for reddit Comment Sentiment Analysis

Project 1, Phase 2: NoSQL Storage Proof-of-Concept Write-Up

November 2nd, 2018

Brought to you by
DATANAUTS INC.

Julia Garbuz,
Cory Koster,
Jonathan Persgården,
and Daniel Wu

CSCI 5980 (Section 002), Fall 2018
Professor: Donald Sawyer

Table of Contents:

1 Overview

1.1 Background Information

Reddit is a social media website consisting of communities and sub-communities ("subreddits") where people can have discussions on various topics. Contributors can make posts with text, images, gifs, and emojis in the discussion. Reddit members are not confined to one topic or discussion and can participate in as many discussion threads or sub-communities as they want. As they post, users may interact with other members by responding to comments or posts as well as voting "up" or "down" on them. The more users contribute and post, the more interactions they will have with an increasingly greater number of people.

1.2 Purpose / Business Question

DATANAUTS INC. wants to explore the relationship of a user's sentiment in their comments to a variety of fields, including age of user, number of years using Reddit, which Subreddit the comment was in, and number of up/down votes on the comment.

Particularly, the goal of this project is to research, explore, and analyze these two more-involved relationships:

(1) The changes in a user's sentiment over time spent as a Reddit user.

(2) The effects on a user sentiment in a particular subreddit after participating in other subreddits with a differing general sentiment

1.3 Solution Overview

From the perspectives of speed, ease of addition of supplementary information, logical alignment to the current questions, and ability to handle high-velocity real-time data, DATANAUTS INC. believes that the best technology for performing this analysis would be the NoSQL graph database, **JanusGraph**. However, the core data to be used for the Reddit comment sentiment analysis is currently stored in Google's **BigQuery**, a NoSQL column-family database. The data will be extracted, transformed, and then finally loaded in large batches by **Python** scripts. The transformation processes will include analyzing each comment's sentiment using an emotion and sentiment lexicon and then supplementing each comment with additional information from Reddit.

2 Team Logistics

2.1 Team Background

DATANAUTS INC.'s team members were assembled completely randomly resulting in a group of very diverse backgrounds and experience levels.

The team consists of one undergraduate, two masters, and one PhD student. Three of the four team members live off-campus, one of which lives in another state, and two of the team's members have full-time jobs while the others work part-time. This led to a lot of difficulty in scheduling meeting times that worked for everyone and made it virtually impossible to meet in person.

2.2 Approach

DATANAUTS INC. took an agile-like approach by planning incremental goals for approximately one-week sprints.

2.2.1 Distribution of Workload

Every member was assigned a component of the goal for the "sprint". For example, in the first "sprint", the goal was to implement the basic functionality of each of the data-manipulation scripts. Every member was assigned one of the four components.

2.2.2 Communication and Collaboration

DATANAUTS INC. scheduled official bi-weekly stand-ups to check-in on other team member progress and communicated on how the pieces would all work together. One of the stand-ups was via video call and the other was in person.

In addition to this, DATANAUTS INC. also communicated between these official meetings via Slack and other video calls as needed.

In terms of collaboration, the code and set-up instructions were all pushed to a shared GitHub repository.

2.3 Analysis

DATANAUTS INC. encountered a few logistical issues when it came to meeting and performing the work. The team had some hurdles relating to distance and timing of meetings; however, when meetings were able to be held, the team worked well together and got along without issue. The work was fairly divided at the start, and when some problems were encountered that forced us to go a different route, members of the team stepped forward to take on the modified workload.

The biggest problem DATANAUTS INC. encountered was a lack of communication at times and work sometimes being done in isolation. Every team member is busy with work, other classes, and life in general, so it wasn't expected to be available at all hours of the day. There were times though that team members did not respond to group messages for several days and the rest of the team were left wondering where the project was at.

If the project were restarted with the same criteria, DATANAUTS INC. would start smaller and work more collaboratively on major sections of the project so not one person was in charge of a major piece of the architecture. In this way, those issues encountered the first time would hopefully be avoided.

3 Technology and System Design

3.1 **Overview of Primary Data Source:** Reddit Comment Dataset from BigQuery

The primary source of data for the analysis was a real-time database of every publicly available Reddit user comment. This data has already been collected and stored in a Google BigQuery database for others' use and analysis.

The Reddit comment dataset was over 250 GBs compressed so it is far beyond what personal computers are capable of handling. Additionally, with BigQuery being a columnar database, it would be very computationally expensive to try to perform the desired relationships analyses keeping the data in its current form.

3.2 **Overview of Primary Data Storage Technology:** JanusGraph

JanusGraph, first released in 2017, is a scalable graph database optimized for massive amounts of data distributed over multiple nodes. It is a fork of Titan, a Graph Database technology that was eventually bought by DataStax and converted into an enterprise product.

One of the main benefits of JanusGraph is the "pluggable" nature of utilizing different back-end and search technologies like Cassandra, HBase, Solr, and ElasticSearch. Because of the ability to customize the backend of JanusGraph, one can tune properties of the database to suit business needs, such as consistency. One of the differentiating factors of JanusGraph compared to Neo4j is that JanusGraph can be configured to be eventually consistent (through a Cassandra backend) compared to Neo4j being a strongly consistent database.

One of the primary reasons DATANAUTS INC. opted to use JanusGraph is that all features are freely available. The initial plan was to utilize Memgraph, however, Memgraph only provided "horizontal-scalability" with the enterprise version, so the system had to be adjusted to utilize JanusGraph. Similarly, a downside of Neo4j is that some features required in Big Data production environments, like clustering, are only available on its enterprise version.

On the other hand, some of the primary difficulties of the technology include its relative immaturity and the lack of community support of this particular open source technology. The original Titan project was shut down and JanusGraph has not become widely adopted as a successor. Neo4j, however, is widely considered the most popular graph database, and has a lot of community support and tutorials including a Neo4J browser which has an intuitive friendly web interface.

JanusGraph has support for Gremlin and TinkerPop drivers for transactional functions and analysis with plenty of configuration and optimization options. Additionally, JanusGraph provides the option for OLAP analysis to read and write data utilizing a Spark cluster to parallelize data loading.

3.3 System Design

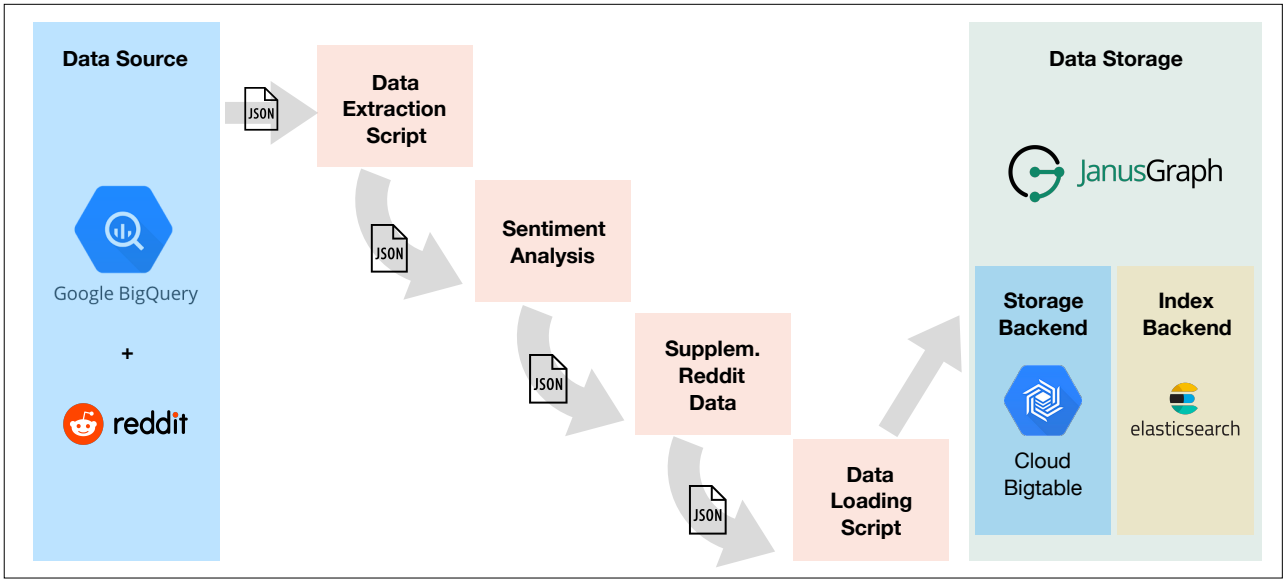

Diagram 3.3 – 1: System Diagram and Data Flow

Referencing the POC Proposal, the original plan was to stream the data from BigQuery to JanusGraph (then Memgraph) via Pulsar, processing the data transformation in separate steps.

Despite the significant amount of time spent working trying to get Apache Pulsar deployed, the documentation failed to help us resolve the configuration issues. For this reason DATANAUTS INC. selected to default to batch processing the data step-by-step.

As seen in **Diagram 3.3 – 1**, each batch of data will be initially extracted by the “data-extraction” script. The output of that script is then sent on to “sentiment-analysis” and likewise on to “supplementary-reddit-data” to be enriched until finally reaching “data-load” where it is loaded into JanusGraph.

The details of each of these steps are outlined below:

3.3.1 **Data Extraction** from Google BigQuery

BigQuery is a service provided by Google and is a part of the Google Cloud Platform. Therefore, the first step required to use BigQuery is to create a Google Cloud Platform account.

Once the account is created, to be able to authenticate the account from the script, a service account key is needed. This key can be acquired within the Google Cloud Platform and is then downloaded as a JSON document and referenced from within the script.

Google provides the package "google-cloud-bigquery" for their BigQuery API in Python (among many other programming languages). Queries are constructed using SQL-like syntax. **Code Snippet 3.3.1 – 1** demonstrates how to create a BigQuery client using the service account key acquired earlier ('key.json') and then perform a simple query.

```python
from google.cloud import bigquery

#Set up the client
bigquery_client = bigquery.Client.from_service_account_json('key.json')
#Run a query
query = 'SELECT * FROM `fh-bigquery.reddit_comments.2015_05`'
query_job = bigquery_client.query(query)
#Wait for results
raw_data = query_job.result()
#Print the body attribute of each row.
for row in raw_data:
    print(row.body)
```

Code Snippet 3.3.1 – 1: Google BigQuery Sample Set-Up and Query

This script is the initial data-extraction phase of the entire system where the data will be broken into batches for later processing. The initial batch was extracted by merely limiting the results of the "SELECT * FROM" query. However, there is no way to begin the following query from where the previous left off, so to perform more than on batch extraction it is important to stream the data as it is extracted. This is why streaming the data via Apache Pulsar would have been an ideal option, but for now, the data was just extracted in one large batch.

Once the batch of data is extracted from Google BigQuery, it is written to a JSON document with one record per line for the following script(s) to process. Details about the form and fields of the Reddit Comment data can be found in **Appendix A**.

For more detailed service account key and "google-cloud-bigquery" package set-up instructions, or to view the whole data-extraction script see https://github.umn.edu/garbu007/reddit-sentiment-analysis/tree/master/data_extraction.

3.3.2 Data Transformation

The next step of the process is to enrich the existing Reddit Comment data. The order of the next two sub-steps are not of any particular importance but in this particular implementation the sentiment and emotion analysis was performed first to be able to drop the comment itself and then the supplementary Reddit data was collected.

3.3.2 (a) Sentiment and Emotion Analysis

The sentiment analysis is performed in accordance with the Sentiment and Emotion Lexicons provided here: http://sentiment.nrc.ca/lexicons-for-research/. This dataset is significantly smaller and is used only as a supplement for analysis. Each line of the text file (downloaded from the link above) includes a term, list of emotions or sentiments, and an indicator (0 or 1) to signify if the given word has an association with the given emotion or sentiment.

This dataset will be read directly from the original text file and to then process the body of each comment. Set up instructions, a complete data dictionary, and a data sample for the Sentiment and Emotion Lexicon dataset are provided in **Appendix B**.

The sentiment analysis script will process the output of the data extraction step. For each record, the "`body`" field will be analyzed to calculate a "`sentiment-score`". The "`sentiment-score`" is calculated as a sum of the sentiment scores for each word divided by the number of words in the comment. This field will then replace the "`body`" field as the data is written to a new output file and passed on to the following process.

For set-up/usage instructions and full code see https://github.umn.edu/garbu007/reddit-sentiment-analysis/tree/master/sentiment-analysis.

3.3.2 (b) Supplementary Reddit Data

The next step in the enrichment and transformation stage is to acquire the "`created_utc`" for each Reddit user (date the account was created). This field is important to the analysis of Reddit user sentiment over time.

The first step of this is to create a Reddit account and acquire developer credentials to access Reddit's API. Detailed instructions are provided here: https://github.com/reddit-archive/reddit/wiki/OAuth2, but only the "Getting Started" steps need to be completed.

Once the credentials have been created, they must be added to a "praw.ini" file that the script can then reference to authenticate the client.

Reddit's API is called using the PRAW (Python Reddit API Wrapper) package. To read more about PRAW visit https://praw.readthedocs.io.

```python
import praw

# Reddit user for which we are searching:
comment_author = "datanauts-inc"

# Reference to OAuth credentials (from praw.ini):
# (See detailed set-up instructions for praw.ini configuration)
praw_config_site = "supplementary-reddit-data"

# Create Reddit instance:
reddit = praw.Reddit(praw_config_site)

# Get user, and then 'created_utc'
user = self.__reddit.redditor(comment_author)
user_created_utc = user.created_utc
```

Code Snippet 3.3.2 – 1: Getting Reddit user's "created_utc" using PRAW

Code Snippet 3.3.2 – 1 demonstrates how simple it is to get a Reddit user's "`created_utc`" using PRAW. For detailed information about the fields of a Reddit user object, see **Appendix C**.

The supplementary-reddit-data script batch processes the output of the sentiment-analysis script. Initially, it reads in the data and splits it into

sub-batches for parallel processing. Next, each process creates an instance of PRAW and collects the user's "`created_utc`".

A couple of issues were encountered at this stage. Firstly, some Reddit users have since been deleted and cannot be found by Reddit's API. When this was encountered, the record was "dropped" and only a message stating that the record was dropped was passed on.

Secondly, some users (for example those who were suspended) can still be found by Reddit's API, but they are lacking information such as their "`created_utc`". This is the reason that the boolean flag "added_`created_utc`" was added. In cases where the user and "`created_utc`" were successfully found this was set to 'True' and the record was appended with both this flag and the "`created_utc`" itself. Otherwise, this flag was set to 'False', this was appended to the record, and it was passed on without a "`created_utc`" field.
Once all processes completed processing all of their records, they were written to a new output file to pass on to the final step and the counts of dropper records were recorded for future reference.

For detailed set-up/usage instructions and complete code see https://github.umn.edu/garbu007/reddit-sentiment-analysis/tree/master/supplementary-reddit-data.

One major change to this component since the proposal is that DATANAUTS INC. had initially intended to collect both user's "`created_utc`" and also birthday via BeautifulSoup web-scraping. After more research, it was discovered that "cake day" is not the user's date of birth but rather the day their Reddit account was born (equivalent to "`created_utc`"), so this required adjustments to the business questions initially considered.

3.3.3 **Data Loading** into JanusGraph

To emulate a production environment, the cluster was deployed on the Google Cloud Platform (GCP). Utilizing Google Bigtable as our storage backend and Elasticsearch as our indexing backend, we deployed a two-node (4 vCPUs, 15GB memory) cluster in GCP's Compute Engine. Using Kubernetes and Helm (a package manager for deploying applications with Kubernetes), JanusGraph was configured and setup to utilize a Google Bigtable backend and a local deployment of Elasticsearch. A web-socket endpoint was configured to enable remote access to the cluster. Configuration of the cluster was defined through property and YAML files.

An important configuration that was not discovered until later on is the `QueryExecutionTimeout` parameter, which is set to 30000 ms by default. This led to serious limitations when querying the database and emphasized the necessity to index important properties and lean on the indexes in constructing queries.

For detailed set-up instructions please see https://github.umn.edu/garbu007/reddit-sentiment-analysis/blob/master/data-loading/Setup%20for%20JanusGraph%20on%20Google%20Cloud%20BigTable.md.

Schema and data modeling in JanusGraph is limited but straightforward. While a schema can be implicitly defined, it is encouraged to explicitly define the graph schema, which later became useful for defining particular data types (like floats and integers) explicitly.

One aspect of schema definition that was a limitation in JanusGraph was the inability to define the primary key of a node. Instead, the primary key is predefined by JanusGraph at the time of creation of the node. Additionally, the schema of labels and properties cannot be deleted, only renamed, which can result in a dirty schema that would not be ideal in a production environment.

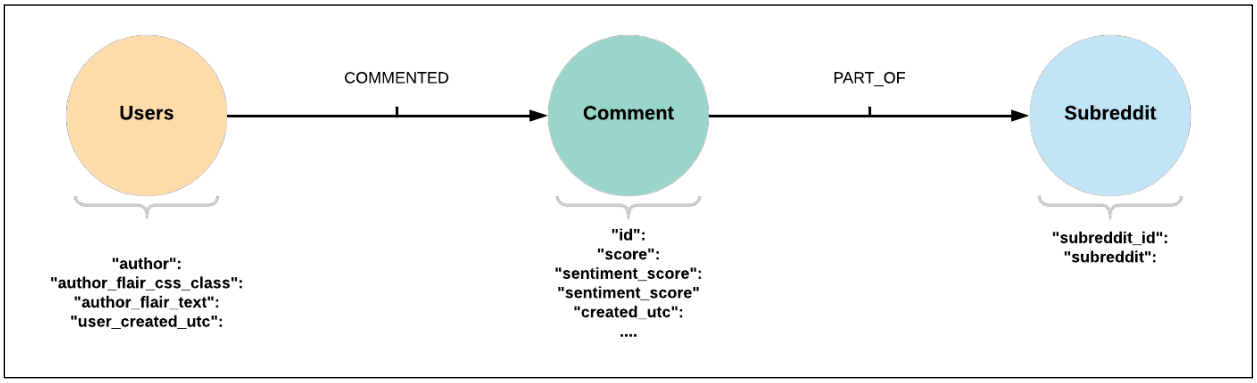

Diagram 3.3.3 – 1: JanusGraph Schema

The schema (as seen in **Diagram 3.3.3 – 1**) was designed around the use cases of the proposed "business questions". Since the goal was to observe the relationship between users, their comment sentiment, and which subreddits those comments are in those will be the three types of nodes represented in the graph. The relationships between them are represented by the edges "COMMENTED" and "PART_OF". One example of the "PART_OF" relationship is displayed in **Diagram 3.3.3 – 2** below.

The last, but very important, part of the set-up is to properly define indexes in JanusGraph to improve query and traversal time. JanusGraph supports two different kinds of indexes: graph indexes and vertex-centric indexes. Graph indexes are indexes on particular property keys of vertices or edges for

selective filters and queries. Vertex-centric indexes are structures built individually per vertex to speed up traversals where there are significant numbers of incident edges. DATANAUTS INC. utilized graph indexes. In retrospect, because subreddit and author nodes can potentially have thousands of incident edges, vertex-centric indexes may have assisted with the queries that timed out.

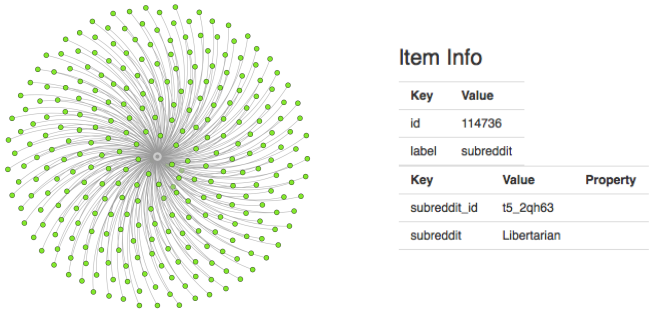

Diagram 3.3.3 – 2: All comments “PART_OF” subreddit “Libertarian”

There are two main types of graph indexes that JanusGraph supports: composite and mixed. Composite indexes are useful for keys that will be retrieved by an exact value match. An additional benefit of composite indexes is that you can enforce a uniqueness constraint, effectively utilizing the property indexed as a pseudo-primary key for preventing a duplicate node from being saved in the database. After analyzing the business questions, DATANAUTS INC. agreed it would make sense to use mixed indexes.

Mixed Indexes utilize the index backend, Elasticsearch, for faster text string searches or inequality constraints. Mixed Indexes allow for fuzzy searches, regex searches, and numerical inequality constraints on property keys, which was useful in being able to filter on sentiment score, comment timestamps, and comment score.

See https://github.umn.edu/garbu007/reddit-sentiment-analysis/blob/master/data-loading/schema_creation.md for schema and index creation scripts .

Now that JanusGraph is entirely set up and configured, it is ready for the batch processing. Python was used to construct gremlin queries and submit them through a web socket endpoint and a driver provided by the gremlin-python package. The data-loading script lays out how to both connect to JanusGraph in Python and how the data was batch-loaded into the database: https://github.umn.edu/garbu007/reddit-sentiment-analysis/blob/master/data-loading/janusgraph_batch_transaction_script.py.

3.3.4 Data Analysis

The final step in the system design was to analyze the data after it was ingested into JanusGraph. Because it was difficult to find and obtain a free and available graphical user interface and subsequent visualization for the analytical results for JanusGraph, it was necessary to submit all queries via a remote terminal connection to the Gremlin console in Google Cloud Platform. The queries were written directly in the console using the query language Gremlin, which is a path-oriented traversal language. The syntax of Gremlin is not like a traditional database query language. Once the basic structures of the language were established, the next difficulty was in understanding how the data of the graph is handled at each point in a graph traversal. At times the data types and formats were ambiguous, causing queries to fail or produce results that were unexpected.

An additional issue DATANAUTS INC. experienced during data analysis occurred after having loaded about 23,000 vertices and over 25,000 edges. After that point, almost every query submitted from the local remote client connection to the server timed-out. The timeout period seemed to be a default setting within the database system, but because JanusGraph was installed using Helm, it was difficult to find the correct file to modify to adjust this. This issue could also have been related to the fact that the version installed was not the most recent. It was found that the timeout could be avoided by limiting the number of vertices searched to a subset of the total, typically anywhere from 5,000 to 15,000, depending on the query. While limiting the search set was not ideal for finding accurate results to the business questions, it did allow DATANAUTS INC. to formulate the logic that can be applied to the full data once these issues are resolved.

For set-up/connection instructions and copies of the queries run, please see https://github.umn.edu/garbu007/reddit-sentiment-analysis/tree/master/gremlin-data-analysis.

4 Findings of Data Analysis

4.1 Easy Questions

- **Comments per user:**
  Minimum: 16
  Maximum: 1
  Average: 1.3

- **Number of comments per sentiment:**
  Positive: 4286
  Negative: 3207
  Neutral: 4856

- **Ratio of positively to negatively classified comments:**
  4286 : 3207 ==> ~1.3 positive comments per negative comment

- **Number of users per "age" (time since joining Reddit) group**:
  < 1 year: 0
  1 - 2 years: 0
  2 - 3 years: 0
  3 - 4 years: 1476
  4 - 5 years: 1916
  5+ years: 5496

- **Most popular subreddit** (by number of comments)
  1355 comments, followed by 1352

4.2 Moderately Difficult Questions

- **Correlation of Reddit usage to more positive or negative sentiment:**
  This proved to be difficult to answer for the entire data set, but sentiment appeared to be slightly more positive over time on the data that was observed.

- **Relationship of sentiment of comment to number of up/down votes:**
  Positively classified comments had 1.55x more upvotes than negatively classified comments.

4.3 Challenging Questions

- **Changes in a user's sentiment over time spent as a Reddit user:**
  This was also difficult to answer. The query finds the sentiment over time grouped by user and could be used to analyze the sentiment of comments for any user in particular, but there wasn't an easy to analyze this for the whole user base.

5 POC Analysis and Lessons Learned

The POC was an exciting and challenging project. DATANAUTS INC encountered many issues and problems and attempted to overcome them. In the end, the team had many takeaways from the project and what it takes to implement a production-level NoSQL database.

One lesson learned was to never assume a system's documentation is complete, no matter how well it appears to be laid out. The team learned this difficult lesson when attempting to install and run Apache Pulsar on a cluster. The documentation appeared thoughtful and designed as a step-by-step walkthrough; however, it ended up being scattered and incomplete. The team never successfully installed Pulsar in a production-level environment.

Another lesson the DATANAUTS INC. learned is that, even for a POC, it is essential to not take on too much. There were many aspects of this project that could have been considered a smaller POC in its own way. Being ambitious is a great characteristic, but this must be tempered when taking on these types of technologies, especially for the first time.

Related to the challenge of taking on too much is the idea of starting small. DATANAUTS INC. set out with a lofty goal and probably would have benefited from starting smaller. An example would have been to begin with one day of Reddit data, pass it through the pipeline as a batch process, load it into JanusGraph, and query the database to ensure the basic functionality was working correctly throughout the system. When confident the pieces were in place, the team could have started to expand, such as working to automate the process, building small components to the pipeline, and ensuring the entire pipeline functioned correctly before moving on.

The significance of the business questions remained central throughout the entirety of the POC. While the technologies were fun to use and learn, and there always seemed like an additional component or technology that could be added to the pipeline to provide additional functionality, it was key to remember the goal of the POC: to answer the business questions. It was easy to lose sight of this at times, but the DATANAUTS INC. did a good job of keeping each other on point.

Moving forward, DATANAUTS INC. would like to continue improving their system by working to implement Apache Pulsar again to stream the data end-to-end, resolve some configuration limitations in JanusGraph, and lastly, implement visualizations for the business questions.

6 Appendix

Appendix A **Reddit Comment Data:** Data Dictionary and Data Sample

Data Dictionary:

Reddit Comments JSON Data Dictionary	
"gilded"	*String:* the number of times this comment received reddit gold
"author_flair_text"	*String:* the text of the author's flair. subreddit specific
"author_flair_css_class"	*String:* the CSS class of the author's flair. subreddit specific
"retrieved_on"	*Int:* UNIX timestamp of date comment was retrieved
"subreddit_id"	*String:* the id of the subreddit in which the thing is located
"subreddit"	*String:* subreddit of thing excluding the /r/ prefix. "pics"
"parent_id"	*String:* ID of the thing this comment is a reply to, either the link or a comment in it
"edited"	*Boolean/Timestamp:* false if not edited, edit date in UTC epoch-seconds otherwise.
"controversiality"	*Int:* Number of flagged comments
"body"	*String:* the raw text. this is the unformatted text which includes the raw markup characters
"created_utc"	*String?*: UNIX timestamp in UTC timezone.
"downs"	*Int:* Number of down votes
"score	*Int:* Summation of 'ups' and 'downs'
"author"	*String:* Reddit account name of person who wrote the comment
archived	*Boolean:* 'true' if comment is archived.
"distinguished"	*Boolean:* add a "distinguish" button to your post to mark it as official
"ups"	*Int:* Number of up votes
"id"	*String:* ID of the account
"score_hidden"	*Boolean:* Whether the comment's score is currently hidden
"name"	*String:* name of the username
"link_id"	*String:* ID of the link this comment is in

Data Sample:


```
{
   "gilded":0,
   "author_flair_text":"Male",
   "author_flair_css_class":"male",
   "retrieved_on":1425124228,
   "ups":3,
   "subreddit_id":"t5_2s30g",
   "edited":false,
   "controversiality":0,
   "parent_id":"t1_cnapn0k",
   "subreddit":"AskMen",
  "body":"I can't agree with passing the blame, but I'm glad to hear it's at least helping you with the anxiety. I went the other direction and started taking responsibility for everything. I had to realize that people make mistakes including myself and it's gonna be alright. I don't have to be shackled to my mistakes and I don't have to be afraid of making them. ",
   "created_utc":"1420070668",
   "downs":0,
   "score":3,
   "author":"TheDukeofEtown",
   "archived":false,
   "distinguished":null,
   "id":"cnasd6x",
   "score_hidden":false,
   "name":"t1_cnasd6x",
   "link_id":"t3_2qyhmp"
}
```

Appendix B **Sentiment and Emotion Lexicons:** Set-up, Data Dictionary and Sample

Set-Up Instructions:

This dataset is available as a text file and can be downloaded from the Sentiment and Emotion Lexicons website (http://sentiment.nrc.ca/lexicons-for-research/).

After navigating to the homepage, the first lexicon listed under the Sentiment and Emotions Lexicons table, *NRC Word-Emotion Association Lexicon*, is the version used. Selecting this version will provide more information about the lexicon as well as the non-commercial licensed link. Accessing this licensed version will provide a link to download the compressed data.

Once downloaded and unzipped, the folder will contain several lexicons pertaining to different word types associations. The file is located in the folder *NRC-Sentiment-Emotion-Lexicon-v0.92*. The file is named *NRC-Emotion-Lexicon-Wordlevel-v0.92.txt*.

Data Dictionary:

Lexicon Sentiment Data Dictionary	
term	*string;* word for which emotion associations are provided
affect category	*string;* one of eight emotions (anger, fear, anticipation, trust, surprise, sadness, joy, or disgust) or one of two polarities (negative or positive)
association flag	*int;* one of two possible values: 0 or 1. 0 indicates that the target word has no association with affect category, whereas 1 indicates an association

Data Sample:

```
accolade  negative  0
accolade  positive  1
. . .
```

Appendix C **Reddit User Object (from Reddit API):** Data Dictionary

Reddit User (or "Redditor") Object:

Attribute	Description
`comment_karma`	The comment karma for the Redditor.
`comments`	Provide an instance of **`SubListing`** for comment access.
`created_utc`	Time the account was created, represented in Unix Time.
`has_verified_email`	Whether or not the Redditor has verified their email.
`icon_img`	The url of the Redditors' avatar.
`id`	The ID of the Redditor.
`is_employee`	Whether or not the Redditor is a Reddit employee.
`is_friend`	Whether or not the Redditor is friends with the authenticated client.
`is_gold`	Whether or not the Redditor has active gold status.
`link_karma`	The link karma for the Redditor.
`name`	The Redditor's username.
`subreddit`	If the Redditor has created a user-subreddit, provides a dictionary of additional attributes. See below.
`subreddit['banner_img']`	The url of the user-subreddit banner.
`subreddit['name']`	The name of the user-subreddit (prefixed with 't5').
`subreddit['over_18']`	Whether or not the user-subreddit is NSFW.
`subreddit['public_description']`	The public description of the user- subreddit.
`subreddit['subscribers']`	The number of users subscribed to the user-subreddit.
`subreddit['title']`	The title of the user-subreddit.

For additional information, see PRAW documentation: https://praw.readthedocs.io/en/latest/code_overview/models/redditor.html#praw.models.Redditor