

Impact estimation of earthquakes with a twitter analysis

Final report

School: Lucerne University of Applied Sciences and Arts

Program: Master of Science in Applied Information and Data Science (MScIDS)

Course: Data Warehouse and Data Lake Systems 2

Team: Earthquake

Place/Date: Lucerne, 16.01.2022

Supervisors: José Mancera and Luis Teràn

Authors:

Sandro Huber
Spitalackerstrasse 20a
3013 Bern
+4178 840 09 09
sandro.huber@stud.hslu.ch

Thomas Schwendimann
Spitalstrasse 8b
6210 Sursee
+41 79 129 19 43
thomas.schwendimann@stud.hslu.ch

Lea Senn
Treppenweg 19
5300 Turgi
+4179 960 74 79
lea.senn@stud.hslu.ch



Table of Contents

1	Background.....	1
1.1	Business questions.....	1
2	Review of the first part.....	1
4	Procedure – Data lake	2
4.1	Twitter data	2
4.2	Earthquake data.....	4
4.3	Data lake storage overview	6
5	Procedure – data warehouse	8
5.1	Twitter data	8
5.2	Cleaning earthquakes table.....	10
5.3	Preparation of the twitter_count table for the merge	13
5.4	Final data warehouse table	13
6	Visualization	15
6.1	Individual graphics	16
6.2	Dashboard filters.....	18
7	Conclusions.....	20
7.1	Research Questions.....	20
8	Sources.....	22
8.1	Images	22
8.2	Project results.....	Error! Bookmark not defined.

List of Figures

Figure 1 - DAG get_twitter.py.....	3
Figure 2 - Abstract table tweets	4
Figure 3 - Abstract table tweets_user.....	4
Figure 4 - Access API	4
Figure 5 – security access rules	5
Figure 6 - Connection via jupyter notebook.....	5
Figure 7 – DAG earthquake_daily.py	6
Figure 8 - Creation of datalake table for earthquakes	6
Figure 9 - Datalake	6
Figure 10 – planned data structure	7
Figure 11 - Architecture	8
Figure 12 – Dag to clean twitter data	8
Figure 13 - Abstract data warehouse dw_twitter.....	9
Figure 14 – Unprepared earthquake table.....	10
Figure 15 – Cleaned earthquake table.....	10
Figure 16 – Time column in milliseconds	11
Figure 17 – New Date and Exact time columns.....	12
Figure 18 – Creation of new columns	12
Figure 19 – Preparation of twitter table	13
Figure 20 – Load data into data warehouse.....	13
Figure 21 – Final data warehouse table	14
Figure 22 – Final dashboard	15
Figure 23 – World map.....	16
Figure 24 – Hovering effects.....	16
Figure 25 – Detailed earthquakes diagram	17
Figure 26 – Detailed chart about number of tweets	17
Figure 27 – Filtering options.....	18
Figure 28 – Additional highlighting / filtering options	18
Figure 29 - Additional highlighting / filtering options.....	19
Figure 30 – Word cloud of the tweets after Japan earthquake in March 2011	21

1 Background

As a global reinsurance company, our client SwissRe is affected by global catastrophes on a broad scale. In 2020, SwissRe estimated the global insured catastrophe losses to be USD 83 billion (SwissRe, 2020). A large amount of these costs are caused by earthquakes. To be aware what costs are going to be expected, it is important for SwissRe to get as much information about the damages as early as possible. This project attempts to estimate as accurately as possible the coming cost of an earthquake based on a twitter analysis.

1.1 Business questions

We assume that bigger damages also lead to a bigger response in social media. To estimate the damage, we want to try to analyse the "twitter echo" after big earthquakes and compare the amount of tweets with the numbers from comparable earthquakes in the past.

We plan to make this comparison available to SwissRe's claims experts on a dashboard in an intuitive way. The claims experts from SwissRe can then use this information and match it with their historical cost data. They might find a relationship between the number of tweets and the costs and can use the information to forecast future costs for earthquakes.

We decided to answer the following research questions:

- 1 *At what magnitude do earthquakes trigger a reaction on twitter?*
- 2 *Is there a dependency between the strength of the earthquake and the number of tweets sent?*
- 3 *How did the sentiment in the tweets change based on the magnitude?*

2 Review of the first part

In the first half of the semester, the team developed the business case and the business question. A comprehensive analysis was carried out using the cross-industry standard procedure for data mining (CRISP-DM). Through insights gained by consulting "state of the art" projects, a clear project plan was established to answer the business questions.

The required data sources were analyzed and extensively tested to ensure at an early stage that their quality was sufficient to answer the research question.

Details of all these steps, the used data sources and the first attempts to load the data into the data lake can be found in the project documentation of the first half of the semester.

4 Procedure – Data lake

4.1 Twitter data

In a first step, the twitter API is used to access the historical data for as far back as possible (max. 10 years) with the limit of 10 million tweets a month. Python's library 'requests' is used for all GET requests to the API. The code is executed in a Directed Acyclic Graph (DAG) from Apache Airflow. This chapter provides an overview of the steps involved in extracting the data and storing it in a postgresSQL database – the data lake – using a DAG.

4.1.1 Preparation

Due to the monthly limit of tweets that can be accessed (10 mio. tweets per month), it was not possible to create a query which returns every tweet containing the word 'earthquake'. A single day already returns roughly over 10'000 tweets, which would limit the time frame that is accessible for this project. In order to increase this time frame, test queries have been executed to optimize the final search criteria. For example, retweets, replies and the word 'minor' are excluded from the search.

Also, a lot of tweets are created by bots which do not deliver insights to our research. Those tweets usually contain no relevant information, such as earthquakes with a low value on the Richter scale. Therefore, the number of tweets a single account posts a day were counted for a short time period. In order to further reduce the number of results we obtain, the accounts with the highest number of tweets were extracted from the data and excluded from the query. Since the length of the query is limited to 1024 characters, only around 40 usernames have been excluded. Once the data is in the data lake, the other bots will be removed when transforming the data before it is loaded into the data warehouse.

4.1.2 Setting up the Data lake

For the data storage, a postgresSQL database has been created in Amazon's Relational Database Service (RDS). This database serves as a data lake, where all the twitter data will be stored in its raw format. This includes on one hand the tweets matching the criteria, but also information about the users which posted them. These data will be stored in separate tables.

4.1.3 Extracting the data

Figure 1 represents the DAG for the process of extracting and loading the twitter data into the data lake. Due to the access limitation of the API, such as the number of allowed requests every 15 minutes, this DAG is executed every six hours and will extract half a year's worth of data. The defined time interval is 1 January 2010 to 31 October 2021.

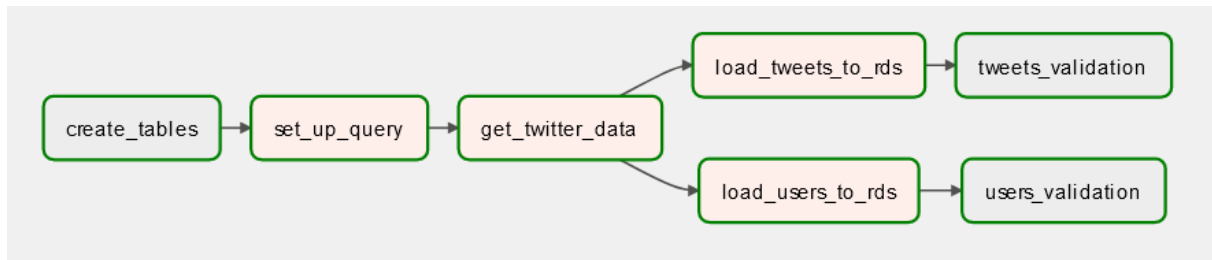


Figure 1 - DAG `get_twitter.py`

The first task checks if the corresponding tables ('tweets' and 'users') in the data lake exist, and if not creates them. The tables are therefore ideally only created with the first execution. Next up are the query parameters, which are defined in a separate task. Arguably, this step could be combined with the extracting of the data, but was chosen to be separate for a better overview. Two variables were added to the Apache Airflow environment, allowing access to them without writing the values as code. First, the bearer token required for the authentication when making the API requests. Second, the filter for the query, which consists of the prefix '-from:' and the usernames that are excluded from the search. These variables are accessed and used in the setup of the query. The query parameters, URL and headers required for the request are saved using XComs (cross-communications), a mechanism that allows sending data between tasks.

The next step pulls the XComs with the relevant variables and accesses the API and requests the twitter data matching the query. Only 500 tweets can be accessed per request from the API. With every request the API returns a token in the metadata – assuming more results are there but were not returned due to the limit. Using this token, one can then request again and will get the data beginning from where the previous request left off. This is done in an infinite while loop, which stops if there is no token in the metadata, indicating that all the results have been delivered. For the tweets, the fields 'created_at', 'text', 'author_id' and 'id' are returned as per query defined. For the users, the fields 'name', 'username', 'location' (if available) and 'id' (which matches the author id). All the responses are stored in lists containing a dictionary (json format) for each tweet and user respectively – and then once again saved using XComs.

4.1.4 Loading the data

The loading of the tweets and user information into the database is done in two separate tasks. In both cases, an iterator is created from the list containing the data – in order to save memory. Since not all user entries contain the key 'location', the key and an empty string as value is added in its place if not in the dictionary. This allows for easy insertion into the database. To connect to the database, a PostgresHook accesses the connection details, saved in Apache Airflow – once again to not have sensitive data visible in the code. Due to the volume of data, the `execute_batch` function from `psycopg2`'s library is used to insert the rows into the tables. This requires a separate import `psycopg2.extras`. If the data volume were even higher, the `copy_from` method would have been used, which has the fastest execution time when combined with an iterator (Benita, 2019). Finally, once the data has been loaded, a simple SELECT request is done for each table to validate if the process has been successful or not. In the tables below is the output of a simple SELECT statement using `pgAdmin4`, a software with a GUI for PostgreSQL databases.

	text text	author_id bigint	id bigint	created_at text
1	#Adele #COVID19 #DOYOUNG #earth...	1457150492896030720	1460228489777270793	2021-11-15T12:49:36.000Z
2	地震情報を受信しました	1187648090469064709	1460228425663266817	2021-11-15T12:49:21.000Z
3	4.4 magnitude #earthquake. 72 km from...	362523555	1460228354578153474	2021-11-15T12:49:04.000Z
4	يمكنك التواصل مع السائل من خلال التطبيق	1293809522402897926	1460228324677062656	2021-11-15T12:48:56.000Z
5	#Earthquake of magnitude 4.4 at 71 km ...	1291969748599574528	1460228263607943173	2021-11-15T12:48:42.000Z
6	Random #DoctorWho thought: I'd love t...	1028393985352642561	1460228244398039040	2021-11-15T12:48:37.000Z
7	Mwr 4.4 earthquake (reviewed) occured ...	107816182	1460228225414701056	2021-11-15T12:48:33.000Z

Figure 2 - Abstract table tweets

	id bigint	username text	name text	location text
1	1457150492896030720	_akumaemoji	🐼🐼🐼	
2	1187648090469064709	nexryai	NEXYAI	
3	362523555	QuakesToday	Earthquake Alerts	
4	1293809522402897926	Ahtaajmohamoon	أحتاج محامي	
5	1291969748599574528	earthquake247	Earthquake Live	
6	1028393985352642561	isoskramer	Isosceles Kramer: Cosmic Hobo	St. Cedd's College, Cambridge
7	107816182	earthshook	Quake Reports	

Figure 3 - Abstract table tweets_user

4.2 Earthquake data

The earthquake data was retrieved from the earthquake.usgs.gov API. The API is provided by the United States Geological Survey (USGS) and can directly be accessed over the URL without any tokens or other verifications needed. The data can already be filtered by setting the GET parameters in the URL of the API. To save computational power and space on our web server, this filtering method was used to pre-filter all earthquakes that are not relevant for the project. This was done with the definition of the following parameters:

- Starttime
- Endtime
- Minimum magnitude

```

starttime = '2011-01-01' # start with data from sep 2021
endtime = '2021-10-31'
minimum_magnitude = '5'
url = 'https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime='+starttime+'&endtime='+endtime+'&minmag='+minimum_magnitude

r = requests.get(url) # request the API

r_json = json.loads(r.text) # turn the result into a json and extract all "Features"
df = json_normalize(r_json) # turn it into a pandas dataframe

len(df) # 19'288 earthquakes found!

```

19288

Figure 4 - Access API

The time frame of the observation is between 1 January 2011 until the 31 October 2021. These dates were chosen to limit ourselves to a certain time frame that is possible to cover with the twitter data. Additionally, the minimum magnitude of five has been defined. With help of this first filtering of the data, the API delivers almost 20'000 relevant earthquakes.

4.2.1 Data storage

To have a uniform data storage, the earthquake data is stored in a PostgreSQL database, similar to the twitter data. For this purpose, the database first needed to be created in the Amazon RDS. When it comes to safety aspects in terms of limited access to the database from specific IP-addresses, our group could not follow the best practice of a limited access. It was necessary to leave the access open to all IP-addresses since no one of our group member has a static IP-address. Therefore, this option was not possible for our team. However, the data is not sensitive, therefore, safety is not a big issue in this case. Nevertheless, the database is secured with help of a password, which is sufficient for this project.

Sicherheitsgruppe	Typ	Regel
default (sg-00000000)	CIDR/IP - Inbound	0.0.0.0/0
default (sg-0d637cf3186fb768)	CIDR/IP - Outbound	0.0.0.0/0

Figure 5 – security access rules

4.2.2 Data loading

This step was first tried to be accomplished with help of a small sample dataset with help of a Jupyter Notebook. This was done with help of the following steps:

- Downloading the data via USGS API
- Small data preparation step
 - Unfolding of the geodata points
- Connecting to the database with the according credentials

```

endpoint = ''
dbname = ''
dbuser = ''
password = ''

try:
    conn = psycopg2.connect("host="+endpoint+" dbname="+dbname+" user="+dbuser+" password="+password)
except psycopg2.Error as e:
    print("Error: Could not make connection to the Postgres database")
    print(e)

```

Figure 6 - Connection via jupyter notebook

- Creation of a table in the PostgreSQL database with the following code snippet
- Implementation of the data
- Close the connection again

After the successful implementation of the data into the PostgreSQL server with help of this little local prototype, the implementation was automated with help of Apache Airflow.



Figure 7 – DAG earthquake_daily.py

```

sql="""
CREATE TABLE IF NOT EXISTS earthquakes_table (
  type varchar(255), id varchar(255), properties.mag real, properties.place varchar(255), properties.time real,
  properties.updated real, properties.tz varchar(255), properties.url varchar(255),
  properties.detail varchar(255), properties.felt real, properties.cdi varchar(255),
  properties.mmi varchar(255), properties.alert varchar(255), properties.status varchar(255),
  properties.tsunami varchar(255), properties.sig varchar(255), properties.net varchar(255),
  properties.code varchar(255), properties.ids varchar(255), properties.sources varchar(255),
  properties.types varchar(255), properties.nst varchar(255), properties.dmin real,
  properties.rms real, properties.gap real, properties.magType varchar(255),
  properties.type varchar(255), properties.title varchar(255), geometry.type varchar(255),
  geometry.coordinates varchar(255))
  
```

Figure 8 - Creation of datalake table for earthquakes

4.3 Data lake storage overview

Finally, the created data lake includes the following information:

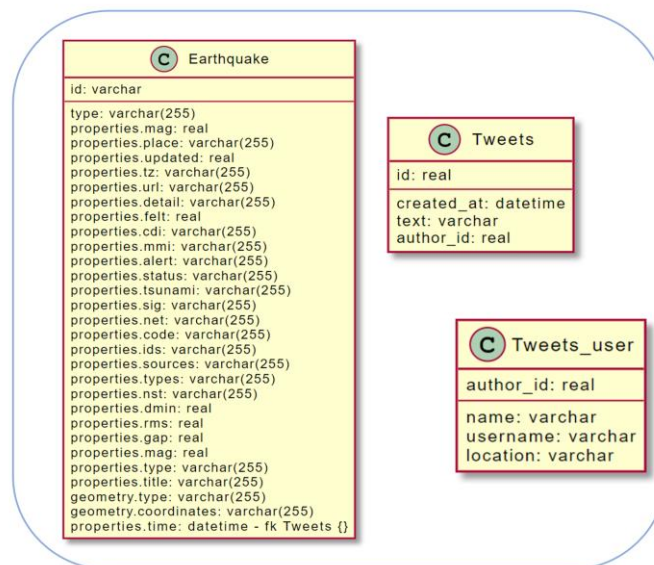


Figure 9 - Datalake

After the first semester half, the data was floating in the database in loose tables. Later, the data was prepared and rearranged to combine these raw data tables. Therefore, the following relationship model has been drawn:

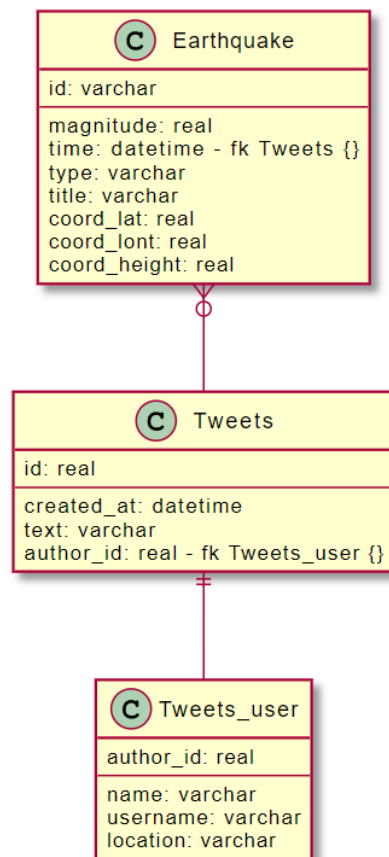


Figure 10 – planned data structure

Assigning a tweet to a certain earthquake can be quite challenging. Our first idea was to assign it based on the location. But we soon noticed that hardly any tweets contain geographic location. Therefore, for this project we worked under the assumption that a tweet refers to the biggest earthquake from the past 24h. This is, of course, not the reality. In a future attempt, it could be helpful to try and extract information about the location of earthquakes from the tweeted text.

Based on this assumption, the tweets can be associated with the earthquakes according to the “time” of the earthquake and the “created_at” of the tweets. An earthquake can have zero or multiple tweets. Additionally, a twitter user can directly be assigned to a tweet with help of the “author_id”. One tweet must have exactly one user who created the tweet.

5 Procedure – data warehouse

As stated in previous chapters, the extracted data was loaded into the data lake in a mostly raw format, with only slight changes. The data lake serves as the basis for all business questions, as it contains all information available. For our project, only some of this data is useful. As such, we used Apache Airflow to clean and filter the data and load it into the data warehouse (Figure 11).

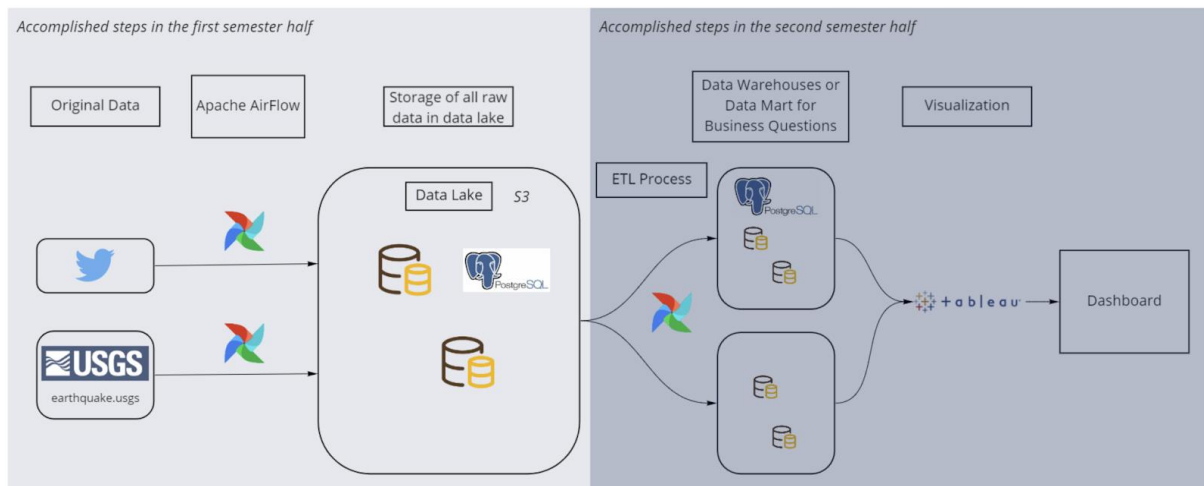


Figure 11 - Architecture

5.1 Twitter data

Cleaning data is usually an explorative process. Looking at the data reveals insights of what needs to be changed, adjusted or dropped altogether. Once changes have been made, exploring other aspects of the data reveals even more. However, since the cleaning and loading of the data into the warehouse was to be done in a one step process – a DAG using Apache Airflow – a Jupyter notebook was used to explore the data and find what changes need to be done. The resulting transformations were then executed in the respective DAG. In both cases, sensitive data, such as credentials and tokens for accessing APIs, has been hidden from the code. In Apache Airflow, the built-in function was used to store variables. In Jupyter notebook, the dotenv library enabled to store the variables inside an .env file, which is ignored by the git repository (.gitignore). Figure 12 represents the individual steps of the DAG.

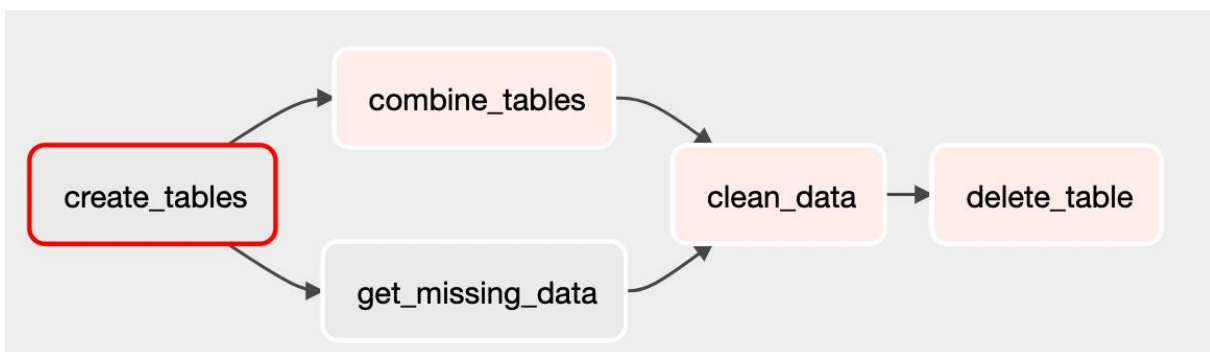


Figure 12 – DAG clean_datalake.py

Just as for the data lake, the tables for the data warehouse are created in the first step if they do not already exist. This step could potentially be omitted, since the way the data is later loaded into the

data warehouse does not require existing tables. They are instead created dynamically if they do not already exist. In this case, a temporary table is created additionally to the regular data warehouse table to store the data between DAG tasks.

In parallel, the two tables containing information of the users and tweets are combined into one, while at the same time missing data is extracted from the twitter API. To clarify, some user data was not loaded into the data lake in the first step due to invalid characters which the PostgreSQL database could not handle. As a result, if one username contained this character, all the usernames remaining in the iterator were not loaded into the table. At the time, this was ignored since there was a save way to find the periods with the usernames missing and later extract them again. However, as the project progressed it was apparent that it would have been more efficient to simply continue the loading process of the data once the error message had been raised. Since an iterator is used, the username containing the invalid character would have no longer been in said iterator, and the rest could have been inserted safely. This would have ensured that only a few usernames would have been missing, which could have been ignored and the time-consuming step of extracting the data would have not been necessary. Ultimately, the user data did not serve a purpose for the project and would not have been needed in the first place. However, in a real-life scenario the data could still be valuable for different analysis and should at least be stored in the data lake.

As a final step of the cleaning process, the bots which consistently tweeted about every earthquake were removed from the data. To identify these bots, two rules were defined and applied. First, every user containing the string sequence 'quake' or 'bot' were dropped. Second, all users that tweeted about earthquakes more than a hundred times were dropped.

Finally, the data was loaded into the data warehouse and the temporary table was dropped. The whole process reduced the number of tweets from around 13.5 million to 6 million.

	tweet_id bigint	created_at text	text text	author_id bigint	username text	name text
1	947617502200295425	2017-12-31 23:56:33	A second flood, a simple famine, plagues of locusts everywhere. Or a c...	946816509862703105	Charles50825232	Charles Goodson
2	947617386194018304	2017-12-31 23:56:05	A big earthquake with the strength of 8.1 on the Richter Scale has hit P...	2335781905	Robinhssan	Robin
3	947616574340521984	2017-12-31 23:52:51	#2017Faves RT @playbykay: Dallas had an earthquake and Houston is ...	33885120	Damey07	WMN THE BLOWH
4	947616006867787776	2017-12-31 23:50:36	QUEEN SLAY ME TONIGHT MAKE THE WORLD SHAKE WITH YOUR BEL...	552069606	loverboyale	Alex
5	947615370227208192	2017-12-31 23:48:04	I can hear Prince say earthquake #earthquakewine @ Michael David Wi...	440321768	blessedpoetpat	Patricia A. Saunders
6	947614973265559552	2017-12-31 23:46:30	正月早々「揺れた」という感覚はイヤだね...	284972313	hiroichi3	ひろいち(元JGSDF)
7	947613925306232832	2017-12-31 23:42:20	Irwin Allen night on TCM needs to be a thing. #ThePoseidonAdventure, ...	1517386392	TeamRickandIlsa	Elise
8	947613476121382912	2017-12-31 23:40:33	It was like an earthquake in the press box when Decker dropped that p...	345798491	mike_e_kaye	Mike Kaye
9	947613459952349185	2017-12-31 23:40:29	2017, in 7 minutes https://t.co/U25RAV3crK according to @voxdotcom ...	34033882	AkumalBay	Gerardo H. Dominguez
10	947613259204583424	2017-12-31 23:39:41	It was much cooler in person, the bass was on high it felt like an earthq...	422977413	Chopythes	~ Nelly ~ (ʘωʘ)

Figure 13 - Abstract data warehouse dw_twitter

5.1.1 Counting tweets

During the later stage of the project, it was apparent that using the extracted data to create the dashboard in Tableau would be difficult due to the long loading process. As a result, Twitter API was used to simply count the number of tweets matching the defined criteria (DAG get_twitter_count.py). This resulted in one single data point per day, which significantly reduced the data load. Given more time and more computational power, the analysis of the actual tweets could provide better insights, as there is no way to exclude bots from a simple count of tweets.

5.2 Cleaning earthquakes table

This chapter describes how the earthquake data was transformed into a data warehouse. The data cleaning was not only done in the second part of the project but started already with the data loading into the data lake. What happened in which parts will be described next.

5.2.1 Cleaning during the first stage (loading of the data in the data lake)

As mentioned above, the earthquake data from the USGS website consists of the following table columns.

 Earthquake
id: varchar
type: varchar(255) properties.mag: real properties.place: varchar(255) properties.updated: real properties.tz: varchar(255) properties.url: varchar(255) properties.detail: varchar(255) properties.felt: real properties.cdi: varchar(255) properties.mmi: varchar(255) properties.alert: varchar(255) properties.status: varchar(255) properties.tsunami: varchar(255) properties.sig: varchar(255) properties.net: varchar(255) properties.code: varchar(255) properties.ids: varchar(255) properties.sources: varchar(255) properties.types: varchar(255) properties.nst: varchar(255) properties.dmin: real properties.rms: real properties.gap: real properties.mag: real properties.type: varchar(255) properties.title: varchar(255) geometry.type: varchar(255) geometry.coordinates: varchar(255) properties.time: datetime - fk Tweets {}

Figure 14 – Unprepared earthquake table

Since our group has done a precise planning of how the data should be visualized and what data is needed to realize it, we were able to filter out some information from the table from the beginning. Due to this, computational power, time and space on the database could be saved. Additionally, the structure within the data lake was clearer, since only a reduced data set was loaded in it. The relevant columns that were loaded into the data lake were the following.


 Earthquake
id: varchar
magnitude: real time: datetime - fk Tweets {} type: varchar title: varchar coord_lat: real coord_lont: real coord_height: real

Figure 15 – Cleaned earthquake table

Therefore, at this stage, a big part of data cleaning has already been done. This step was realised with the first DAG of loading the data lake. Used DAG: “earthquake_daily.py”

5.2.2 Cleaning during the second stage (from data lake into data warehouse)

The data did not need much cleaning, since it came already well structured and complete from the API. There were for example neither duplicate entries present nor NaN values. However, there was still an additional task to fulfil. To combine the Twitter and earthquake datasets, it was necessary to create a new column that could be used to make the join. It was defined to use the time of the occurrence of the earthquake and the tweets. The earthquake data has stored its time information in terms of milliseconds according to the following illustration:

Time
1'293'752'823'930
1'293'744'150'350
1'293'738'996'380
1'293'736'886'463
1'293'728'174'930
1'293'719'814'560
1'293'710'561'800

Figure 16 – Time column in milliseconds

Since the time of the tweet is stored in another format, it was decided to transform this column into two new columns. First, it was aimed to transform it into a datetime format. Therefore, the exact date as well as the hours, minutes and seconds should be shown. Second, a new column should be created with only the date. This is the column on which the merge of the two tables should happen. This was defined, because a day is an ideal timeframe to aggregate the earthquakes and the twitter data. The target columns can be seen in the following illustration.

Date	Exact time
30.12.2010	30.12.2010 23:47:03
30.12.2010	30.12.2010 21:22:30
30.12.2010	30.12.2010 19:56:36
30.12.2010	30.12.2010 19:21:26
30.12.2010	30.12.2010 16:56:14
30.12.2010	30.12.2010 14:36:54
30.12.2010	30.12.2010 12:02:41

Figure 17 – New Date and Exact time columns

For the creation of these columns, the data was loaded from the data lake and was transformed within another DAG and then loaded into the final data warehouse. Additionally, the original column with the milliseconds was dropped, since this would have been redundant information.

This process was done with the following code:

```
#create new column and change to date typ
earthquakes_df["date"] = earthquakes_df["time"]
# Change datatype from milliseconds to datetime (column "time") and from milliseconds to date (column "Date")
count = 0
for i in earthquakes_df.iloc[:, 8]:
    earthquakes_df.iloc[count, 8] = datetime.datetime.fromtimestamp(earthquakes_df.iloc[count, 8] / 1000.0)
    count += 1
earthquakes_df["time"] = pd.to_datetime(earthquakes_df["date"])
earthquakes_df["date"] = pd.to_datetime(earthquakes_df["date"]).dt.date
```

Figure 18 – Creation of new columns

Within the “time” column, the exact time is listed and in the “date” column, only the date. The “date” column was used for the merge.

5.3 Preparation of the twitter_count table for the merge

For the data warehouse, the `twitter_count` table was needed as a completion of the earthquake data. The table holds the information about how many tweets were sent per day. The time frame is defined by a start- and endtime of the day. This is fine but it does not fit the requirement for the merge with the earthquake dataset. Only one column with the date is needed. Therefore, the starttime was transformed into the same datatype as in the earthquake table. This was done within the same DAG as the earthquake “date” column was created.

```
##Access twitter data and create df
twitter_df = db_hook.get_pandas_df("SELECT * FROM tweetcount")
#Adjust twitter table
#change type of "starttime"
twitter_df["starttime"] = pd.to_datetime(twitter_df["starttime"]).dt.date
# Rename column "starttime"
twitter_df.rename(columns={"starttime": "date"}, inplace=True)
# Drop Endtime
twitter_df.drop("endtime", axis=1, inplace=True)
```

Figure 19 – Preparation of twitter table

After this last preprocessing, the two data sets were ready for the merge.

5.4 Final data warehouse table

In a last step, the two tables were merged to create the final data warehouse for the visualization. The creation of the described steps was all done within the DAG “Datawarehouse_quakes_tweetcount.py”

```
# merge tables
earthquake_tweetcount = earthquakes_df.merge(twitter_df, on='date', how='left')

logging.info(f'{earthquake_tweetcount.head(10)} records_quakes')
logging.info(f'{earthquake_tweetcount.columns} records_quakes')

# write data to RDS table
# Now set up insert statement
sql = 'INSERT INTO datawarehouse_earthquake_tweetcount(id, magnitude, time, type, title, coord, tweetcount)'
for index, row in earthquake_tweetcount.iterrows():
    try:
        sql += ("('{}', {}, '{}', '{}', '{}', {}, {}, '{}', {}), ".format(row['id'], row['magnitude'], row['time'], row['type'], row['title'], row['coord'], row['tweetcount'], row['lat'], row['lon']))
    except:
        logging.info("Problem with earthquake ({} , {})".format(row['id'], row['title']))

dag = DAG(
    'earthquake_tweetcount_datawarehouse',
    schedule_interval='@daily',
    start_date=datetime.datetime.now() - datetime.timedelta(days=30))
```

Figure 20 – Load data into data warehouse

The final data warehouse table looks in the end like this:










 datawarehouse_ea... Id	 datawarehouse_eart... Magnitude	 datawarehouse_earthquak... Time	 datawarehouse_... Type	 datawarehouse_earthquake_t... Title	 datawarehouse_ea... Coord Lat	 datawarehouse_earth... Coord Long	 datawarehouse_... Date	 datawarehous... Count
usp000jc8y	5.10000	16.12.2011 16:47:26	earthquake	M 5.1 - 248 km NW of ...	94.023	7.725	16.12.2011	5'443.00
usp000jc8u	5.80000	16.12.2011 13:54:25	earthquake	M 5.8 - 254 km W of P...	-76.056	-45.766	16.12.2011	5'443.00
usp000jc8s	5.40000	16.12.2011 13:02:57	earthquake	M 5.4 - 252 km W of P...	-76.020	-45.828	16.12.2011	5'443.00
usp000jc8e	5.00000	16.12.2011 02:37:22	earthquake	M 5.0 - Vanuatu region	171.590	-17.079	16.12.2011	5'443.00
usp000jc80	5.60000	15.12.2011 16:12:48	earthquake	M 5.6 - Izu Islands, Ja...	141.631	31.717	15.12.2011	6'950.00
usp000jc7t	5.10000	15.12.2011 14:28:03	earthquake	M 5.1 - 49 km SSW of ...	-88.714	12.867	15.12.2011	6'950.00
usp000jc7r	5.30000	15.12.2011 14:05:55	earthquake	M 5.3 - Izu Islands, Ja...	141.665	31.675	15.12.2011	6'950.00
usp000jc7p	6.00000	15.12.2011 11:10:07	earthquake	M 6.0 - south of the K...	-179.099	-32.718	15.12.2011	6'950.00
usp000jc7e	5.20000	15.12.2011 08:12:08	earthquake	M 5.2 - Kermadec Isla...	-176.317	-29.492	15.12.2011	6'950.00
usp000jc76	5.30000	15.12.2011 03:59:12	earthquake	M 5.3 - South Sandwic...	-28.269	-55.367	15.12.2011	6'950.00
usp000jc75	5.00000	15.12.2011 03:49:25	earthquake	M 5.0 - 179 km SE of ?...	143.338	38.069	15.12.2011	6'950.00
usp000jc68	5.50000	14.12.2011 12:07:01	earthquake	M 5.5 - Pacific-Antarc...	-143.727	-56.533	14.12.2011	8'685.00
usp000jc5z	7.10000	14.12.2011 06:04:59	earthquake	M 7.1 - 25 km SSE of ...	146.809	-7.551	14.12.2011	8'685.00
usp000jc5y	5.10000	14.12.2011 05:01:08	earthquake	M 5.1 - 4 km S of Toki, ...	137.188	35.308	14.12.2011	8'685.00
usp000jc5u	5.80000	14.12.2011 01:48:10	earthquake	M 5.8 - 127 km WNW ...	-174.826	-15.382	14.12.2011	8'685.00

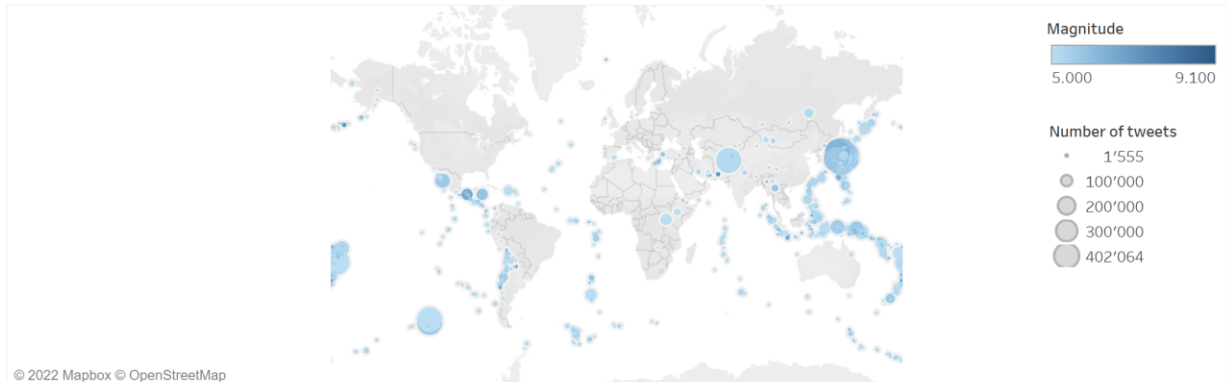
Figure 21 – Final data warehouse table

The data warehouse is basically the earthquake dataset, that is enriched with the tweet count per day. What can be seen is that the count per day is assigned to every earthquake with the same day.

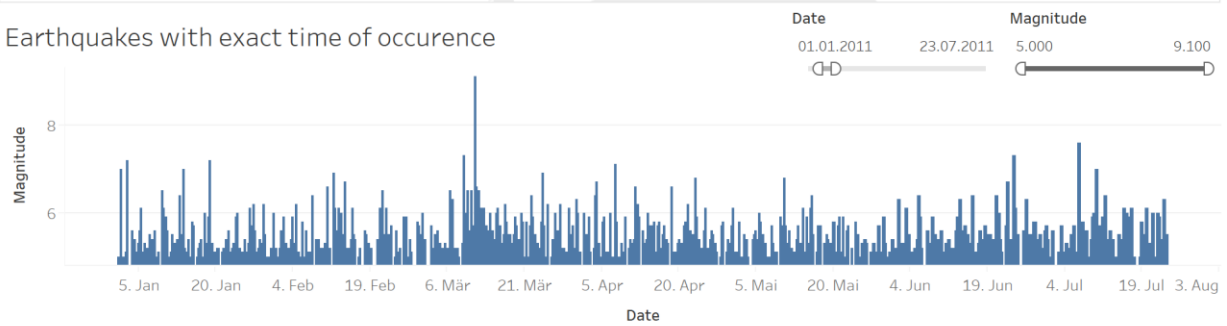
6 Visualization

The final visualization can be seen in the following illustration:

Worldmap



Earthquakes with exact time of occurrence



Number of tweets



Figure 22 – Final dashboard

The dashboard was created with the software Tableau and is showing three graphics. They are briefly described in the following chapters.

An interactive version of the dashboard can also be found online under the following link:

https://public.tableau.com/app/profile/sandro4133/viz/earthquake_twitter_dashboard_final/Dashboard1?publish=yes

6.1 Individual graphics

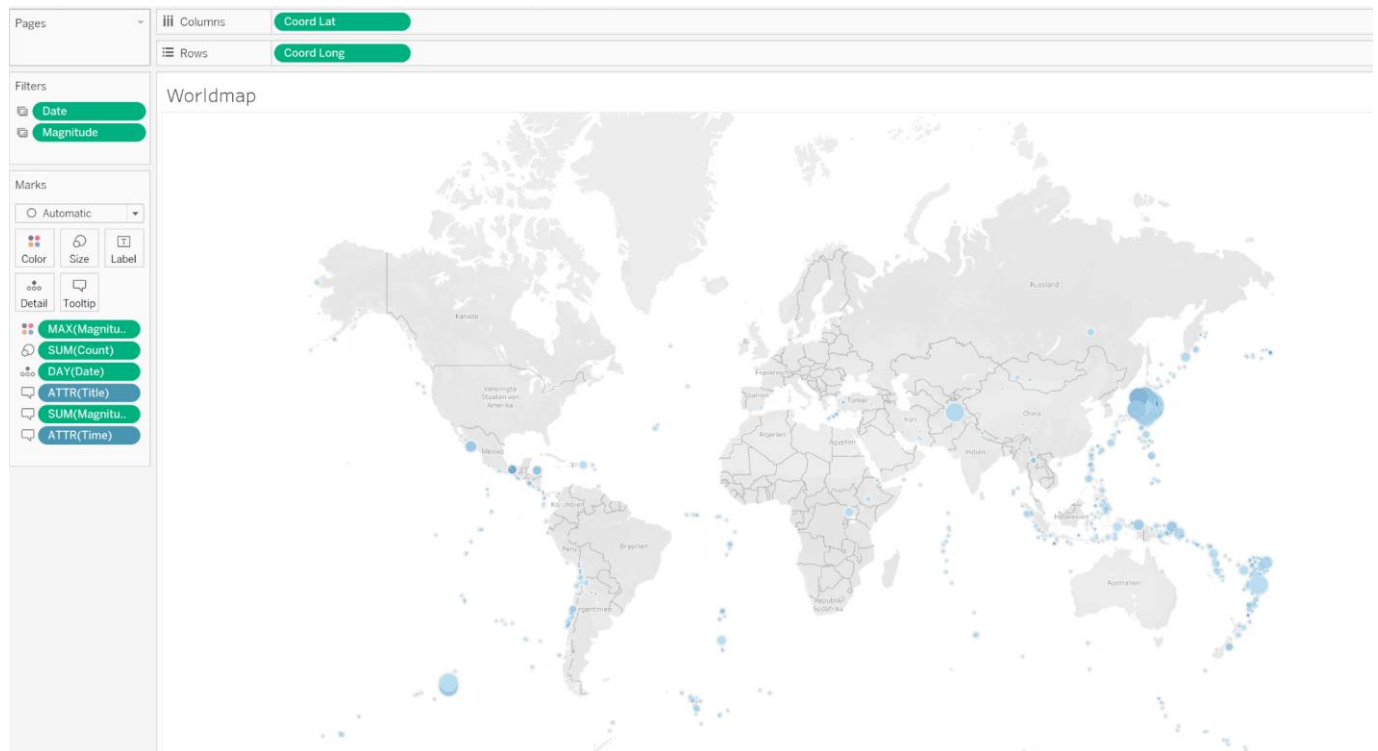


Figure 23 – World map

The world map shows where earthquakes were occurring on a map with help of blue dots. The colour indicates how severe the earthquake was, a light blue indicates a low magnitude, and a dark blue indicates a high magnitude. The size of the dots reflects the twitter echo - the bigger the dot, the more tweets has been sent at the respective date. Moreover, when hovering over the dots, additional information is revealed:

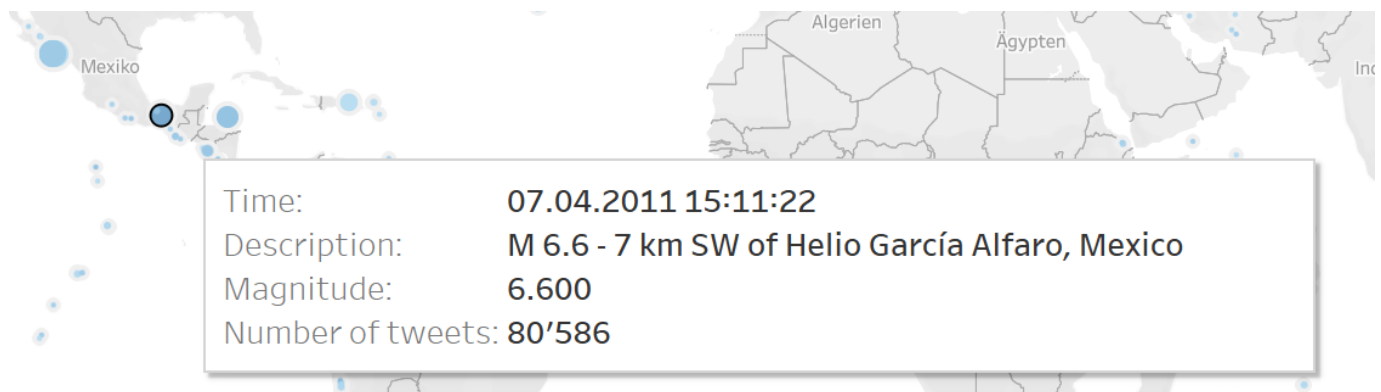


Figure 24 – Hovering effects

In the next visualization, each earthquake is listed and structured to the exact time of occurrence. Also here when hovering over an earthquake, more information is revealed:

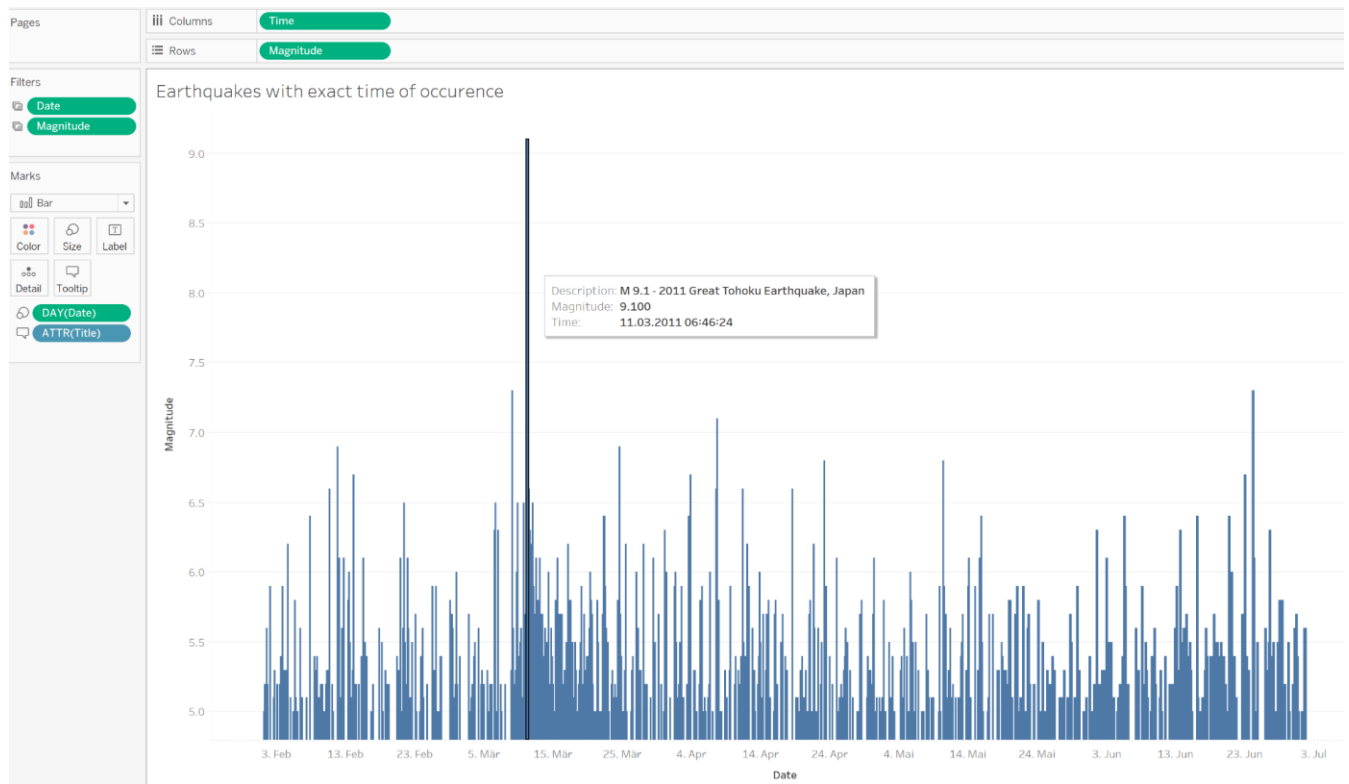


Figure 25 – Detailed earthquakes diagram

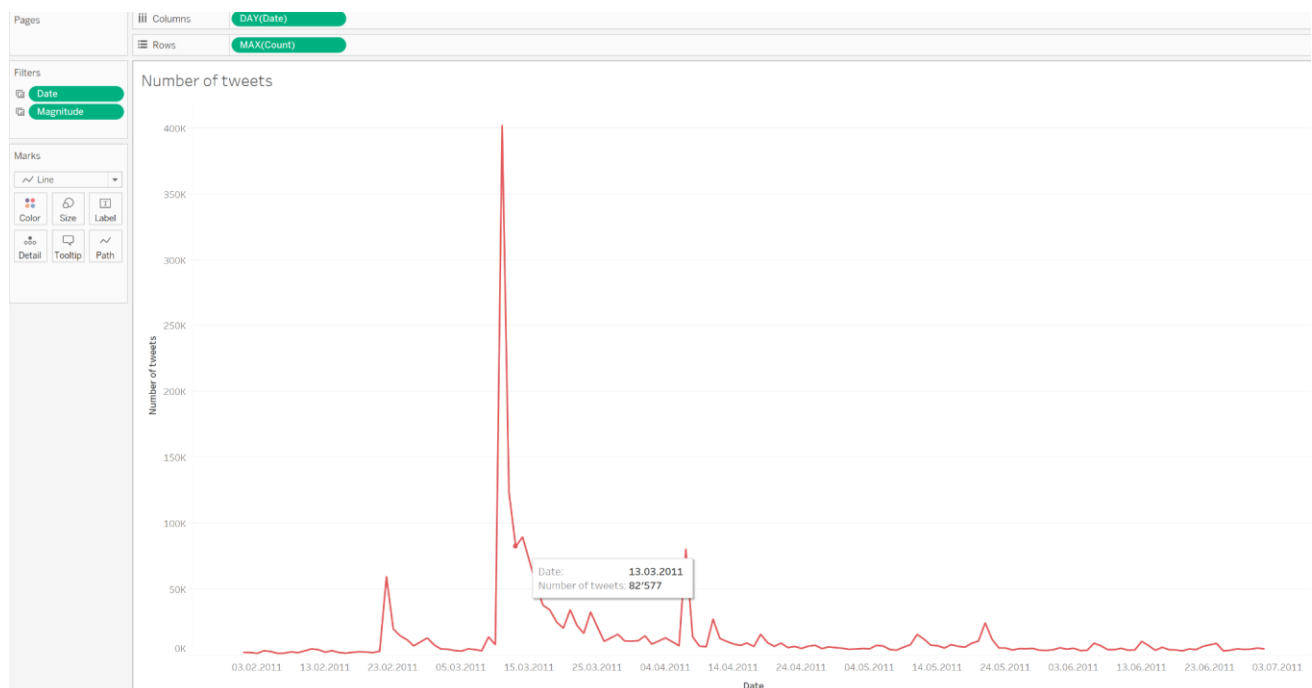


Figure 26 – Detailed chart about number of tweets

This chart finally shows the number of tweets sent per day with numerical information revealed by hovering over it.

6.2 Dashboard filters

The dashboard can be filtered with two main indicators, those are the date and the magnitude:

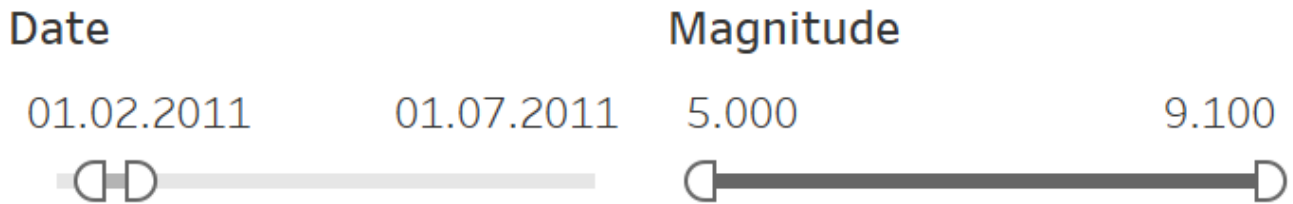


Figure 27 – Filtering options

The reader can easily filter out earthquakes that were happening in a certain timeframe and can for example also analyse how many tweets were sent for earthquakes with a certain magnitude. Other questions can easily be answered by applying these filters. Additionally, there are other filters and highlighters available on the board.

When clicking on a point on the map, the other graphics get adjusted as well and the concerning earthquake is highlighted.

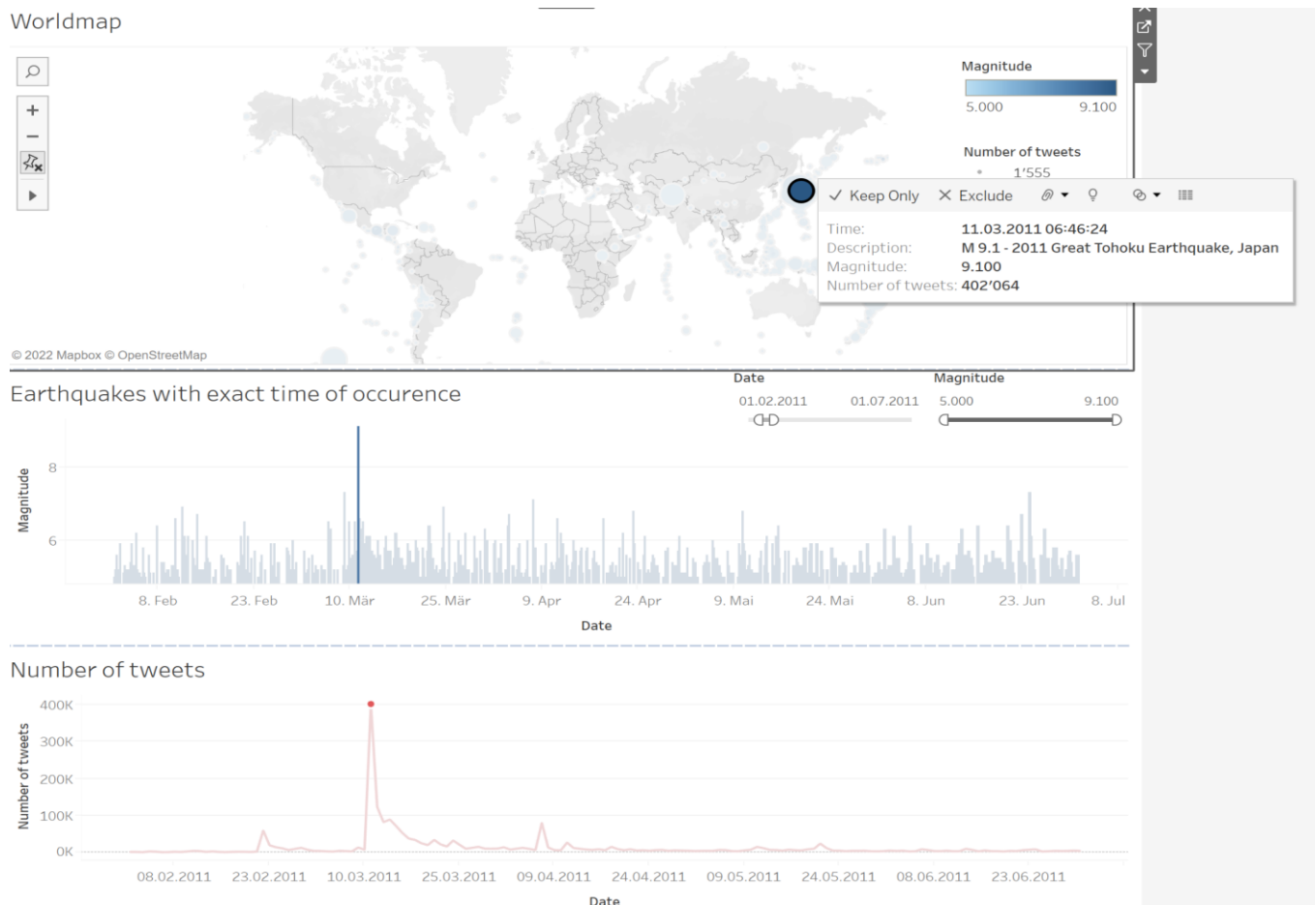


Figure 28 – Additional highlighting / filtering options

When clicking on a date from the number of tweets-graphic, this day gets filtered in the other graphic.

Worldmap

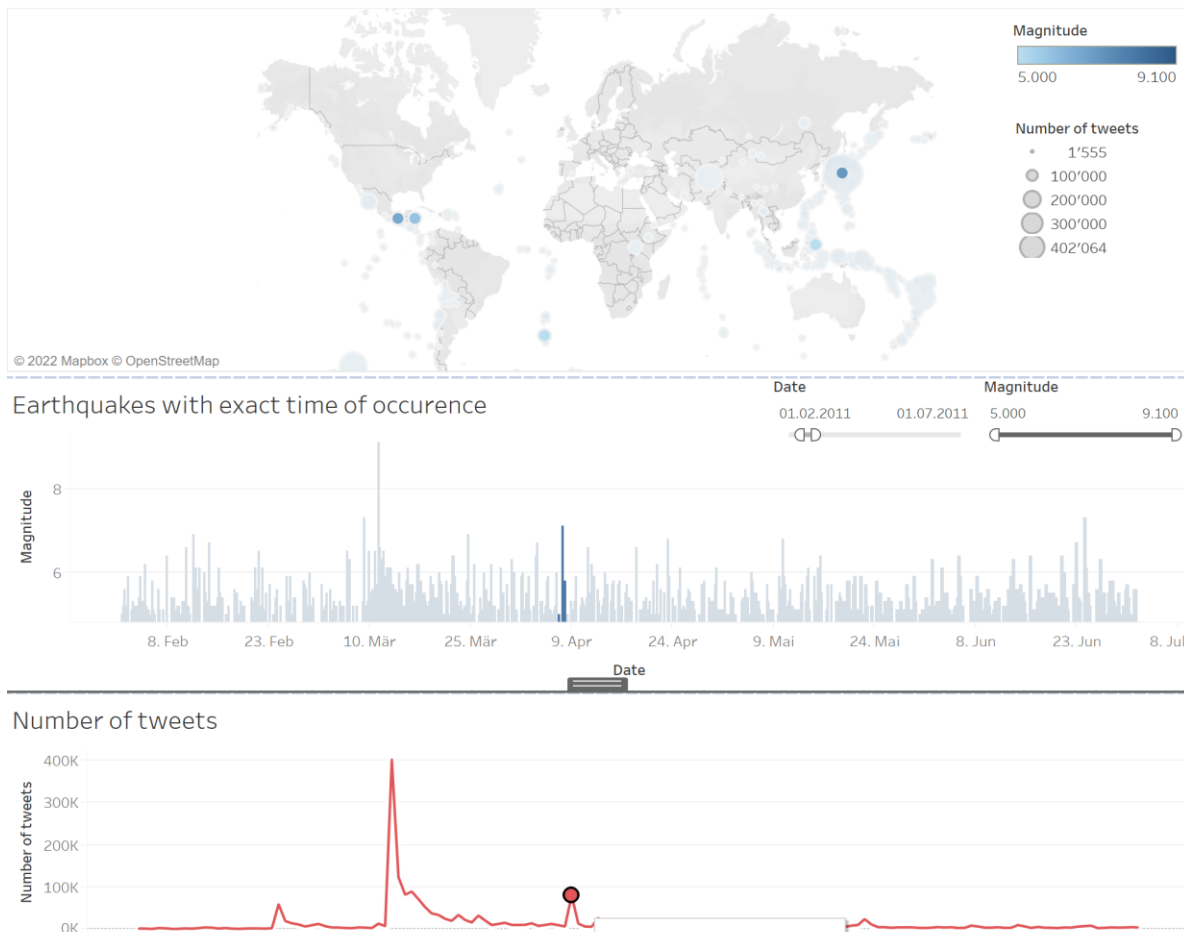


Figure 29 - Additional highlighting / filtering options

7 Conclusions

Analysis of earthquake-related tweets over the past 10 years has shown that strong earthquakes are reflected on twitter, making such an evaluation suitable as an early indicator of upcoming damage reports in the future. However, there are some limitations to this: Of course, this correlation can only be seen in highly populated areas of the world with a large distribution of the social media service twitter. The final dashboard cannot provide reliable information for poorer regions without the corresponding mobile network coverage or smartphone density. From the customer SwissRe's point of view, however, this should cover the financially relevant areas.

7.1 Research Questions

7.1.1 At what magnitude do earthquakes trigger a reaction on twitter?

In our research, we limited ourselves to earthquakes with a minimum magnitude of 5 on the Richter scale. These are classified as "moderate" and defined by serious damage to vulnerable buildings and light damage to robust buildings – i.e. the category relevant to our clients SwissRe. On average, around 1,800 such earthquakes occur each year. (Wikipedia Richter scale)

As can be visually seen from the final dashboard, the highest peaks in the tweet counts relate to some of the largest earthquakes in recent years, for example the Tohoku earthquake with the subsequent nuclear catastrophe in Fukushima, Japan in March 2011 with a magnitude of over 9. However, there are also a handful of earthquakes with significantly lower magnitudes (mostly from around 6 on the Richter scale) which are already generating a discernible response on Twitter. These include, for example, the Ridgecrest earthquakes in the Los Angeles region in early July 2019 with magnitudes between 5.4 and 7.1.

7.1.2 Is there a dependency between the strength of the earthquake and the number of tweets sent?

According to current findings, the degree of dependence between the strength of the earthquake and the number of tweets sent depends primarily on the region in which the earthquake occurs. In the USA in particular, twitter's home country, even relatively weak earthquakes can lead to strong reactions on Twitter. A general statement for all parts of the world cannot be made with the data used in this project.

It is further noteworthy that the inclusion of tweets from bots distorted the desired result. As the finalized dashboard uses only the number of tweets per day from all users, even earthquakes with barely any effect will trigger a response of at least a couple hundred tweets. It is assumed that the exclusion of these bots would reduce the reaction on Twitter for weak earthquakes, while still showcasing a strong response for impactful earthquakes. In a next step, the project could be expanded by working with the tweets where most of the bots were removed. As a first test, it would also be possible to simply aggregate the count of this data set. Due to the limited number of tweets that were extractable with the Twitter API, it was not possible to get the data for the full predefined time frame, and consequently this test was omitted.

7.1.3 How did the sentiment in the tweets change based on the magnitude?

Word clouds were created to analyze the sentiment in tweets after strong earthquakes. After the above-mentioned earthquake in Japan in March 2011, in addition to the keyword "Japan", the words "tsunami", "hope" or - of particular interest to SwissRe - "damage" were also found very often. Because NLP was not the main topic of this course, we only scratched the surface here. However, we see great potential to gain more valuable insights for our client SwissRe through deeper analysis



Figure 30 – Word cloud of the tweets after Japan earthquake in March 2011

8 Sources

Benita, H. (2019, July 9). *Haki Benita*. Retrieved at 08.11.2021 from <https://hakibenita.com/fast-load-data-python-postgresql>

SwissRe. (2020, December 15). *Swiss Re Institute estimates USD 83 billion global insured catastrophe losses in 2020, the fifth-costliest on record* Retrieved at 01.11.2021 from <https://www.swissre.com/media/news-releases/nr-20201215-sigma-full-year-2020-preliminary-natcat-loss-estimates.html>

USGS (2010, May 24). *Earthquake Facts and Statistics*. Retrieved at 15.10.2021 from <https://web.archive.org/web/20100524161817/http://earthquake.usgs.gov/earthquakes/eqarchives/year/eqstats.php>

Wikipedia Richter scale (2022). <https://de.wikipedia.org/wiki/Richterskala>

8.1 Images

Title page: Unsplash.com

All other images: Screenshots produced by the group

8.2 Code

All referenced code and results can be found on GitHub:

https://github.com/finite001/DWL_earthquakes