DATASCI W205 Final Project Report
April 20, 2016
Omar Al Taher || Peter Zhou || Jen Jen Chen

Personalized Doomsday Clock

Section 1: Background

Originally created in 1947, the Doomsday Clock was designed by the Bulletin of the Atomic Scientists, a group founded in 1945 by the Manhattan Project scientists who "could not remain aloof to the consequences of their work." At the time, it was analogous to the threat of global nuclear war. Over decades, the clock has evolved to include other human-driven factors that could cause irreversible harm to humanity, such as climate change and geopolitical developments. Today, the Doomsday Clock is a visual representation of the likelihood of a human-caused global event. Since its original creation, depicted at 11:53pm, it has been changed 22 times since then and most recently following President Trumps comments regarding nuclear involvement and his administration's skepticism over climate change, has set the clock currently at 11:57:30pm.

The current political climate and polarization of the American public led our group to consider development of a personalized Doomsday Clock. Recently this year, the American Psychological Association released the results of several surveys evaluating the degree of stress in the American public. They found that over the last year, there have been rising stress levels due to a variety of reasons, such as uncertainty about the future of the country, rapidly changing technology, political events, and effects of social media. It is often difficult for the public to judge their threat level because the information may be skewed or inaccurate, with much doubt cast on the media given the advent of "fake news."

To alleviate this confusion and uncertainty, we implemented a personalized Doomsday Clock which illustrates the user's "danger" levels and is intuitively easy to understand. However, instead of focusing on nuclear extinction as the original authors did, we combined the use of periodically updated databases with real-time social media platform to create a personalized Doomsday Clock for the user based on his or her location. Functionally, the user simply has to input his/her zip code, and based on their geographic location, the clock returns an assessment of the threat level he or she faces

at any given moment and location.

Section 2: Data Sources

All static datasets are downloaded from their respective sources at the time of the user running the program. Static data was downloaded directly from the source unless the source required user input (i.e. specification of state or length of time), in which case the data was manually downloaded and then uploaded to a github repository.

- Arrests[1]

  Our arrest dataset comes from the State of California Department of Justice. This dataset contains arrests from 2005 to 2014 by the county. An arrest is considered to be a person taken in custody, the person does not necessarily need to be convicted. This data contains many different kinds of crimes. We decided to focus on the more serious offenses and thus went with felonies. We are presenting the data per 1,000 people In order to make the numbers easier to interpret for the reader.

- Bridge Integrity[2]

  The National Bridge Inventory Database is a compilation of bridge data supplied by the states to the Federal Highway Administration for bridges on public roads. The data is updated annually and our dataset contains information through 2016. The challenge presented to us by this dataset was the lack of a complete downloadable file from their website. Instead we scraped the website and gathered information by county and then merged the individual files into a complete file for the state of California which we are able to then load into HDFS and Hive.

  For this dataset, we are focusing on bridges they categorized as functionally obsolete and structurally deficient. Functionally obsolete refers to bridges that are no longer functionally adequate for its task (ex: lack of appropriate number of lanes, lack of space for emergency vehicles, etc). Structurally deficient refers to bridges that have one or more structural defects that require attention.

- Fire[3]

---

[1] https://oag.ca.gov/sites/all/files/agweb/pdfs/cjsc/stats/arrest_data_2005-2014.zip

[2] http://nationalbridges.com/index.php

[3] https://firms.modaps.eosdis.nasa.gov/active_fire/viirs/text/VNP14IMGTDL_NRT_USA_contiguous_and_Hawaii_7d.csv

The Fire Information for Resource Management Systems (FIRMS) is a program funded by NASA and the United Nations that provides near real-time active fire locations by using satellites to detect infrared radiations from fires. We are gathering the last 7 days of fires for this project. The challenges faced by FIRMS include fires that start and stop before the satellite passes over the fire, clouds, trees and other obstructions, and if the fire was too small or too cold to be detected.

This dataset is organized by longitude and latitude. It provided an extra challenge for us since we are using the zip code as the primary key. To join the tables, we first filter for fires that are within 1.0 longitude and latitude of zip codes within California. We then use the zip code that has the minimum distance between the fire and the longitude and latitude provided in our zip code table.

- Landslides[4]

  Our landslides dataset comes from the Global Landslide Catalog held by NASA. They gather the mass movements of land triggered by rainfall that are reported by media, disaster databases, scientific reports, and other credible sources. The dataset that we are looking at is from 2008 through 2014.

- Death[5]

  The California Department of Public Health publishes the cause of death for California residents annually. This dataset, last updated February 21, 2017, is from 1999 to 2013. The death count is represented by the zip code of residence regardless of the place of the death occurrence. This also does not include deaths from non-California residents. We are reporting the top three leading causes of death in each zip code.

- Air Quality[6]

  The Environmental Protection Agency implemented a Tracking Network to provide information regarding air pollution, specifically ozone and particulate matter (PM2.5) to the Centers for Disease Control and Prevention. The data is collected from roughly 2000 monitoring stations around the country and focuses more on urban and heavily polluted areas. The monitors collect air quality information every 2-3 days.

---

[4] https://data.nasa.gov/api/views/9ns5-uuif/rows.csv?accessType=DOWNLOAD

[5] https://chhs.data.ca.gov/api/views/q4et-a8rk/rows.csv?accessType=DOWNLOAD

[6] https://catalog.data.gov/dataset/air-quality-measures-on-the-national-environmental-health-tracking-network

The data is downloadable as a CSV file, and ranges from 12/2015 to 2/2017 and consists of state, county, and includes values from multiple categories:

1. Number of days with maximum ozone concentration exceeding standards set by NAAQS (National Ambient Air Quality Standard)
2. Average concentration of PM2.5 (2.5 micrograms per cubic meter, the cutoff for particulate matter) based on seasonal average
3. Percent of days in the last year exceeding NAAQS PM2.5 level standards

- Water Quality[7]

  The US Geological Survey monitors the National Water Quality Network (NWQN) which includes 113 rivers and streams in the US. Data is collected in two CSV files, the first containing the river/stream data and the second containing the water sites and county data. The former provides historical data from 1980-2015, and includes concentration of nitrates, nitrogen, phosphorus and suspended sediment.

Figure 1: Water Site and Water Quality Dataset

SITE_FLOW_ID	CONSTIT	DATE	WY	CONCENTRATION
11074000	NO3_NO2	1992-11-10	1993	5.8
11074000	NO3_NO2	1993-01-26	1993	1.2
11074000	NO3_NO2	1993-03-25	1993	3.6
11074000	NO3_NO2	1993-05-12	1993	4.1
11074000	NO3_NO2	1993-07-13	1993	8.4
11074000	NO3_NO2	1993-09-15	1993	6.8
11074000	NO3_NO2	1993-11-03	1994	7.8
11074000	NO3_NO2	1994-01-26	1994	6.9
11074000	NO3_NO2	1994-03-17	1994	6.0
11074000	NO3_NO2	1994-05-26	1994	4.5
11074000	NO3_NO2	1994-07-14	1994	5.9
11074000	NO3_NO2	1994-09-15	1994	6.8
11074000	NO3_NO2	1996-10-09	1997	8.0
11074000	NO3_NO2	1996-10-30	1997	8.9
11074000	NO3_NO2	1996-10-31	1997	8.4
11074000	NO3_NO2	1996-11-21	1997	7.3
11074000	NO3_NO2	1996-11-22	1997	3.7
11074000	NO3_NO2	1996-11-24	1997	3.1
11074000	NO3_NO2	1996-11-25	1997	2.74
11074000	NO3_NO2	1996-12-09	1997	5.2
11074000	NO3_NO2	1996-12-10	1997	3.6
11074000	NO3_NO2	1996-12-11	1997	3.1
11074000	NO3_NO2	1996-12-12	1997	3.31
11074000	NO3_NO2	1996-12-13	1997	3.11
11074000	NO3_NO2	1996-12-14	1997	2.87
11074000	NO3_NO2	1996-12-16	1997	3.61
11074000	NO3_NO2	1996-12-17	1997	3.1
11074000	NO3_NO2	1997-01-20	1997	3.7
11074000	NO3_NO2	1997-01-21	1997	3.15
11074000	NO3_NO2	1997-01-25	1997	4.5

[7] https://www.sciencebase.gov/catalog/item/57d1c97de4b0571647cfd3d3

1.	Station_name	SITE_QW_ID	SITE_FLOW_ID	Site_type	Drainage
2.	USGS National Water Quality Assessment Program, National Fixed Site Network				
3.	Wild River at Gilead, Maine	01054200	01054200	Reference	70
4.	Green River near Colrain, MA	01170100	01170100	Reference	41
5.	Connecticut River at Thompsonville, CT	01184000	01184000	Coastal rivers	9,660
6.	Norwalk River at Winnipauk, CT	01209710	01209700	Urban	33
7.	Canajoharie Creek near Canajoharie, NY	01349150	01349150	Agriculture	60
8.	Hudson River near Poughkeepsie, NY	01372043	01372058	Coastal rivers	11,700
9.	Neversink River near Claryville, NY	01435000	01435000	Reference	67
10.	Delaware River at Trenton, NJ	01463500	01463500	Coastal rivers	6,780
11.	McDonalds Branch in Byrne State Forest NJ	01466500	01466500	Reference	2
12.	Young Womans Creek near Renovo, PA	01545600	01545600	Reference	46
13.	Susquehanna River at Conowingo, MD	01578310	01578310	Coastal rivers	27,100
14.	Waites Run near Wardensville, WV	01610400	01610400	Reference	13
15.	Potomac River at Chain Bridge at Washington, DC	01646580	01646500	Coastal rivers	11,570
16.	Accotink Creek near Annandale, VA	01654000	01654000	Urban	24
17.	Swift Creek near Apex, NC	02087580	02087580	Urban	21
18.	Neuse River at Kinston, NC	02089500	02089500	Large inland rivers	2,692
19.	Contentnea Creek at Hookerton, NC	02091500	02091500	Agriculture	733
20.	Edisto River near Givhans, SC	02175000	02175000	Coastal rivers	2,730
21.	Altamaha River at Everett City, GA	02226160	02226000	Coastal rivers	14,000
22.	Sopchoppy River near Sopchoppy, FL	02327100	02327100	Reference	102
23.	Sope Creek near Marietta, GA	02335870	02335870	Urban	31
24.	Chattahoochee River near Whitesburg, GA	02338000	02338000	Large inland rivers	2,430
25.	Hillibahatchee Creek near Franklin, GA	02338523	02338523	Reference	17
26.	Apalachicola River near Sumatra, FL	02359170	02359170	Coastal rivers	19,200
27.	Alabama River at Claiborne AL	02429500	02428400	Large inland rivers	22,000
28.	Tombigbee River near Coffeeville, AL	02469762	02469762	Large inland rivers	18,417
29.	Ohio River at Cannelton Dam at Cannelton, IN*	03303280	03303280	Large inland rivers	97,000
30.	White River at Hazleton, IN5	03374100	03374000	Large inland rivers	11,305
31.	Wabash River at New Harmony, IN*	03378500	03377500	Large inland rivers	29,234
32.	Little River above Townsend, TN	03497300	03497300	Reference	106
33.	Tennessee River at Highway 60 near Paducah, KY*	03609750	Kentucky Dam outflow (Tennessee Valley Authority)	Large inland rivers	40,330
34.	Ohio River at Olmsted, IL	03612600	03612600	Large inland rivers	203,100
35.	Popple River near Fence, WI	04063700	04063700	Reference	139
36.	Clinton River at Sterling Heights, MI	04161820	04161820	Urban	309

- Earthquakes[8]

  The US Geological Survey updates the CSV files every 5 minutes on significant earthquakes, but requires manual downloading of the data. Earthquake parameters include time, location, magnitude, and depth.

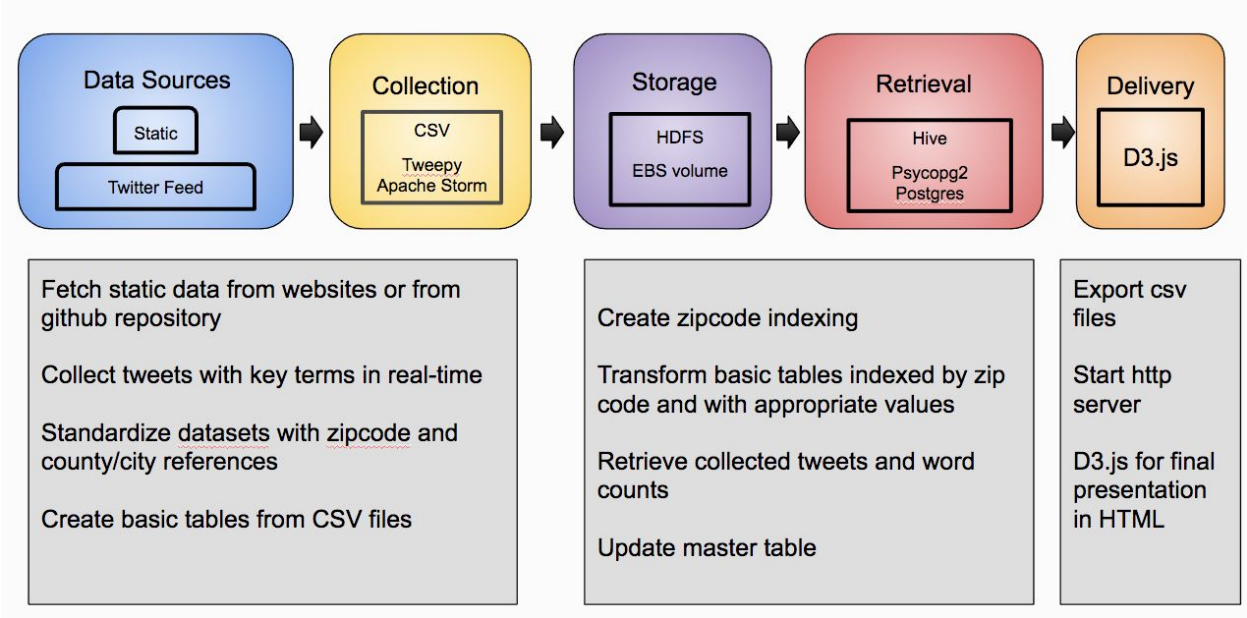
- Storms[9]

  Storm data is obtained from the NOAA (National Centers for Environmental Information). This is one of the datasets that required manually downloading from the website as we had to choose the state and duration (1/2013 - 1/2017), also event type (coastal flood, drought, excessive heat, flash flood, storm

[8] https://earthquake.usgs.gov/earthquakes/feed/v1.0/csv.php
[9] https://www.ncdc.noaa.gov/stormevents/choosedates.jsp?statefips=6%2CCALIFORNIA

surge/tide, tsunami, tornado, and tropical storm). Parameters that are recorded include indirect or direct deaths, injuries and damage, as well as type of event.

Section 3: Architecture Overview



Static data sources were fetched from the websites directly, or if the website required manual input, the files were downloaded into our github repository. The streaming data was sourced real-time from a Twitter API. Using Hive, basic tables were created from the static website and stored in HDFS located on an EBS volume. We then performed calculations and manipulations for these static datasets in Hive before exporting the final table into a CSV format to be used later in our serving layer.

The collected tweets, in two forms - word count and tweet/time of tweet pairs - were also saved as CSV files. A personal Twitter application was created and the Twitter API was linked to postgres via a psycopg2 connection. Using postgres, the word count and tweet/time pairs were created into tables which were later saved in the serving directory as a CSV.

The serving layer was created using d3.js and set up as a simple HTTP server in the EC2 instance using python. A user can access this serving layer by entering in IP

address of the instance and port 8080 into a browser. The user will then be able to interact with our dataset by entering in a California zip code of their choice.

Section 4: Data Collection and Processing

Hive Tables

In order for us to present the different threats from the data sources listed above to a user, we had to first design a database structure that would allow us to easily manipulate and query the data into a readable file that can then be used for our serving layer. We tried other methods including Spark-SQL but ran into a few issues during while transforming our datasets. Therefore, we chose to use Hive for its fault tolerance, even though Hive is slower than Spark in its processing time.

After loading the data sources into HDFS, we then created the individual table schemas in Hive. After that we created transform tables to clean up the datasets, filter out unnecessary records, and perform the appropriate calculations. These individual transformations set up the tables in such a way for us to be able to join all of our static datasets into one master table. We then export this master table that uses zip code as its primary key into a csv file that can be queried against quickly in the serving layer.

Some of our datasets, such as arrests and landslides, are reported by the county instead of by zip code. We wanted to have a quick response time in the serving layer so once the user inputs a zip code, it can instantly return a result. For this reason, we decided to replicate the county data across all zip codes in the county. This way, instead of querying against two tables and performing a calculation of finding out which county the zip code belongs in once the user inputs a zip code, we instead are querying against one table and perform zero calculations in our serving layer. This does create extra processing time in our transformation phase, but it saves time when the end user interacts with our product.

Sample Hive SQL Initial Table Schema:

```
DROP TABLE Zip_Code_Reference;

CREATE EXTERNAL TABLE Zip_Code_Reference
(
  Zipcode int,
  ZipCodeType string,
  Decommissioned string,
  PrimaryCity string,
  AcceptableCity string,
  UnacceptableCity string,
  State string,
  County string,
  TimeZone string,
  AreaCode string,
  WorldRegion string,
  Country string,
  Latitude int,
  Longitude int,
  IRSPopulation2014 int
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar" = '"',
  "escapeChar" = '\\'
)
STORED AS TEXTFILE
LOCATION '/user/w205/project/zip_reference'
;
```

Sample Transform Table SQL Command:

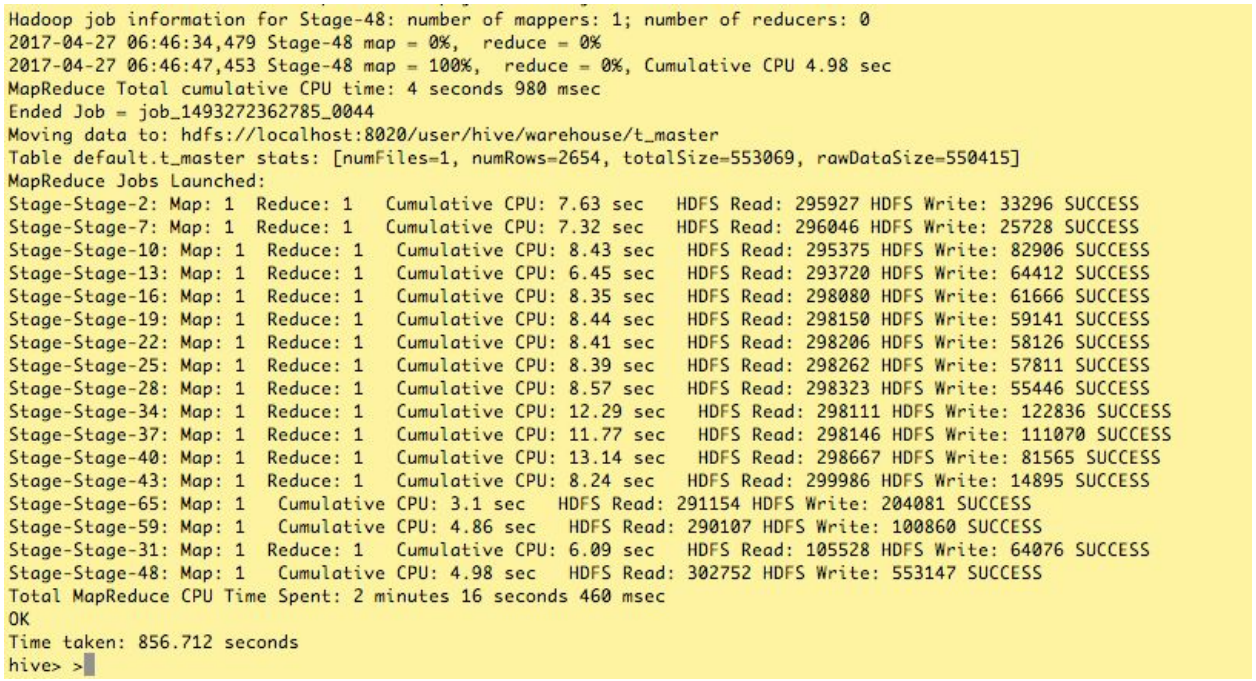
```
DROP TABLE t_landslide;
CREATE TABLE t_landslide AS SELECT
  county,
  count(id) as landslide_count,
  rank() OVER (
    ORDER BY count(id) DESC) as county_rank,
  max(cast (concat
            (substr(date_, 7, 4),
            '-',
            substr(date_, 1, 2),
            '-',
            substr(date_, 4, 2)
            )
       as date)) AS latest_date

FROM landslide_data
WHERE state = "California" AND county <> "obe"
GROUP BY county
SORT BY landslide_count DESC;
```

Part of Master Table SQL Command:

```
CREATE TABLE T_Master as
SELECT zip_code_reference_ca.zipcode,
   zip_code_reference_ca.primarycity,
   zip_code_reference_ca.county,
   obsolete_bridge.obsolete_bridge_count,
   deficient_bridge.deficient_bridge_count,
   earthquake.num_of_earthquakes_last_7_days,
   earthquake.most_recent_earthquake,
   earthquake.max_magnitude_last_7_days,
   fire.fire_last_7_days,
   death1.cause_of_death1,
   death1.num_of_deaths_of_cause1,
   death2.cause_of_death2,
   death2.num_of_deaths_of_cause2,
   death3.cause_of_death3,
   death3.num_of_deaths_of_cause3,
   death4.cause_of_death4,
   death4.num_of_deaths_of_cause4,
   death5.cause_of_death5,
   death5.num_of_deaths_of_cause5,
   water_quality.most_recent_water_quality,
   water_quality.avg_water_quality_concentration,
   aq_annual.aq_annual_year,
   aq_annual.aq_annual_value,
   aq_percent.aq_percent_year,
   aq_percent.aq_percent_value,
   aq_days.aq_days_year,
   aq_days.aq_days,
   storm.storm_deaths,
   storm.storm_injuries,
   storm.storm_damages,
   storm.storm_type,
   master_by_county.homicide_per_1000,
   master_by_county.violent_per_1000,
   master_by_county.property_per_1000,
   master_by_county.drug_per_1000,
   master_by_county.sex_per_1000,
   master_by_county.other_per_1000,
   master_by_county.landslide_count,
   master_by_county.landslide_rank,
   master_by_county.latest_landslide
FROM zip_code_reference_ca
LEFT JOIN (
   SELECT zip_code_reference_ca.zipcode,
      COUNT(*) as obsolete_bridge_count
   FROM zip_code_reference_ca LEFT JOIN t_bridge ON zip_code_reference_ca.zipcode=t_bridge.zipcode
   WHERE t_bridge.status = 'Functionally Obsolete'
   GROUP BY zip_code_reference_ca.zipcode
) obsolete_bridge ON obsolete_bridge.zipcode = zip_code_reference_ca.zipcode
LEFT JOIN (
   SELECT zip_code_reference_ca.zipcode,
      COUNT(*) as deficient_bridge_count
   FROM zip_code_reference_ca LEFT JOIN t_bridge ON zip_code_reference_ca.zipcode=t_bridge.zipcode
   WHERE t_bridge.status = 'Structurally Deficient'
   GROUP BY zip_code_reference_ca.zipcode
) deficient_bridge ON deficient_bridge.zipcode = zip_code_reference_ca.zipcode
LEFT JOIN (
   SELECT zip_code_reference_ca.zipcode,
      CASE
         WHEN MAX(t_earthquake.time) IS NULL
            THEN 0
            ELSE COUNT(*)
      END as num_of_earthquakes_last_7_days,
      MAX(t_earthquake.time) as most_recent_earthquake,
      MAX(t_earthquake.mag) as max_magnitude_last_7_days
   FROM zip_code_reference_ca LEFT JOIN t_earthquake ON zip_code_reference_ca.zipcode=t_earthquake.zipcode
   GROUP BY zip_code_reference_ca.zipcode
) earthquake ON earthquake.zipcode = zip_code_reference_ca.zipcode
```

Screenshot of Creating the Master Table in Hive:



```
Hadoop job information for Stage-48: number of mappers: 1; number of reducers: 0
2017-04-27 06:46:34,479 Stage-48 map = 0%,  reduce = 0%
2017-04-27 06:46:47,453 Stage-48 map = 100%,  reduce = 0%, Cumulative CPU 4.98 sec
MapReduce Total cumulative CPU time: 4 seconds 980 msec
Ended Job = job_1493272362785_0044
Moving data to: hdfs://localhost:8020/user/hive/warehouse/t_master
Table default.t_master stats: [numFiles=1, numRows=2654, totalSize=553069, rawDataSize=550415]
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 7.63 sec   HDFS Read: 295927 HDFS Write: 33296 SUCCESS
Stage-Stage-7: Map: 1  Reduce: 1   Cumulative CPU: 7.32 sec   HDFS Read: 296046 HDFS Write: 25728 SUCCESS
Stage-Stage-10: Map: 1  Reduce: 1   Cumulative CPU: 8.43 sec   HDFS Read: 295375 HDFS Write: 82906 SUCCESS
Stage-Stage-13: Map: 1  Reduce: 1   Cumulative CPU: 6.45 sec   HDFS Read: 293720 HDFS Write: 64412 SUCCESS
Stage-Stage-16: Map: 1  Reduce: 1   Cumulative CPU: 8.35 sec   HDFS Read: 298080 HDFS Write: 61666 SUCCESS
Stage-Stage-19: Map: 1  Reduce: 1   Cumulative CPU: 8.44 sec   HDFS Read: 298150 HDFS Write: 59141 SUCCESS
Stage-Stage-22: Map: 1  Reduce: 1   Cumulative CPU: 8.41 sec   HDFS Read: 298206 HDFS Write: 58126 SUCCESS
Stage-Stage-25: Map: 1  Reduce: 1   Cumulative CPU: 8.39 sec   HDFS Read: 298262 HDFS Write: 57811 SUCCESS
Stage-Stage-28: Map: 1  Reduce: 1   Cumulative CPU: 8.57 sec   HDFS Read: 298323 HDFS Write: 55446 SUCCESS
Stage-Stage-34: Map: 1  Reduce: 1   Cumulative CPU: 12.29 sec   HDFS Read: 298111 HDFS Write: 122836 SUCCESS
Stage-Stage-37: Map: 1  Reduce: 1   Cumulative CPU: 11.77 sec   HDFS Read: 298146 HDFS Write: 111070 SUCCESS
Stage-Stage-40: Map: 1  Reduce: 1   Cumulative CPU: 13.14 sec   HDFS Read: 298667 HDFS Write: 81565 SUCCESS
Stage-Stage-43: Map: 1  Reduce: 1   Cumulative CPU: 8.24 sec   HDFS Read: 299986 HDFS Write: 14895 SUCCESS
Stage-Stage-65: Map: 1   Cumulative CPU: 3.1 sec   HDFS Read: 291154 HDFS Write: 204081 SUCCESS
Stage-Stage-59: Map: 1   Cumulative CPU: 4.86 sec   HDFS Read: 290107 HDFS Write: 100860 SUCCESS
Stage-Stage-31: Map: 1  Reduce: 1   Cumulative CPU: 6.09 sec   HDFS Read: 105528 HDFS Write: 64076 SUCCESS
Stage-Stage-48: Map: 1   Cumulative CPU: 4.98 sec   HDFS Read: 302752 HDFS Write: 553147 SUCCESS
Total MapReduce CPU Time Spent: 2 minutes 16 seconds 460 msec
OK
Time taken: 856.712 seconds
hive> >
```

Twitter API

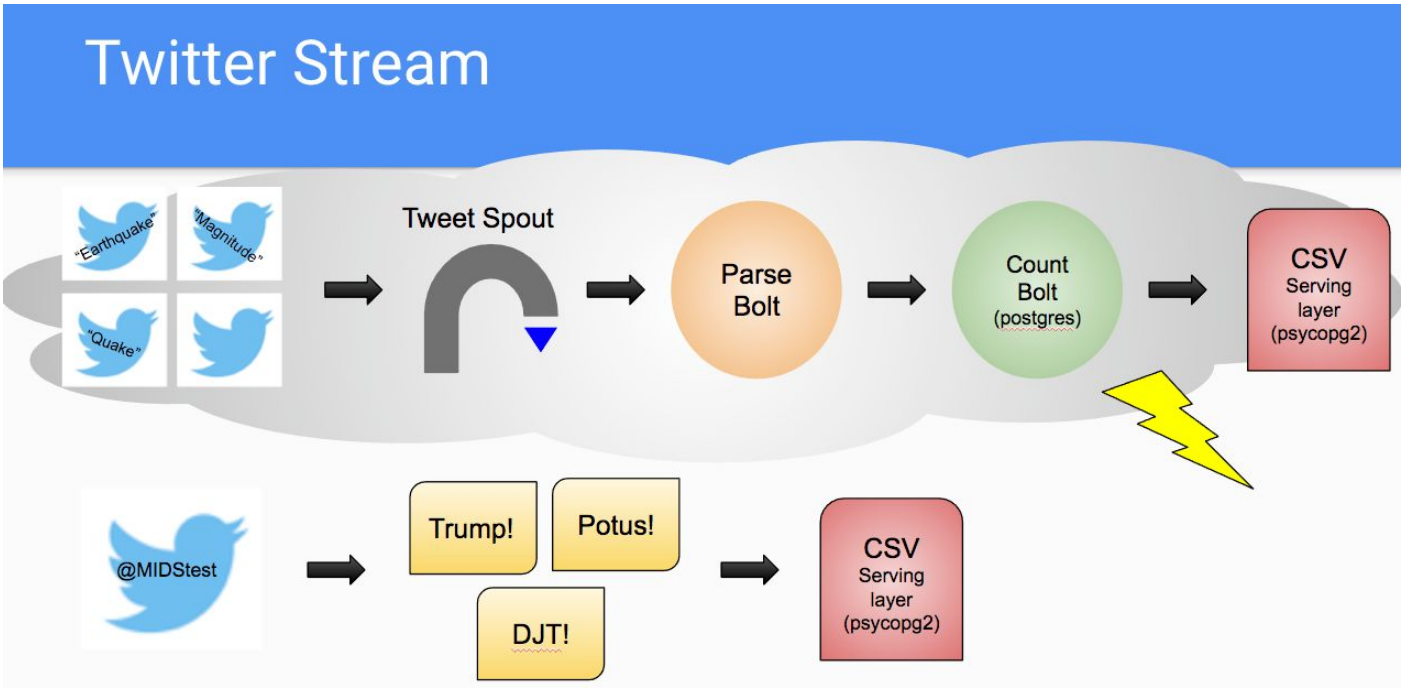
An important aspect of our Doomsday dashboard is that we wanted to show up to date information about certain types of threats. Because Twitter has played such a large role in political communications since the last election, we decided to collect information about the current President as the APA (American Psychological Association) has shown him to be one of the sources of stress in the American public. Alternatively, we wanted some type of weather or environmental related data to also be collected in real-time to reflect sudden dangers to the user. Initially, we searched for storm or severe weather related user handles, but the frequency of new tweets was low due to the fact that California does not have much in severe weather or storms. Certainly the drought was prevalent in the state, but that did not translate to a large volume of tweets on the subject. Instead, we chose to collect tweets from QuakesinCA which posts tweets when an earthquake occurs in California.

Storm

For the political data collection, the “extweepywordcount” directory from Exercise 2 was used as it already had the packages and libraries that were necessary to run the

program. For the same reason, we also used the “UCB MIDS W205 EX2-FULL” EC2 instance. A Storm application was initially created with 1 spout, 2 parse bolts and 2 word counter bolts. However, we ran into an issue with conflicting relational identification (ID) numbers for each row in the wordcount table. After discussion with our instructor, it was recommended to pare the bolts down to just 1 bolt each, and this solved the conflict. We believe that there were temporary tables that were created during the Storm process which were improperly shut down as one of the computers had crashed while running the program, and therefore had relational ID numbers which were the same.

Twitter API: Storm Architecture and get_tweet program



The tweet spout was configured to:

- Use our groups Twitter application access key, token, consumer key, and secret token.
- Track tweets that included the words: Trump, Potus, or DJT (which are the president’s initials)

Two bolts were created in the Storm DAG:

Bolt #1 - Parse

- Splitting the tweets into lower case words
- Creating a list of stop-words, which are approximately ~100 of the most commonly used words from Wikipedia, to filter out un-important words in tweets. We also included the tracked words in this list.
- Filtering out punctuation, links, and retweets

Bolt #1 Stop Words

```
def ascii_string(s):
  return all(ord(c) < 128 for c in s)

class ParseTweet(Bolt):

    def process(self, tup):
        tweet = tup.values[0]  # extract the tweet

        # Split the tweet into words
        words = tweet.split()

        # Change all words to lowercase
        words = [x.lower() for x in words]

        # Filter out the RT, @, stopwords and urls
        valid_words = []
        stop_words = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he",
                      "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they", "them", "their", "theirs",
                      "themselves", "what", "which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was", "were",
                      "be", "been", "being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an", "the", "and", "but",
                      "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about", "against", "between", "into",
                      "through", "during", "before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over",
                      "under", "again", "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both",
                      "each", "few", "more", "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so", "than", "too",
                      "very", "can", "will", "just", "dont", "should", "now", "want", "will", "rt", "trump", "trumps", "trump's", "donald"]


        for word in words:

            # Filter the user mentions
            if word.startswith("@"): continue

            # Filter out retweet tags
            if word.startswith("RT"): continue

            # Filter out the urls
            if word.startswith("http"): continue

            # Strip leading and lagging punctuations
            aword = word.strip("\"?><#,'.:;)")

            # Filter out commonly used words
            if word in stop_words: continue

            # now check if the word contains only ascii
            if len(aword) > 0 and ascii_string(word):
                valid_words.append([aword])
```

Bolt #2 - Word Count

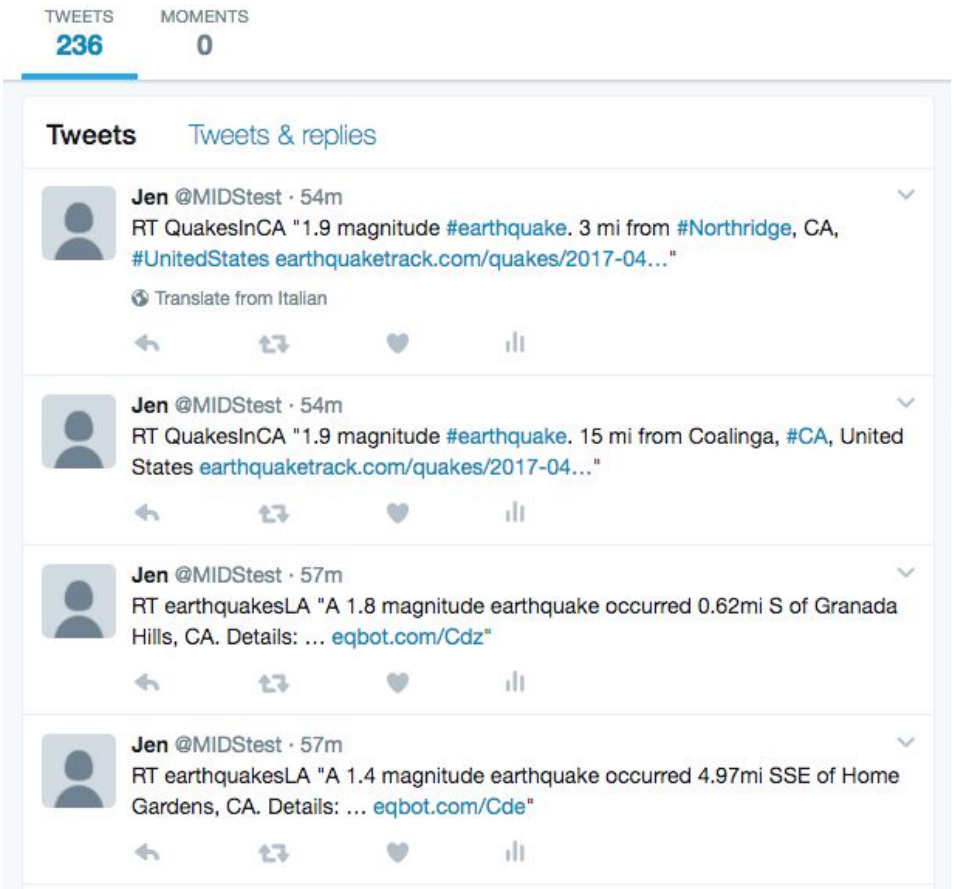
- Psycopg2 connection with postgres
- Creation of a database named “tweetcount”
- Creation of a table within tweetcount database named “wordcount”
- Iterated through the words parsed by the Parse bolt, then updated counts for each if exists, or inserted the word into the table if it was not present

A python program, “tweet_count.py”, initiated a psycopg2 connection with the tweetcount database in postgres, and copied the wordcount table to a CSV by using a sql query and writing the wordcount table to the file, under the serving folder in HDFS. The table was written to the CSV file with words that were with the highest count listed first to facilitate retrieval in the serving layer.

Get_tweets.py

We initially wanted for the second program, get_tweets.py, to follow specific user handles which posted earthquake-related tweets. However, we ran into an obstacle where private user handles were not able to be tracked. To rectify this, a new twitter account was made specifically to retweet the user handle (QuakesinCA) using an applet (ifttt.com).

MIDStest retweet bot



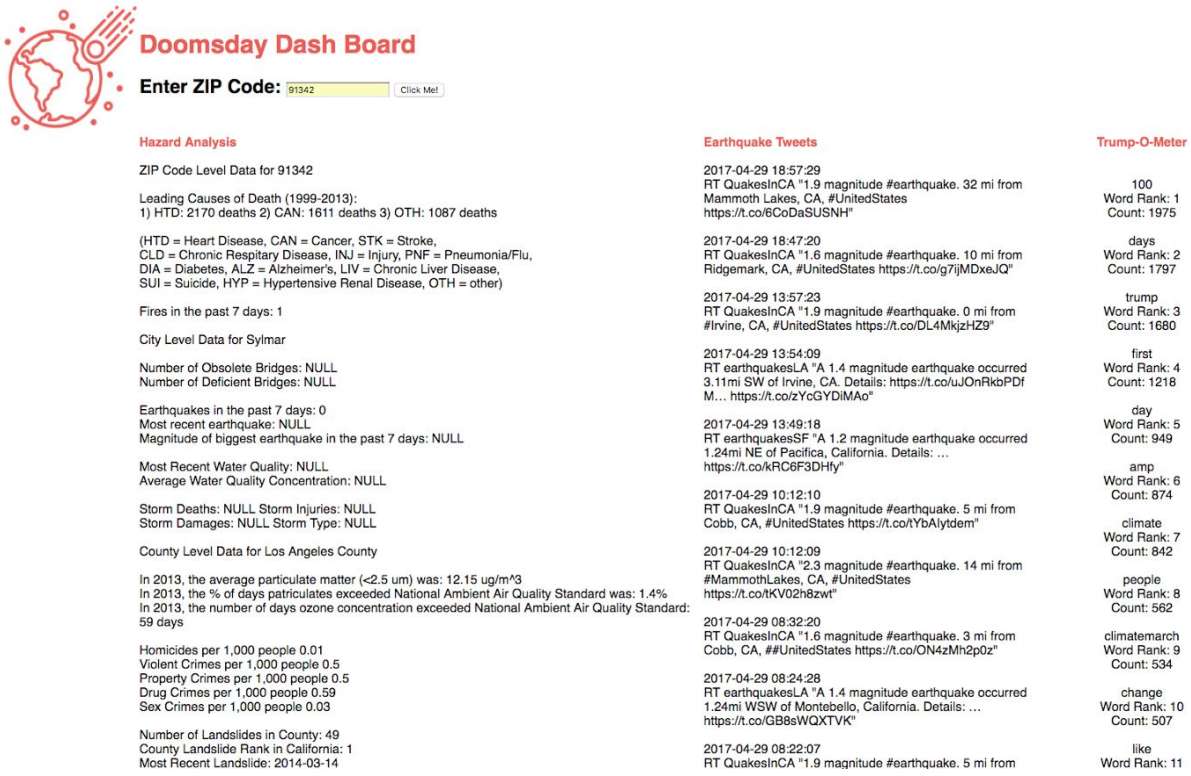
In this python program, another connection to the Twitter API was created using the same twitter credentials as that in Storm. The user name was specified to be "MIDStest" which was our retweet bot profile. Using the api.user_timeline function, we could collect the most recent 20 tweets from the user, and then convert it into a CSV file, again located in the serving layer. In each row of the CSV, there is information on when the post was posted, and the text of the tweet itself. We included time so that the user would be able to see approximately when the earthquake occurred.

Section 5: Results

The end result of this project allows the user to see interesting risks in zip codes of their choice. This dashboard gives a person access to information on different threats all in one easy to access place. We summarized the static datasets on the left side of the dashboard, the live earthquake tweets in the middle, and the top words or Trump-O-Meter on the right side.

For the static data, we broke down the data into zip code level, city level, and county level. Leading causes of death and the number of fires in the past 7 days are reported at a zip code detailed level. City level data include the number of unsafe bridges, earthquakes in the past 7 days, water quality tests, and recent storm damage. County level information are the air quality, number of felonies per 1,000 people, and number of landslides. The live earthquake tweets show the user up to the minute recent earthquakes in California. Hopefully, equipping people with this kind of information in an easily accessible manner will help people make preventative care decisions in their daily lives. We hope that one day, this could also be used to inform people of live imminent threats in their neighborhoods.

Screenshot of Dashboard:



Section 6: Future Direction

We successfully created a Doomsday Dashboard which allows the user to input his/her zip code, and then return updated information regarding fire, water quality, landslides, air quality, severe weather, arrest, bridge integrity, deaths, political tweets, and earthquake tweets. Looking forward, there are certain aspects that are scalable and can be improved upon.

First, the processing time is lengthy, likely owing to the use of Hive versus querying methods, such as SparkSQL. The data flow through our current architecture may be improved with more knowledge on methods to optimize SQL processing. This could allow us to use SparkSQL rather than Hive since we would encounter less issues with fault tolerance. We also designed this to be run from scratch. If we were to keep this product running, we would modify our structure to only update new data into our existing master table instead of the current method of creating the master table every time.

Second, the information gleaned from the twitter feeds can also be more meaningful to the user. Rather than solely relying on a word count, the words collected from tweets referencing the current president can be placed into a corpus, for a “bag of words” to employ text analysis through natural language processing. Using logistic regression, naive bayes, or decision trees, we could use the predictive labels to warn the user that a high probability of danger exists due to the impact of Twitter. For instance, a tweet from the current President could lead to a sharp decline in Dow Jones, a new travel ban targeting specific countries, or nuclear war with North Korea. Although it is unlikely that we would be able to provide any substantial prediction on any one of these incidents, using models to predict general threats may be possible.

Also, the get_tweets.py program can be improved by collecting from multiple users involved in earthquake tweets as a collective. By doing so, we may even be able to find a user that tweets from the location of the earthquake, so that we could use the latitude/longitude attributes of the stream to notify the user of earthquakes specifically in their zip code. Also, there is a chance that a single user cannot capture all earthquakes, so following more users would provide a more complete picture of the earthquake trend. Another approach would be to parse the tweets for the location of the earthquake. This seems easy, however the place names used by the USGS as reported by @QuakesinCA account are not always consistent and entity resolution to a ZIP code was a challenge.

We also wanted the twitter stream to be automated as a timed recurring collection rather than a user-initiated process. By doing so, many more tweets would be collected and provide a richer dataset. This would allow for more meaningful analysis and illustration of trends to allow for a better threat analysis.

Another significant limitation is in our serving layer. Running the simple HTTP server using python is a convenient and elegant solution when serving to a handful of users. A key part of any future work would be to implement a more robust and scalable solution such as setting up an Apache Web Server.

In our initial proposal, we wanted to create an actual clock to deliver to the user as an end result. However, we found it very difficult to decide which tables should be considered high priority hazards versus those which present less of a danger. How all the inputs can be combined to create a certain “time”on the clock is an open ended and ultimately subjective process. If we had expert input on the various fields from which we collected the data, we could have been able to specify reasonable thresholds that, when exceeded, could change the minute hand closer or away from midnight. In the absence

of such an expert, it was considered more factual, less subjective, and potentially more useful to report the different hazards separately as a dashboard.

In the end, we felt that we were successful in creating a Doomsday Dashboard that enlightened the user on various threats in their area from both predefined and real-time data. The result of our project was easy to understand and informative, which were our primary goals. However, like all baseline models, there are several directions that we could adopt to scale up our dashboard. Certainly this project allowed us better understand our own strengths and limitations and provided many thoughtful lessons on data storage, processing, and delivery.