoodForGroups STRING,
        WiFi STRING,
        Smoking STRING,
        DogsAllowed STRING,
        BusinessAcceptsBitcoin STRING,
        Open24Hours STRING);

INSERT OVERWRITE TABLE business_attributes
SELECT
        regexp_extract(colname, '^(?:([^,]*),?){1}', 1) business_id,
        regexp_extract(colname, '^(?:([^,]*),?){2}', 1) ByAppointmentOnly,
        regexp_extract(colname, '^(?:([^,]*),?){3}', 1) BusinessAcceptsCreditCards,
        regexp_extract(colname, '^(?:([^,]*),?){4}', 1) GoodForKids,
        regexp_extract(colname, '^(?:([^,]*),?){5}', 1) WheelchairAccessible,
        regexp_extract(colname, '^(?:([^,]*),?){6}', 1) Alcohol,
        regexp_extract(colname, '^(?:([^,]*),?){7}', 1) HasTV,
        regexp_extract(colname, '^(?:([^,]*),?){8}', 1) RestaurantsGoodForGroups,
        regexp_extract(colname, '^(?:([^,]*),?){9}', 1) WiFi,
        regexp_extract(colname, '^(?:([^,]*),?){10}', 1) Smoking,
        regexp_extract(colname, '^(?:([^,]*),?){11}', 1) DogsAllowed,
        regexp_extract(colname, '^(?:([^,]*),?){12}', 1) BusinessAcceptsBitcoin,
        regexp_extract(colname, '^(?:([^,]*),?){13}', 1) Open24Hours
from temp_table;
```

SELECT * FROM business_attributes LIMIT 5;

```
37 SELECT * FROM business_attributes LIMIT 5;
38
```

Execute | Save As | Insert UDF | Visual Explain

RESULTS | LOG | VISUAL EXPLAIN | TEZ UI

Filter columns

business_attributes.business_id	business_attributes.byappointmentonly	business_attributes.businessacceptscreditcards	business_attributes.goodforkids	business_attributes.wheelchairaccessible	business_att
FYWN1wneV18bWNgQjJ2GNg	Na	Na	Na	Na	Na
He-G7vWjzVUysIKrfNbPUQ	Na	Na	Na	Na	Na
8DShNS-LuFqpEWIp0HxijA	Na	Na	Na	Na	True
PfOCPjBrlQAnz__NXj9h_w	Na	Na	Na	Na	Na
o9eMRCWt5PkpLDE0gOPtcQ	Na	Na	Na	Na	Na

Business hours: Preprocessing

```
import pandas as pd
import numpy as np
from datetime import datetime
df = pd.read_csv('yelp_business_hours.csv')

# Parse into start and end
df['mon_start'], df['mon_end'] = df['monday'].str.split('-', 1).str
df['tue_start'], df['tue_end'] = df['tuesday'].str.split('-', 1).str
df['wed_start'], df['wed_end'] = df['wednesday'].str.split('-', 1).str
df['thu_start'], df['thu_end'] = df['thursday'].str.split('-', 1).str
df['fri_start'], df['fri_end'] = df['friday'].str.split('-', 1).str
df['sat_start'], df['sat_end'] = df['saturday'].str.split('-', 1).str
df['sun_start'], df['sun_end'] = df['sunday'].str.split('-', 1).str
```

```
df.head()
```

	business_id	monday	tuesday	wednesday	thursday	friday	saturday	sunday	mon_start	mon_end	...	wed_start	wed_end	thu_start	th
0	FYWN1wneV18bWNgQjJ2GNg	7:30-17:0	7:30-17:0	7:30-17:0	7:30-17:0	7:30-17:0	None	None	7:30	17:0	...	7:30	17:0	7:30	
1	He-G7vWjzVUysIKrfNbPUQ	9:0-20:0	9:0-20:0	9:0-20:0	9:0-20:0	9:0-16:0	8:0-16:0	None	9:0	20:0	...	9:0	20:0	9:0	
2	KQPW8lFf1y5BT2MxiSZ3QA	None	None	None	None	None	None	None	None	NaN	...	None	NaN	None	
3	8DShNS-LuFqpEWIp0HxijA	10:0-21:0	10:0-21:0	10:0-21:0	10:0-21:0	10:0-21:0	10:0-21:0	11:0-19:0	10:0	21:0	...	10:0	21:0	10:0	
4	PfOCPjBrlQAnz__NXj9h_w	11:0-1:0	11:0-1:0	11:0-1:0	11:0-1:0	11:0-1:0	11:0-2:0	11:0-0:0	11:0	1:0	...	11:0	1:0	11:0	

```
# Remove old columns
df.drop(labels=["monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday"], axis=1, inplace=True)

# replace all "None" values with python standard value for Null, i.e. NaN
df.replace(to_replace="None", value="NaN", inplace=True, regex=True)
```

```
# Convert all hours to unix timestamp
# Then calculate the difference: add this difference (i.e. nr of hours that the business has opened each day) as a new column
######  START ########
if df.mon_start.empty:
    df.mon_start = df.mon_start
else: # nanosecond timestamp: ex. 1-01-01 09:00:00
    df.mon_start = pd.to_datetime(df.mon_start, format = "%H:%M")

if df.tue_start.empty:
    df.tue_start = df.tue_start
else:
    df.tue_start = pd.to_datetime(df.tue_start, format = "%H:%M")

if df.wed_start.empty:
    df.wed_start = df.wed_start
else:
    df.wed_start = pd.to_datetime(df.wed_start, format = "%H:%M")

if df.thu_start.empty:
    df.thu_start = df.thu_start
else:
    df.thu_start = pd.to_datetime(df.thu_start, format = "%H:%M")

if df.fri_start.empty:
    df.fri_start = df.fri_start
else:
    df.fri_start = pd.to_datetime(df.fri_start, format = "%H:%M")

if df.sat_start.empty:
    df.sat_start = df.sat_start
else:
    df.sat_start = pd.to_datetime(df.sat_start, format = "%H:%M")

if df.sun_start.empty:
    df.sun_start = df.sun_start
else:
    df.sun_start = pd.to_datetime(df.sun_start, format = "%H:%M")
```

```
######  END ########
if df.mon_end.empty:
    df.mon_end = df.mon_end
else:
    df.mon_end = pd.to_datetime(df.mon_end, format = "%H:%M")

if df.tue_end.empty:
    df.tue_end = df.tue_end
else:
    df.tue_end = pd.to_datetime(df.tue_end, format = "%H:%M")

if df.wed_end.empty:
    df.wed_end = df.wed_end
else:
    df.wed_end = pd.to_datetime(df.wed_end, format = "%H:%M")

if df.thu_end.empty:
    df.thu_end = df.thu_end
else:
    df.thu_end = pd.to_datetime(df.thu_end, format = "%H:%M")

if df.fri_end.empty:
    df.fri_end = df.fri_end
else:
    df.fri_end = pd.to_datetime(df.fri_end, format = "%H:%M")

if df.sat_end.empty:
    df.sat_end = df.sat_end
else:
    df.sat_end = pd.to_datetime(df.sat_end, format = "%H:%M")

if df.sun_end.empty:
    df.sun_end = df.sun_end
else:
    df.sun_end = pd.to_datetime(df.sun_end, format = "%H:%M")
```

```
# Add the diff as new column for each day
index_mon = pd.DatetimeIndex(df["mon_end"]-df["mon_start"])
df["mon"] = (index_mon.hour*60 + index_mon.minute)/60

index_tue = pd.DatetimeIndex(df["tue_end"]-df["tue_start"])
df["tue"] = (index_tue.hour*60 + index_tue.minute)/60

index_wed = pd.DatetimeIndex(df["wed_end"]-df["wed_start"])
df["wed"] = (index_wed.hour*60 + index_wed.minute)/60

index_thu = pd.DatetimeIndex(df["thu_end"]-df["thu_start"])
df["thu"] = (index_thu.hour*60 + index_thu.minute)/60

index_fri = pd.DatetimeIndex(df["fri_end"]-df["fri_start"])
df["fri"] = (index_fri.hour*60 + index_fri.minute)/60

index_sat = pd.DatetimeIndex(df["sat_end"]-df["sat_start"])
df["sat"] = (index_sat.hour*60 + index_sat.minute)/60

index_sun = pd.DatetimeIndex(df["sun_end"]-df["sun_start"])
df["sun"] = (index_sun.hour*60 + index_sun.minute)/60

business_hours = df[["business_id", "mon","tue","wed","thu","fri","sat","sun"]]

business_hours.to_csv("FINAL_business_hours.csv", index=False)
business_hours.head()
```

	business_id	mon	tue	wed	thu	fri	sat	sun
0	FYWN1wneV18bWNgQjJ2GNg	9.5	9.5	9.5	9.5	9.5	NaN	NaN
1	He-G7vWjzVUysIKrfNbPUQ	11.0	11.0	11.0	11.0	7.0	8.0	NaN
2	KQPW8lFf1y5BT2MxiSZ3QA	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	8DShNS-LuFqpEWIp0HxijA	11.0	11.0	11.0	11.0	11.0	11.0	8.0
4	PfOCPjBrlQAnz__NXj9h_w	14.0	14.0	14.0	14.0	14.0	15.0	13.0

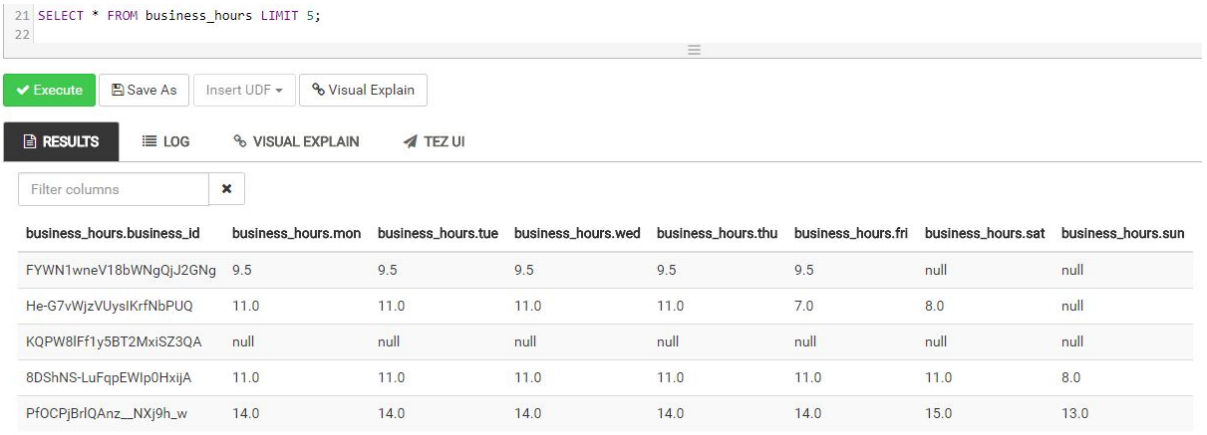
Now we have the desired format and can load this table "FINAL_business_hours.csv": https://drive.google.com/open?id=1mdkS6Ozzjp5FB8qEPsH1WkrZaSQQUtfU into HDFS to use it in Hive.

Create Business hours table in Hive

```
DROP TABLE IF EXISTS temp_table;
CREATE TABLE temp_table (colname STRING) tblproperties("skip.header.line.count"="1");
LOAD DATA INPATH '/user/maria_dev/yelp/FINAL_business_hours.csv' OVERWRITE INTO
TABLE temp_table;
DROP TABLE IF EXISTS business_hours;
CREATE TABLE business_hours (business_id STRING,
        mon FLOAT, tue  FLOAT, wed FLOAT, thu FLOAT,
                        fri FLOAT, sat FLOAT, sun FLOAT);

INSERT OVERWRITE TABLE business_hours
SELECT  regexp_extract(colname, '^(?:([^,]*),?){1}', 1) business_id,
```

```
  regexp_extract(colname, '^(?:([^,]*),?){2}', 1) mon,
  regexp_extract(colname, '^(?:([^,]*),?){3}', 1) tue,
  regexp_extract(colname, '^(?:([^,]*),?){4}', 1) wed,
  regexp_extract(colname, '^(?:([^,]*),?){5}', 1) thu,
  regexp_extract(colname, '^(?:([^,]*),?){6}', 1) fri,
  regexp_extract(colname, '^(?:([^,]*),?){7}', 1) sat,
  regexp_extract(colname, '^(?:([^,]*),?){8}', 1) sun
from temp_table;
```

SELECT * FROM business_hours LIMIT 5;

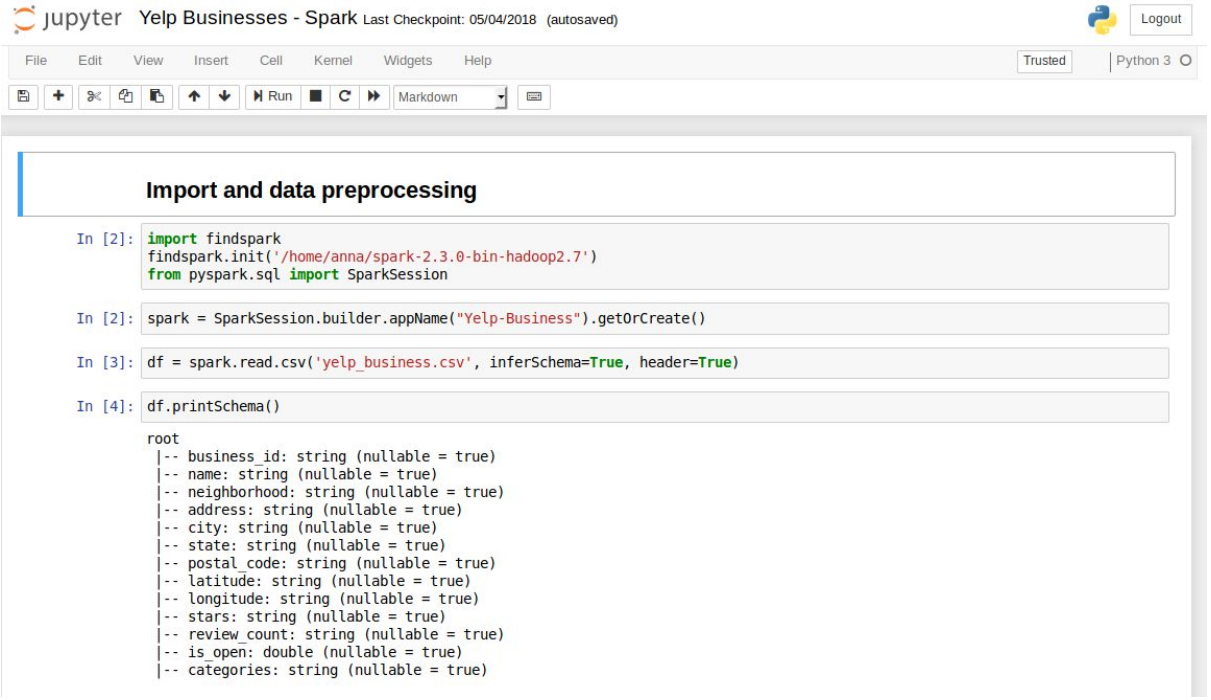


business_hours.business_id	business_hours.mon	business_hours.tue	business_hours.wed	business_hours.thu	business_hours.fri	business_hours.sat	business_hours.sun
FYWN1wneV18bWNgQjJ2GNg	9.5	9.5	9.5	9.5	9.5	null	null
He-G7vWjzVUyslKrfNbPUQ	11.0	11.0	11.0	11.0	7.0	8.0	null
KQPW8lFf1y5BT2MxiSZ3QA	null	null	null	null	null	null	null
8DShNS-LuFqpEWlp0HxijA	11.0	11.0	11.0	11.0	11.0	11.0	8.0
PfOCPjBrlQAnz__NXj9h_w	14.0	14.0	14.0	14.0	14.0	15.0	13.0

Appendix 4.

Yelp with Spark on Ubuntu

In order to test a different environment for Big Data processing, we created a Virtual Machine with Ubuntu, so that we were able to install a stand-alone version of Spark. This step was motivated by an increasing number of problems that we encountered while running Big Data scripts on a local Hortonworks Sandbox installation. Also, we hoped to improve reproducibility of our code by using the same Ubuntu environment on our local machines and sharing Jupyter notebooks for collaborative work.




```
df.createOrReplaceTempView("business")
```

```
spark.sql("SELECT DISTINCT categories FROM business").show(truncate=False)
```

```
+----------------------------------------------------------------------------------------------------------------------------------
------+
|categories
|
+----------------------------------------------------------------------------------------------------------------------------------
------+
|Shopping;Shopping Centers
|
|Nail Salons;Beauty & Spas;Waxing;Hair Removal
|
|Vape Shops;Tobacco Shops;Shopping
|
|Nightlife;Breweries;Restaurants;Food;American (New);Bars
|
|Home Services;Building Supplies;Hardware Stores;Shopping;Home & Garden;Wholesale Stores
|
|Pizza;American (Traditional);Restaurants;Gluten-Free
|
|Shopping;Cosmetics & Beauty Supply;Jewelry;Shoe Stores;Department Stores;Men's Clothing;Beauty & Spas;Fashion;Women's Cl
othing|
|Shopping;Eyewear & Opticians;Fashion;Local Services;Appliances & Repair;Accessories
|
|Fast Food;Portuguese;Chicken Wings;Restaurants
|
|Sewing & Alterations;Dry Cleaning & Laundry;Dry Cleaning;Local Services;Laundry Services
```

```
|
|Chinese;Ramen;Japanese;Barbeque;Restaurants;Vietnamese
|
|Pet Sitting;Pet Services;Pets
|
|Tires;Oil Change Stations;Automotive;Auto Repair
|
|Korean;Barbeque;Restaurants;Asian Fusion
|
|Pet Services;Dog Walkers;Pets
|
|Bars;Gastropubs;Restaurants;Nightlife
|
|Auto Repair;Auto Parts & Supplies;Automotive
|
|Bars;Pubs;Nightlife
|
|Local Services;Shopping;Carpeting;Rugs;Home Decor;Carpet Cleaning;Home Services;Home & Garden
|
|Musical Instrument Services;Guitar Stores;Local Services;Musical Instruments & Teachers;Shopping
|
+--------------------------------------------------------------------------------------------------------------------
------+
only showing top 20 rows
```

```
import pandas as pd
df2 = df.toPandas()
df2.head(3)
```

	business_id	name	neighborhood	address	city	state	postal_code	latitude	longitude	stars	review_cou
0	FYWN1wneV18bWNgQjJ2GNg	"""Dental by Design"""	None	"""4855 E Warner Rd	Ste B9"""	Ahwatukee	AZ	85044	33.3306902	-111.9785992	4
1	He-G7vWjzVUysIKrfNbPUQ	"""Stephen Szabo Salon"""	None	"""3101 Washington Rd"""	McMurray	PA	15317	40.2916853	-80.1048999	3.0	1
2	KQPW8IFf1y5BT2MxiSZ3QA	"""Western Motor Vehicle"""	None	"""6025 N 27th Ave	Ste 1"""	Phoenix	AZ	85017	33.5249025	-112.1153098	1

To sum up, Ubuntu was working properly, but it had two significant disadvantages:

1. The code was still running on a single node. It was sufficient to learn the syntax of Spark, but it was not enough to run some complex computations as those would require a distributed processing.
2. Spark, installed as a stand-alone version, cannot display visualizations on the objects generated during the Spark Session. The only way was to convert each Spark DataFrame to a simple Pandas data frame and create visualizations in a plain Python code. However, at this point, Irina discovered a much better solution: distributed cluster on the Microsoft Azure Cloud, which later turned out to be our central working environment.