2020

DATA7201 Project

PREDICTING NUMBER OF TWEETS BASED ON WEATHER

ROBERT WILLIAMSON (S4575121)

Abstract

(1,493 words excluding this abstract, headers, figures, tables, captions and the appendix)

Predictive services are required in many industries to allow businesses and service providers to plan more effectively into the future. Getting this wrong could mean under or over rostering staff and under or over ordering stock, thereby not satisfying customer expectations or losing money on unused product and underutilized staff. Prediction helps business owners find the sweet spot.

This paper attempts to predict the amount of geo-tagged tweets on a given day and location under certain weather conditions as a proxy for human activity or getting out and about.

Due to the size of the Twitter dataset analysed (circa 420TB) this is not possible on a conventional computer. So distributed systems are used to solve this problem. While the amount of data required to undertake this analysis is not huge, there is a large amount of data to query to access the data required. This includes:

1. Finding tweets that have location data (geo-tagged),
2. Filtering for the location of interest, and finally
3. Extracting relevant data for the analysis such as the time created and hashtags

There is a big assumption that underpins this analysis and that is that Twitter users tag their location when they are out and about, like users on Facebook or Instagram. However, it may be that some portion of users have an application setting enabled that always tags their location. In this case what our model is predicting is the volume of tweets based on the weather.

We were able to fit weather data in the area of New York to the number of tweets tagged in the same area with an R^2 of 53%, which means the model can explain about 53% of the variance. Put into context, an R^2 of 0 would predict the number of tweets as good as taking an average over the period. It is acknowledged that the model is predicting the data it was trained on, so it very well may not perform as well on new data.

Word clouds produced of the hashtags used on the highest tweeted days show some consistency with good weather with tags like #sundayfunday, #goodmorning and #centralpark indicating that people are talking about enjoying the outdoors. However, words like #sailing and #sunset also appear in the low tweeted days.

Table of Contents

Introduction

The motivation for this report was inspired by a Redditor on r/datascience who provides predictive services to local restaurants predicting patronage to allow restaurateurs to plan more effectively for the coming week.

Getting this wrong could mean under or over rostering staff and under or over ordering stock, thereby not satisfying customer expectations or losing money on unused product and underutilized staff. Prediction helps business owners find the sweet spot. It was stated that the strongest predictor is the weather. The analysis presented in this report will attempt something similar.

Purpose

- To predict the amount of geo-tagged tweets on a given day and location under certain weather conditions.
- For the purpose of this report, geo-tagged tweets are those that contain location data as coordinates or a bounding box of coordinates.

How does Big Data help?

The dataset supporting this analysis represents 1% of tweets for 6 months between July and December 2014, this equates to about 420TB. Storing and querying this size dataset is not possible on a conventional computer.

Distributed systems solve this problem. Huge datasets are stored and analysed on distributed networks of computers. The workload is split over each node (computer) in the network and the results are combined back together to return the desired output.

This process is known as map-reduce and was first developed by Google to manage their huge datasets gathered from crawling and indexing webpages. The data provided for this report is stored on a Hadoop distributed file system (HDFS) and Apache Pig is used to query the data. Pig provides a console-based user interface to enter map-reduce jobs for the system to carry out.

While the amount of data required to undertake this analysis is not huge, there is a large amount of data to query to access the data required. This includes:

1. Finding tweets that have location data (geo-tagged),
2. Filtering for the location of interest, and finally
3. Extracting relevant data for the analysis such as the time created and hashtags

Data Analysis

As well as the Twitter dataset, weather data from the New York City area [2] was sourced from the National Oceanic and Atmospheric Administration (NOAA) of the United States of America (US). New York was chosen as a populous and developed location assumed to have a lot of Twitter users.

Location data is stored in two forms in the Twitter dataset. As a bounding box of latitude and longitude coordinates stored in the "place" field and as a point reference representing a location in the "coordinates" field. A spot check of the bounding boxes indicated that these can be as general an entire state or country, while coordinates show an exact location provided by a device or API [1]. Therefore, coordinate data is considered more accurate and preferred. If a bounding box is only available, a point is estimated using the center of the box.

Pre-processing

Getting the data

Three queries were executed using Apache Pig to extract data for analysis, these are as follows[1]:

1. Get geo-tagged tweets – filters for tweets with non-null coordinate or place fields and retrievs time created, hashtags and coordinates
2. Count total number of tweets per day – to provide context on the proportion of tweets geo-tagged
3. Count number geo-tagged tweets per day.

Filtering by location and extracting coordinates

This step proved tricky and was not implemented in Pig. Instead, the output of the first Pig script was batch processed using python to filter for a broad location over the upper east coast of the US. The resulting geo-tags are shown in Figure 1.

On average 4.4m tweets were counted per day in the Twitter dataset. Of these, approximately 2-2.8% contained a geo-tag. A total of 18.4m geo-tagged tweets were processed and filtered to 1.2m tweets.

Refining the search

The broad initial filter was over-kill and only the New York City area are required. To refine the filter a reverse coordinate lookup was applied in python. This identified the counties that each tweet fell within and following counties were filtered for:

- New York County
- Queens County
- Kings County
- Richmond County
- Nassau County
- Suffolk County

The result are shown in Figure 2, which covers about 13% of the 1.2m tweets tagged in the upper east of the US.

[1] Refer Appendix 1 for Pig scripts

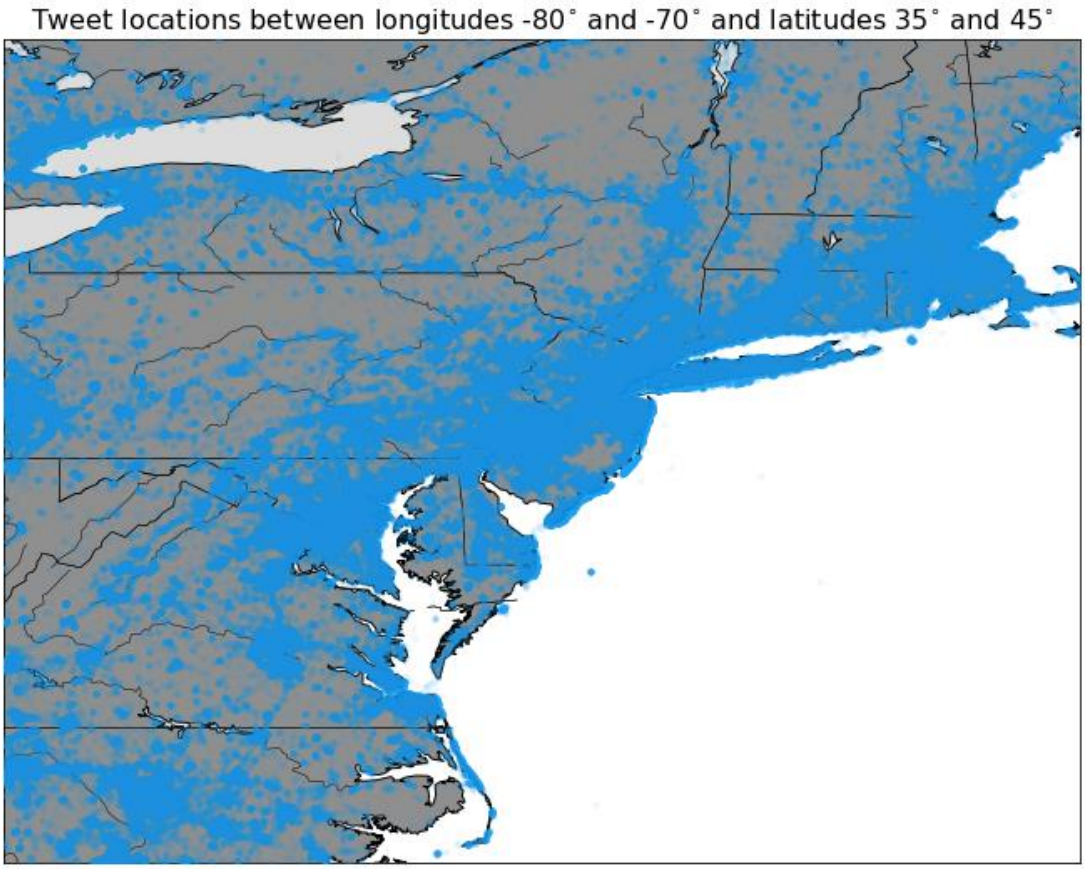


Figure 1: Geo-tagged tweets over the entire 6 months of the dataset with a coordinate within the region bound by the longitudes -80 to -70 and latitudes 35 to 45 (in degrees)

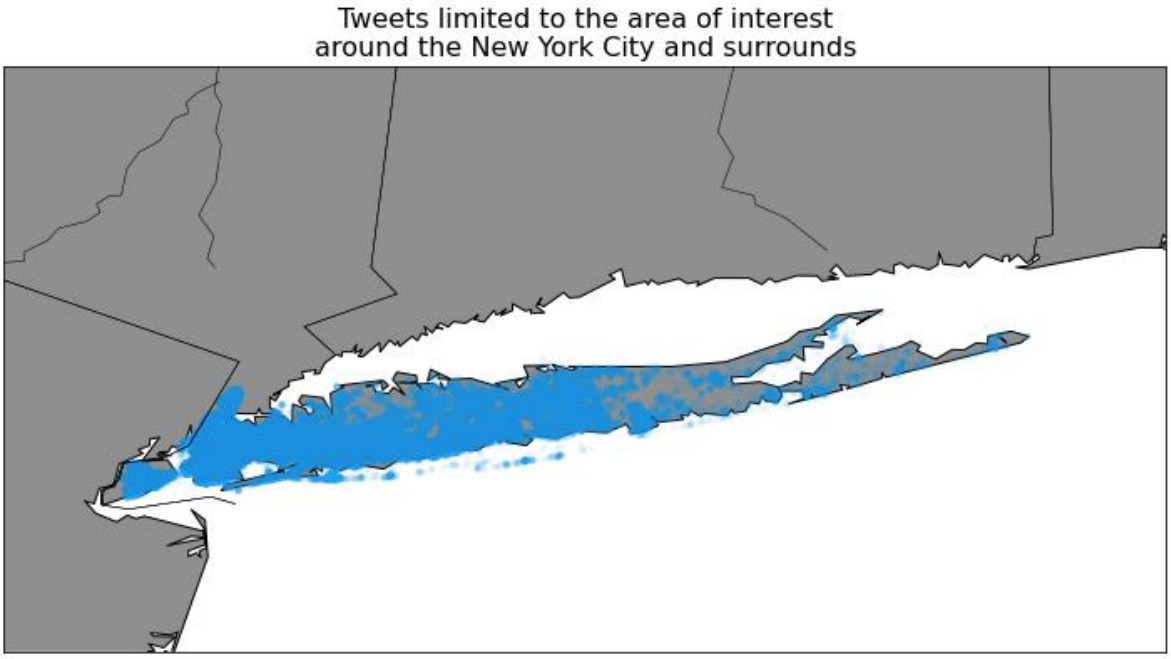


Figure 2: Results of the refined filtering for New York counties in the New York city area

Counting tweets

The tweets were aggregated daily counts shown in Figure 3. The time stamps had to be converted from Universal Time Coordinate (UTC) to US-Eastern time. Note the dramatic drop in geo-tagged tweets in mid-September.

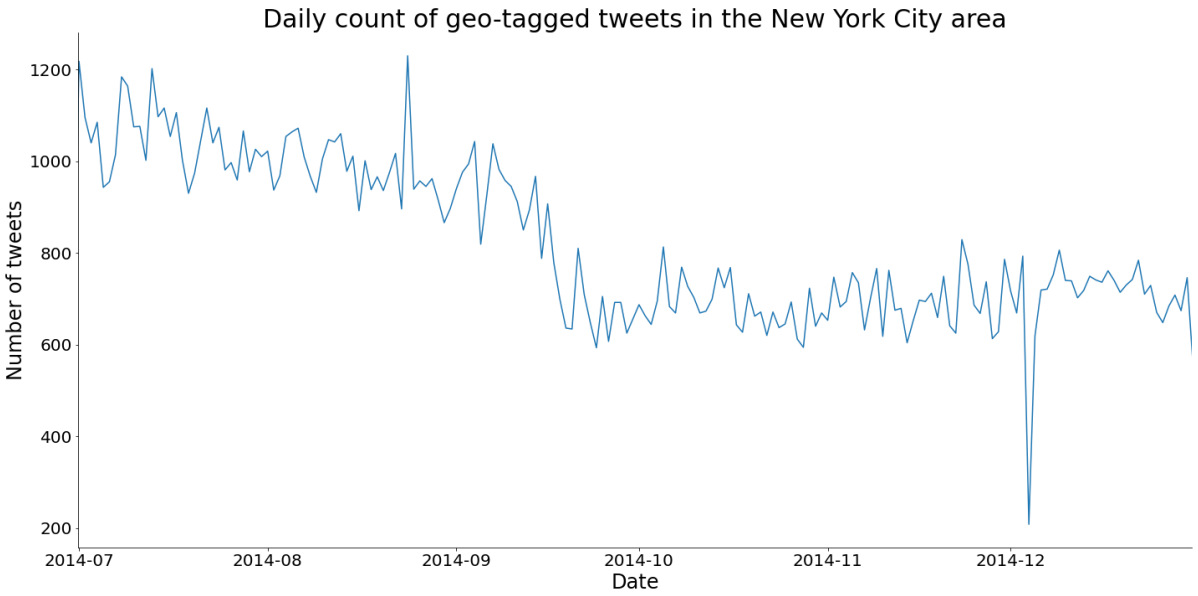


Figure 3: Daily counts of tweets in the New York area.

Weather data

Weather data was downloaded across 89 separate weather stations and aggregated to take the mean value for each parameter per day. The weather parameters considered for analysis and their units are listed in Table 1.

Table 1: Weather data sourced from the NOAA

Parameter	Units
Snowfall	Millimeters
Precipitation	
Average wind speed	Meter per second
Minimum temperature	Degrees Celsius
Average temperature	
Maximum temperature	

Exploratory Analysis

Exploratory analysis of the data was undertaken for the purpose of building a predictive model.

Day of the week

Friday and Saturday have the least tweets on average and Sunday is the highest, but the variation between days is minimal.

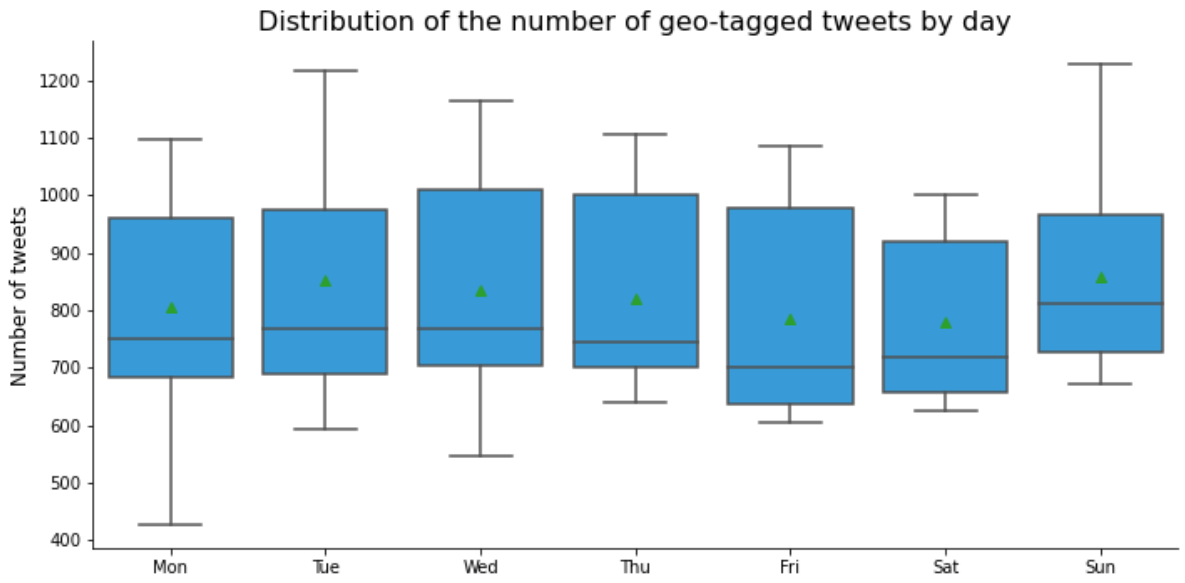


Figure 4: Distribution of tweets by day of the week

Temperature

The temperature data is plotted in Figure 5 below. There are striking similarities in the trend compared to Figure 3. A dramatic drop in temperature occurs in mid-September as the season changes

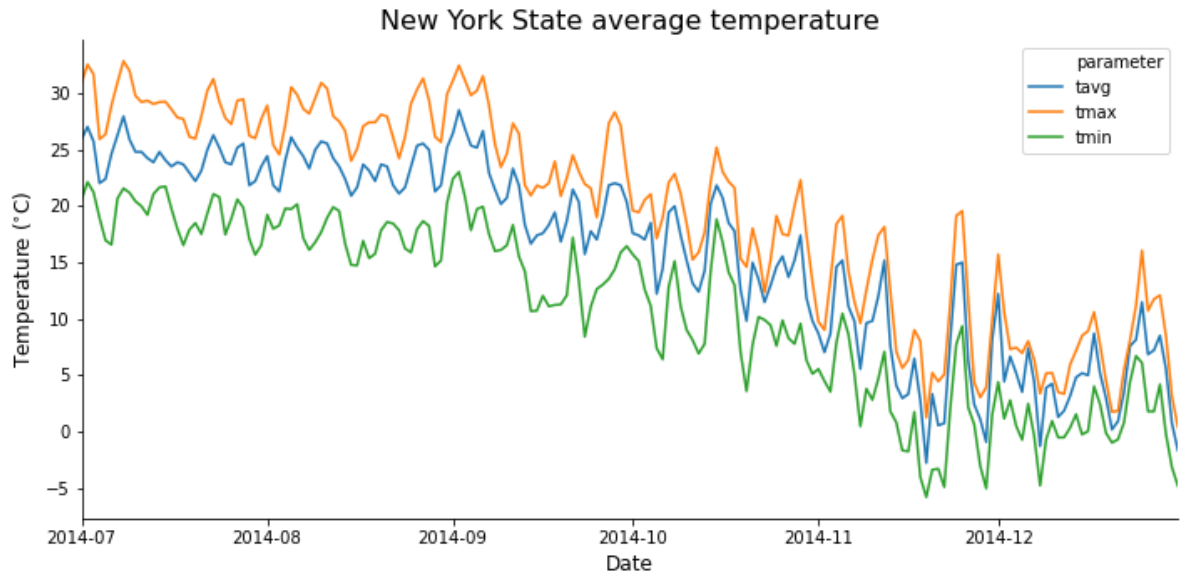


Figure 5: Minimum, average and maximum temperature

The temperatures follow a very similar pattern with each other and to avoid covariance issues only the average temperature will be used and the range between the minimum and maximum will be added to the dataset.

Bad weather parameters

Figure 6 shows discouraging weather parameters. To simplify the analysis these will be converted to binaries based on thresholds in Table 2. Thresholds are based on my own judgement of what are considered uncomfortable conditions.

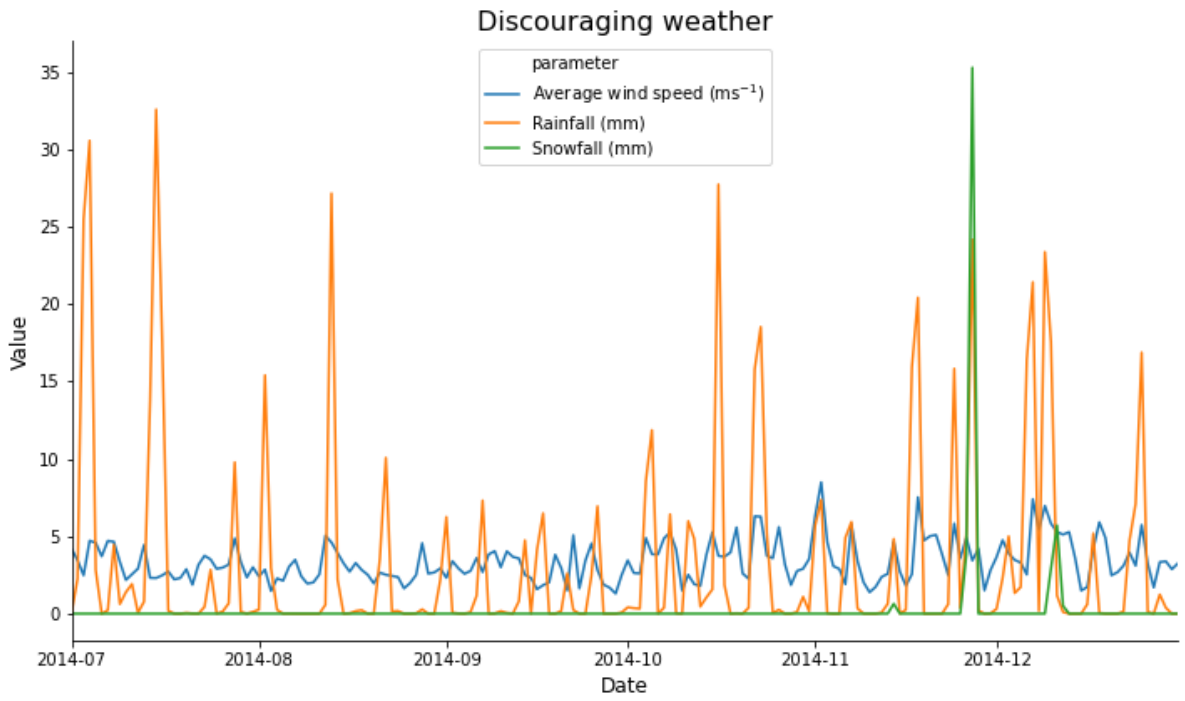


Figure 6: Composite plot of the not so good weather indicators

For wind speed the 75th percentile is used as average wind speed is harder to judge, see Figure 7.

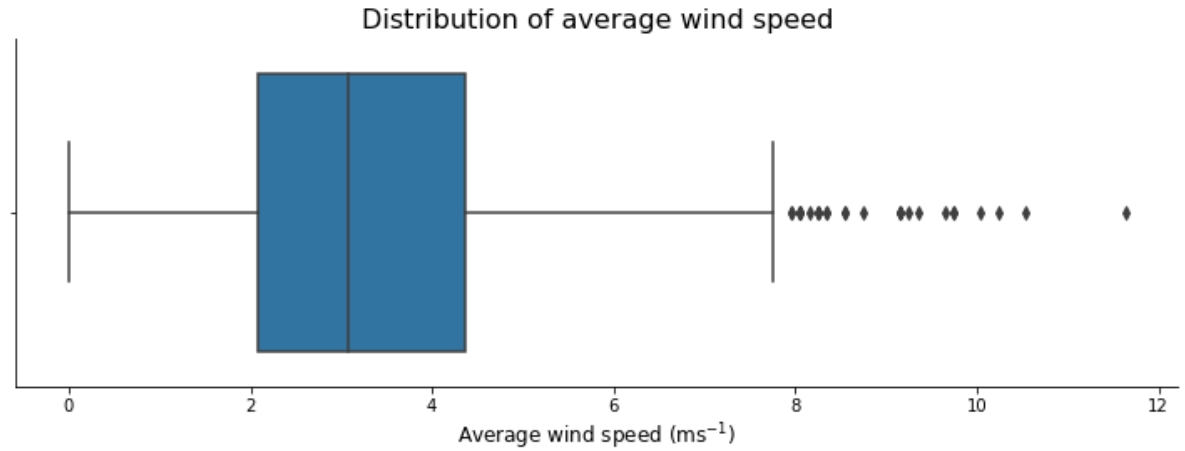


Figure 7: Distribution of wind speed (ms^{-1})

Table 2: Thresholds for bad weather

Parameter	Threshold
Snowfall	Above 5mm
Precipitation	Above 10mm
Average wind speed	Above 4.5 ms^{-1}

Predictors

The final set of predictors are:

- Average temperature (continuous)
- Temperature range (continuous)
- Day of the week (ordinal, 0-6)
- Is snowing (binary)
- Is raining (binary)
- Is windy (binary)

Fitting a model

A multiple regression model with Lasso regularization is fit using the above predictors and the number of tweets each day as the response. Regularization reduces the effect of any predictor that does not contribute to fitting model.

Results of the fit are shown in Figure 8. The model fit the data with an R^2 value of 53%, which means that the model explains approximately 53% of the variance in the data, which is a reasonable result.

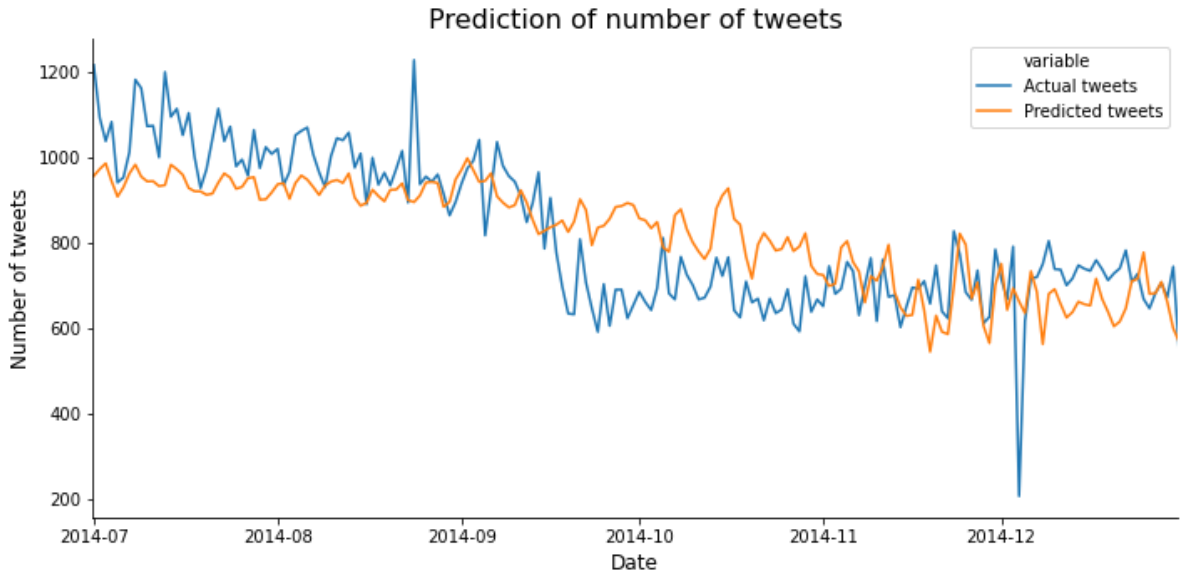


Figure 8: Model fit results

The average error prediction error using 10-fold cross-validation is approximately 134 tweets. Regularization diminished the effect of wind and the day of the week. Therefore indicating that snow, rain, temperature and temperature range are all useful in predicting activity.

High and low days

Finally, some word clouds are shown to explore the hashtags being used on days with the highest tweets and highest predicted tweets. The day with the most tweets in the New York City area is 24 August and the highest predicted day is 2 September. While, the lower day and predicted day are 4 December and 19 November respectively.

Highest days

Words indicating good weather and activity are present in Figure 9 such as #sundayfunday and #beach. Likewise, in Figure 10 outing related words may include #goodmorning and #centralpark.

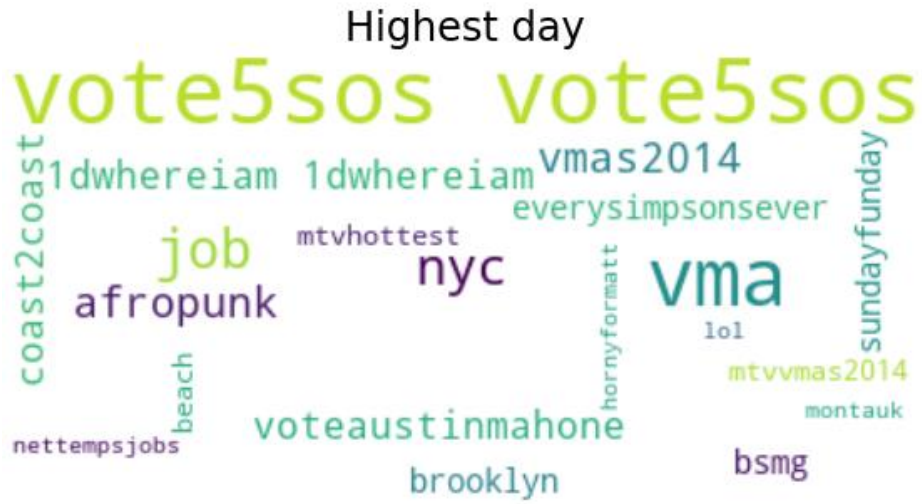


Figure 9: Word cloud showing the highest occurring hashtags on the day with the most tweets

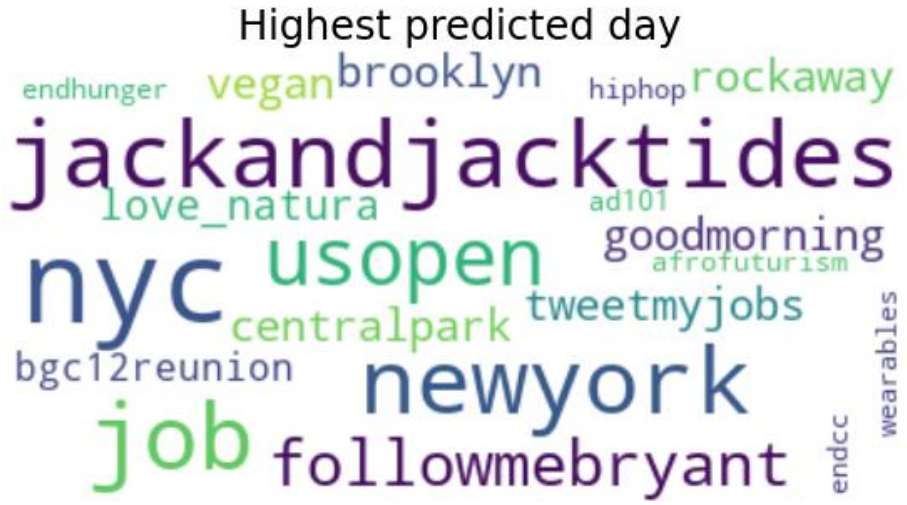


Figure 10: Word cloud showing the highest occurring hashtags on the day predicted to have the most tweets

Lowest days

Here, Figure 11 contains words such as #icantbreath support the claim, but, #sailing and #sunset do not. Likewise, there is nothing to indicate bad weather and staying-in in Figure 12 and hashtags like #winewednesday may indicate going out.

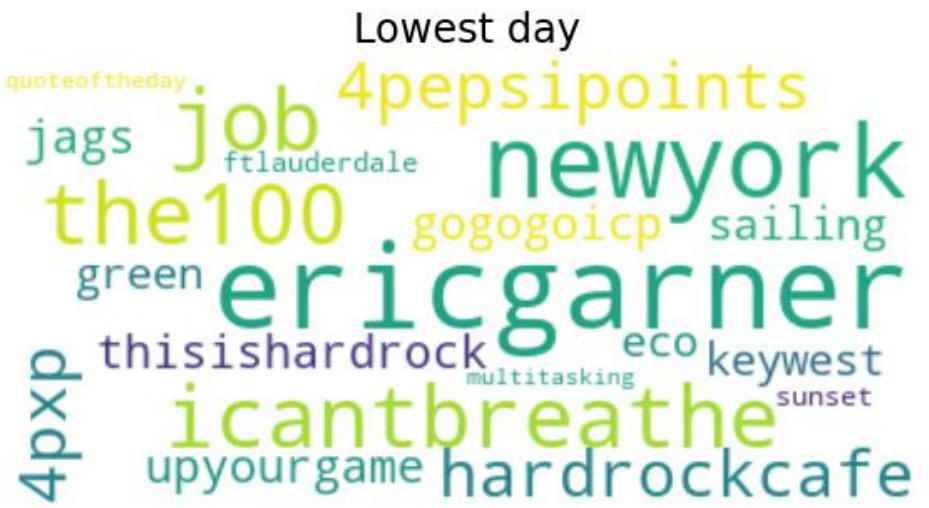


Figure 11: Word cloud showing the highest occurring hashtags on the day with the lowest number of tweets

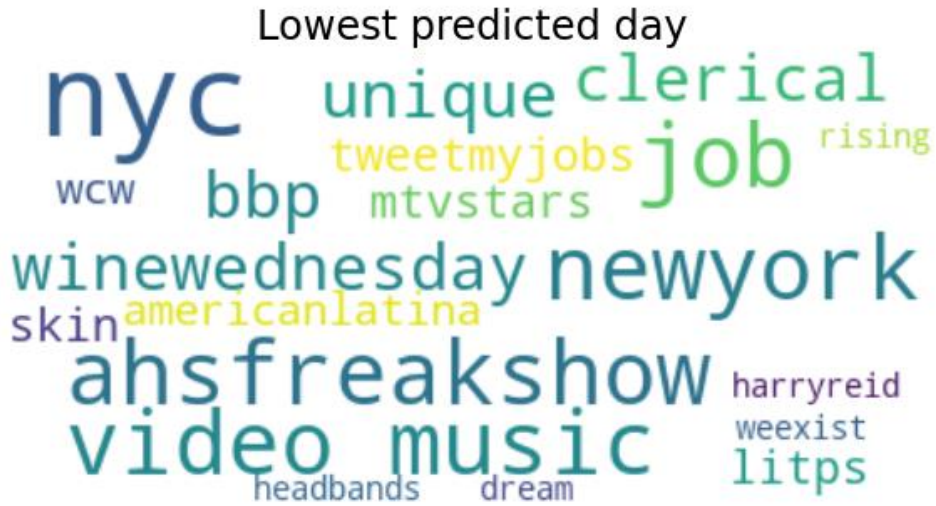


Figure 12: Word cloud showing the highest occurring hashtags on the day predicted to have the lowest number of tweets

Conclusion

There is a big assumption that underpins this analysis and that is that Twitter users tag their location when they are out and about, like users on Facebook or Instagram.

However, it may be that some portion of users have an application setting enabled that always tags their location. In this case what our model is predicting is the volume of tweets based on the weather. Regardless of this the model was able to fit the data with an R^2 of 53% which means it can explain about 53% of the variance. To put that into context an R^2 of 0 explains the amount of tweets as good as taking the average. Although it should be said that the model is predicting the data it was trained on, so it might not perform as well on new data.

The word clouds produced show some consistency with this hypothesis with tags like #sundayfunday, #goodmorning and #centralpark indicating that people are talking about enjoying the outdoors or the weather on high tweeted days. While, words like #sailing and #sunset also appear in the low tweeted days.

Big Data

This analysis would not have been possible without the power of the distributed file system HDFS and Pig to query the huge 420TB of tweets provided for analysis. The analysis would scale better if the geographical filtering was undertaken in Pig also, however this was tricky to implement.

Luckily, because the amount of geo-tagged tweets and the small portion of tweet data used, the filtering was relatively easy to undertake on a laptop using python.

Production

Because of the general nature of the analysis, application of such a model using forecast weather may be useful to predict patronage on transport systems, entertainment precincts and parks. Forecasting would only need to be undertaken using batch processing, so infrastructure such as Pig would be appropriate.

References

1. https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/geo-objects
2. https://www.ncdc.noaa.gov/cdo-web/datasets/GHCND/locations/CITY:US360019/detail

Appendix 1 – Code to get the data

The following sections contain the code used to undertake the analysis

Pig Scripts

Geo tags

This script was used to get the tweets with that contained coordinate data in either the place or coordinate fields

```
REGISTER 'hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020/user/hdfs/json-simple-1.1.jar';
REGISTER '
hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020//user/hdfs/elephant-bird-hadoop-compat-4.1.jar';
REGISTER 'hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020//user/hdfs/elephant-bird-pig-4.1.jar';

-- this is just one day, there is a bunch more data, once the script is working well
tweets_all = LOAD '/data/ProjectDataset/statuses.log.2014*' USING
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') AS (json:map[]);

-- to get the geo locations of tweets and get the hashtags
tweets_all = FOREACH tweets_all GENERATE FLATTEN(json#'created_at') as time_stamp, FLATTEN(json#'id')
as id, FLATTEN(json#'coordinates') as (coords:map[]), FLATTEN(json#'place') as (place:map[]),
FLATTEN(json#'entities') as (hash:map[]);

-- ensure that we remove duplicates
tweets = DISTINCT tweets_all;

-- now we can filter for tweets with geo tags
filtered = FILTER tweets BY (coords IS NOT NULL) OR (place IS NOT NULL);

-- parse the date time and unpack the geo data and hashtags
locs1 = foreach filtered generate coords#'coordinates' as coordinates:chararray,
FLATTEN(place#'bounding_box') as (bbox:map[]), ToDate(time_stamp, 'EEE MMM dd HH:mm:ss Z yyyy') as
time_stamp, FLATTEN(hash#'hashtags') as (tags:map[]), id as id;

-- unpack the hashtags and bounding box coordinates
locs2 = foreach locs1 generate id, time_stamp, coordinates, FLATTEN(bbox#'coordinates') as
bbox_coords:chararray, FLATTEN(tags#'text') as text;

-- group by id and throw the hashtags into a tuple
grpd = GROUP locs2 BY (id, time_stamp, coordinates, bbox_coords);
locs3 = foreach grpd generate FLATTEN(group) as (id, time_stamp, coordinates, bbox_coords),
TOTUPLE(locs2.text);

-- dump the filtered data and count the number of records
STORE locs3 INTO '
hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020/user/s4575121/project_output/geo_hash-2' USING
PigStorage('|');
```

Geo-tag counts

This script was used to count the number of geo-tagged tweets per day

```
REGISTER 'hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020/user/hdfs/json-simple-1.1.jar';
REGISTER '
hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020//user/hdfs/elephant-bird-hadoop-compat-4.1.jar';
REGISTER 'hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020//user/hdfs/elephant-bird-pig-4.1.jar';

-- this is just one day, there is a bunch more data, once the script is working well
tweets_all = LOAD '/data/ProjectDataset/statuses.log.2014*' USING
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') AS (json:map[]);

-- to get the geo locations of tweets
tweets_all = FOREACH tweets_all GENERATE FLATTEN(json#'created_at') as time_stamp, FLATTEN(json#'id')
as id, FLATTEN(json#'coordinates') as (coords:map[]), FLATTEN(json#'place') as (place:map[]);

-- ensure that we remove duplicates
tweets = DISTINCT tweets_all;

-- filter for tweets with geo or place tags
filtered = FILTER tweets BY (coords IS NOT NULL) OR (place IS NOT NULL);

-- parse the date time
time_stamps1 = foreach filtered generate ToDate(time_stamp, 'EEE MMM dd HH:mm:ss Z yyyy') as
time_stamp:DateTime, id as id;

-- just get the date from the datetime
time_stamps2 = foreach time_stamps1 generate id,
CONCAT((chararray)GetYear(time_stamp),'-',(chararray)GetMonth(time_stamp),'-',(chararray)GetDay(time_stam
p)) as date_string:chararray;

-- group by the date and count
by_date = group time_stamps2 by date_string;

-- count the number of records on each day
daily_count = foreach by_date generate Flatten(group) as day:chararray, COUNT(time_stamps2) as
num_tweets:int;

-- dump the filtered data and count the number of records
STORE daily_count INTO '
hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020/user/s4575121/project_output/geo_by_day' USING
PigStorage('|');
```

Tweet counts

This script was used to count the number of tweets in the dataset per day – although seemed to fail for some days

```
REGISTER 'hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020/user/hdfs/json-simple-1.1.jar';
REGISTER '
hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020//user/hdfs/elephant-bird-hadoop-compat-4.1.jar';
REGISTER 'hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020//user/hdfs/elephant-bird-pig-4.1.jar';

-- this is just one day, there is a bunch more data, once the script is working well
tweets_all = LOAD '/data/ProjectDataset/statuses.log.2014*' USING
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') AS (json:map[]);

-- get the time from the tweets
tweets_all = FOREACH tweets_all GENERATE FLATTEN(json#'created_at') as time_stamp, FLATTEN(json#'id')
as id;

-- ensure that we remove duplicates
tweets = DISTINCT tweets_all;

-- parse the date time
time_stamps1 = foreach tweets generate ToDate(time_stamp, 'EEE MMM dd HH:mm:ss Z yyyy') as
time_stamp:DateTime, id as id;

-- just get the date from the datetime
time_stamps2 = foreach time_stamps1 generate id,
CONCAT((chararray)GetYear(time_stamp),'-',(chararray)GetMonth(time_stamp),'-',(chararray)GetDay(time_stam
p)) as date_string:chararray;

-- group by the date and count
by_date = group time_stamps2 by date_string;

-- count the number of records on each day
daily_count = foreach by_date generate Flatten(group) as day:chararray, COUNT(time_stamps2) as
num_tweets:int;

-- dump the filtered data and count the number of records
STORE daily_count INTO '
hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020/user/s4575121/project_output/counts_by_day-1'
USING PigStorage('|');
```

Geo filtering

This python script was used to filter the output of the geo-tagged tweets – I could not get the filtering to work in Pig. This method proved to work well and was reasonably quick. I also developed a convenient console progress bar to ensure something was happening.

```
import pandas as pd
import os

'''
This script opens the geo twitter data and filters for the tweets by a bounding box
'''

COLS = ['id', 'time_stamp', 'coordinates', 'bbox', 'hashtags']
LOCCOLS = ['coordinates', 'bbox']
COORD_PATTERN = r"(-?\d+\.\d+)"

class ConsoleBar:
    def __init__(self, num_ticks):
        # check that end - start is greater than one and that they are both integers
        if type(num_ticks) is not int:
            raise TypeError('arg "num_ticks" must be type int')
        if num_ticks < 1:
            raise ValueError("num_ticks not > 0")

        #  get the absolute size and normalise
        self.num_ticks = num_ticks
        self.ticker = 0

        # start the ticker
        print('\r   {:>3.0f}%[{: <101}]'.format(0, '='*0+'>'), end='\r')

    def tick(self, step=1):
        print_end = '\r'
        self.ticker += step
        if self.ticker > self.num_ticks:
            self.ticker = self.num_ticks
            # print a warning
            print('Warning: The ticker is overreaching and has been capped at the end point', end=
            '\r')
        elif self.ticker + step == self.num_ticks:
            print_end = ''

        progress = int((self.ticker/self.num_ticks)*100)
        print('\r   {:>3.0f}%[{: <101}]'.format(progress, '='*progress+'>'), end=print_end)


def read_data(filename, columns=COLS):
    '''
    read the data as a csv
    path is the location of the file and file name
    '''
    # read in the data add the column headers and set the id as the index
    df = pd.read_csv(filename, sep='|', header=None, encoding='utf-8')
    df.columns = columns
    return df.set_index('id', drop=True)
```

```

def get_coords(df, col='coordinates'):
    '''
    Use regex pattern and extract to get the coordinates from the data
    '''

    try:
        coords = df[col].str.extractall(COORD_PATTERN).unstack(['match']).astype(float)
        coords.columns = ['longitude', 'latitude']
        coords['type'] = 'point'

        # concatenate with the original data and drop the location columns
        df = df.drop(LOCCOLS, axis=1)
        return pd.concat([df, coords], axis=1)

    except Exception as ex:
        print("Error in get_coords:")
        print(ex)

```

```
def get_bbox_centroid(df, col='bbox'):
    '''
    Making a big, but convenient assumption here that the centroid of the bbox is good enough

    From observing the strings the bbox has long-lat coordinates with the following points:
    - 0 bottom-left
    - 1 top-left
    - 2 top-right
    - 3 bottom-right
    '''
    try:
        # get the coords from bbox string
        columns = ["bottom_left_long", "bottom_left_lat",
            "top_left_long", "top_left_lat",
            "top_right_long", "top_right_lat",
            "bottom_right_long", "bottom_right_lat"]
        box_bounds = df[col].str.extractall(COORD_PATTERN).unstack(['match']).astype(float)

        box_bounds.columns = columns

        # calculate the centriods as the average of the bottom left and top right
        longs = box_bounds.loc[:, ['bottom_left_long', 'top_right_long']]
        lats = box_bounds.loc[:, ['bottom_left_lat', 'top_right_lat']]
        centroids = dict()
        centroids['longitude'] = longs.mean(axis=1)
        centroids['latitude'] = lats.mean(axis=1)
        centroids = pd.DataFrame(centroids)
        centroids['type'] = 'centroid'

        # concatenate with the original data and drop the location columns
        df = df.drop(LOCCOLS, axis=1)
        return pd.concat([df, centroids], axis=1)

    except Exception as ex:
        print("Error in get_bbox_centroid:")
        print(ex)
```

```
def get_preference(df, preference=LOCCOLS[0]):
    '''
    To speed up processing first assess if there are LOCCOLS[0] use those.
    Only use LOCCOLS[1] where LOCCOLS[0] is null.
    '''
    bbox = df[df[preference].isnull()]
    coords = df[df[preference].notnull()]

    # process and concatenate these
    bbox_coords = get_bbox_centroid(bbox)
    coordinates = get_coords(coords)
    return pd.concat([bbox_coords, coordinates])

def filter_bybox(df, longcol, latcol, bounds):
    """
    return the df filtered by a bounds
    bounds is a tuple with two tuples containing the lower-left and upper-right bounds
    longcol and latcol are the columns in df
    """
    return df[(df[longcol]>bounds[0][0]) &
              (df[longcol]<bounds[1][0]) &
              (df[latcol]>bounds[0][1]) &
              (df[latcol]<bounds[1][1])]
```

```
def main():
    # do things
    current_loc = os.getcwd()
    print("You are working from", current_loc)
    input_path = input("Where are the files to process? ")
    full_input = os.path.join(current_loc, input_path)
    if not os.path.exists(full_input):
        print(full_input)
        raise Exception("Location does not exist!")

    output_path = input("Where do you want to save the results? ")
    full = os.path.join(current_loc, output_path)
    if not os.path.exists(full):
        if input("  - path does not exist, would you like to create it? [y/other] ").lower()[0] ==
        'y':
            os.mkdir(full)
        else:
            raise Exception("bye!!")

    print("Please enter the lower left and upper right corners of the bounds you desire:")
    ll_lng = float(input("  - lower left longitude: "))
    ll_lat = float(input("  - lower left latitude: "))
    ur_lng = float(input("  - upper right longitude: "))
    ur_lat = float(input("  - upper right latitude: "))
    bounds = ((ll_lng, ll_lat), (ur_lng, ur_lat))

    # go through the list of files in the input path and filter for the geo location input

    # lets use the console progress bar to see our progress
    # get the number of files to process for the progress bar
    files = os.listdir(input_path)
    bar = ConsoleBar(len(files))
    for filename in files:
        # only use the "part* files"
        try:
            # read the file
            if filename[:4] == "part":
                # read the data
                df = read_data(os.path.join(input_path, filename))

                # get the coordinates from the data
                coords = get_preference(df)

                # filter for the des
                filtered = filter_bybox(coords, 'longitude', 'latitude', bounds)

                # save the result in the output folder
                out_name = filename + "_filtered.csv"
                filtered.to_csv(os.path.join(output_path, out_name))
                bar.tick()

        except Exception as ex:
            print(ex)
            print(os.path.join(current_loc, input_path, filename))
            break

if __name__ == "__main__":
    main()
```

Appendix 2 – Analysis code

The following are screen grabs from my project analysis notebook.

DATA7201 Project

The objective of this project is to use big data analytics techniques to explore the twitter dataset and draw some conclusions that inform decision makers.

Method

- Use Pig to extract hashtag and geo tag data to then map with weather data to see if people tag there tweets more during good weather
- Define good weather
- Work out how to group geo-tagged tweets by bounds (for grouping with weather)
- List hashtags used
- Count number of geo-tagged as a proportion of tweets in general

To access hdfs directory

> hdfs dfs -ls /data/ProjectDataset

Don't need to give an output filename, but you do need to provide a new folder name for pig to dump the results into, one file will be an empty file "_SUCCESS" and the other contains the data.

Save data here:\ hdfs://data7201-node1.fabric.zones.eait.uq.edu.au:8020/user/s4575121/project_output

To copy data from the DATA7201 node to our laptops:

- copy from HDFS to DATA7201 node:
  - > hdfs dfs -get hdfs_path/source_file_name target_file_name
- copy from DATA7201 node to remote labs (do from remote labs):
  - > scp data7201-8b39743e.zones.eait.uq.edu.au:/home/s4575121/file_name .
- then scp from the local machine from the zone (do from local machine):
  - > scp s4575121@remote.labs.eait.uq.edu.au:/home/students/s4575121/file_name "C:\Users\robmw\OneDrive\Documents\01 Master of Data Science\INFS7201\project\datasets"

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import seaborn as sns
import os
from IPython.core.display import display, HTML
# display(HTML("<style>.container { width:100% !important; }</style>"))
```

Read in the data

```
df = pd.read_csv('geo_tagged.csv', parse_dates=['time_stamp'])
df.head()
```

	id	time_stamp	longitude	latitude	type
0	483743110271205376	2014-06-30 22:45:19+00:00	-73.977142	40.780881	centroid
1	484021339431632896	2014-07-01 17:10:54+00:00	-70.778783	43.057659	centroid
2	484736120148598785	2014-07-03 16:31:11+00:00	-76.615181	39.396352	centroid
3	484752951890960384	2014-07-03 17:38:04+00:00	-73.977142	40.780881	centroid
4	484863140459741185	2014-07-04 00:55:55+00:00	-78.864466	44.035808	centroid

Summarise the dataset

```
print(len(df))
df['type'].value_counts()
```

```
1176197

point       1066878
centroid     109319
Name: type, dtype: int64
```

Plot the data in space

Use the CEA projection https://proj.org/operations/projections/cea.html

```
import reverse_geocoder as rg
df.head()
```

	id	time_stamp	longitude	latitude	type
0	483743110271205376	2014-06-30 22:45:19+00:00	-73.977142	40.780881	centroid
1	484021339431632896	2014-07-01 17:10:54+00:00	-70.778783	43.057659	centroid
2	484736120148598785	2014-07-03 16:31:11+00:00	-76.615181	39.396352	centroid
3	484752951890960384	2014-07-03 17:38:04+00:00	-73.977142	40.780881	centroid
4	484863140459741185	2014-07-04 00:55:55+00:00	-78.864466	44.035808	centroid

```
# get the citys and counties
def get_locations(df):
    # make a copy of the dataframe
    copy = df.copy()
    copy['coordinates'] = list(zip(copy['latitude'], copy['longitude']))

    # get the detail
    details = pd.DataFrame(rg.search(list(copy['coordinates'].values)))
    details.columns = ['city_lat', 'city_lon', 'city', 'state', 'county', 'cc']
    return pd.concat([copy, details], axis=1).drop(['coordinates'], axis=1)

df1 = get_locations(df)
df1.head()
```

```
Loading formatted geocoded file...
```

	id	time_stamp	longitude	latitude	type	city_lat	city_lon	city	state	county	cc
0	483743110271205376	2014-06-30 22:45:19+00:00	-73.977142	40.780881	centroid	40.78343	-73.96625	Manhattan	New York	New York County	US
1	484021339431632896	2014-07-01 17:10:54+00:00	-70.778783	43.057659	centroid	43.07176	-70.76255	Portsmouth	New Hampshire	Rockingham County	US
2	484736120148598785	2014-07-03 16:31:11+00:00	-76.615181	39.396352	centroid	39.4015	-76.60191	Towson	Maryland	Baltimore County	US
3	484752951890960384	2014-07-03 17:38:04+00:00	-73.977142	40.780881	centroid	40.78343	-73.96625	Manhattan	New York	New York County	US
4	484863140459741185	2014-07-04 00:55:55+00:00	-78.864466	44.035808	centroid	43.90012	-78.84957	Oshawa	Ontario		CA

Checkout popular locations

First plot the top 20 locations for tweets

```
# see what locations are popular
ny_counts = df1[df1['state']=='New York'].groupby(['state', 'city', 'city_lon', 'city_lat'])['id'].agg('count').sort_values(asce
nding=False)
ny_counts[:20]
```

```
state     city              city_lon   city_lat
New York  Manhattan         -73.96625  40.78343    22811
          New York City     -74.00597  40.71427    22238
          Long Island City  -73.94875  40.74482    12763
          Brooklyn          -73.94958  40.6501     11554
          Inwood            -73.9268   40.86566     8935
          The Bronx         -73.86641  40.84985     7897
          Borough of Queens -73.83652  40.68149     6531
          Morrisville       -75.64018  42.89868     6094
          East New York     -73.88236  40.66677     5031
          Bensonhurst       -73.99403  40.60177     4409
          Staten Island     -74.13986  40.56233     4207
          Jamaica           -73.80569  40.69149     3896
          Buffalo           -78.87837  42.88645     3269
          Rochester         -77.61556  43.15478     3162
          Syracuse          -76.14742  43.04812     2402
          Yonkers           -73.89875  40.93121     2311
          Kenmore           -78.87004  42.96589     2225
          Eggertsville      -78.80392  42.96339     2109
          Inwood            -73.7468   40.62205     1679
          Bloomfield        -74.1782   40.6126      1640
Name: id, dtype: int64
```

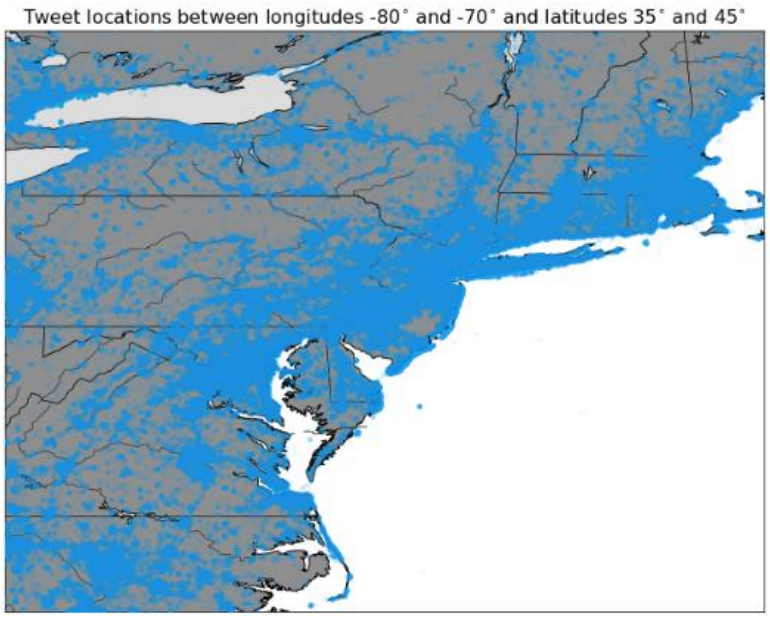
Looks like most are in New York City - so filter out the non-NYC area

```
# first clean the data
to_clean = ['state', 'city', 'county']
for col in to_clean:
    df1[col] = df1[col].str.strip()

# counties to keep
ny_counties = ['New York County', 'Queens County', 'Kings County', 'Richmond County', 'Nassau County', 'Suffolk County', '']
nyc = df1[(df1['state']=='New York') & (df1['county'].isin(ny_counties))]
nyc_counts = nyc[nyc['state']=='New York'].groupby(['state', 'city', 'county', 'city_lon', 'city_lat'])['id'].agg('count').sort_
values(ascending=False)
print("These are the top 10 locations within the area we want")
print(nyc_counts[:10])

# check what percent of tweets this now covers
print()
print(f"This now covers {sum(nyc_counts)/len(df):0.3%} of tweets")
```

```
These are the top 10 locations within the area we want
state     city               county           city_lon   city_lat
New York  Manhattan          New York County  -73.96625  40.78343    22811
          New York City                       -74.00597  40.71427    22238
          Long Island City   Queens County    -73.94875  40.74482    12763
          Brooklyn           Kings County     -73.94958  40.6501     11554
          Inwood             New York County  -73.9268   40.86566     8935
          Borough of Queens  Queens County    -73.83652  40.68149     6531
          East New York      Kings County     -73.88236  40.66677     5031
          Bensonhurst        Kings County     -73.99403  40.60177     4409
          Staten Island      Richmond County  -74.13986  40.56233     4207
          Jamaica            Queens County    -73.80569  40.69149     3896
Name: id, dtype: int64

This now covers 12.899% of tweets
```

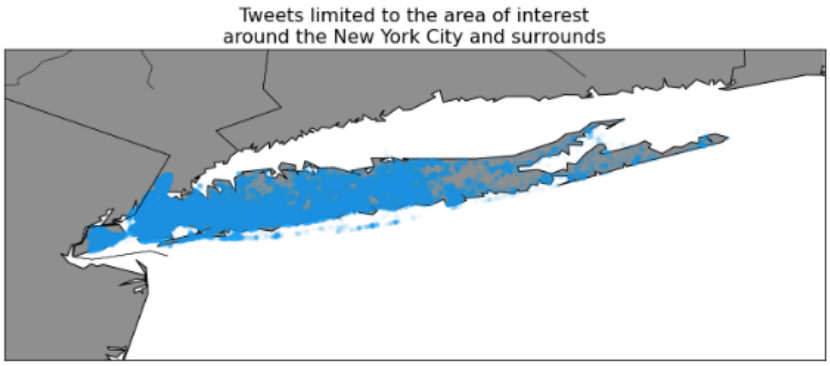
Plot all tweets from the broad filter

```
# plot all of the boxes
fig, ax = plt.subplots(figsize=[10, 12])
# sns.scatterplot(x='longitude', y='latitude', data=df, alpha=0.1)
# plt.ylim([35, 45])
# plt.xlim([-80, -70])
m = Basemap(llcrnrlon=-80, llcrnrlat=35, urcrnrlon=-70, urcrnrlat=45,
            resolution='i',projection='cea')
m.drawcoastlines()
m.fillcontinents(color='#606060',lake_color='white', alpha=0.7)
m.drawstates()
m.drawrivers()

# lets filter for the NY tweets only
x, y = m(df1['longitude'].values, df1['latitude'].values)
twitter_blue = (29/255, 161/255, 242/255)
# m.shadedrelief()
m.scatter(x, y, 10, marker='o', color=twitter_blue, zorder=999, alpha=0.05)
plt.title("Tweet locations between longitudes -80$^{\circ}$ and -70$^{\circ}$ and latitudes 35$^{\circ}$ and 45$^{\circ}$", font
size=16)
ax = plt.gca()
plt.tight_layout()
plt.savefig('map_all.png')
plt.show()
```



Plot the refined search

```
# plot all of the boxes
fig, ax = plt.subplots(figsize=[10, 12])
m = Basemap(llcrnrlon=-74.5, llcrnrlat=40, urcrnrlon=-71.5, urcrnrlat=41.5,
            resolution='i',projection='cea')
m.drawcoastlines()
m.fillcontinents(color='#606060',lake_color='white', alpha=0.7)
m.drawstates(zorder=999)
m.drawrivers()

# lets filter for the NY tweets only
x, y = m(nyc['longitude'].values, nyc['latitude'].values)
twitter_blue = (29/255, 161/255, 242/255)
# m.shadedrelief()
m.scatter(x, y, 10, marker='o', color=twitter_blue, zorder=99, alpha=0.05)
ax = plt.gca()
plt.title("Tweets limited to the area of interest\naround the New York City and surrounds", fontsize=16)
plt.tight_layout()
plt.savefig('local_map.png')
plt.show()
```



Plot the data in time

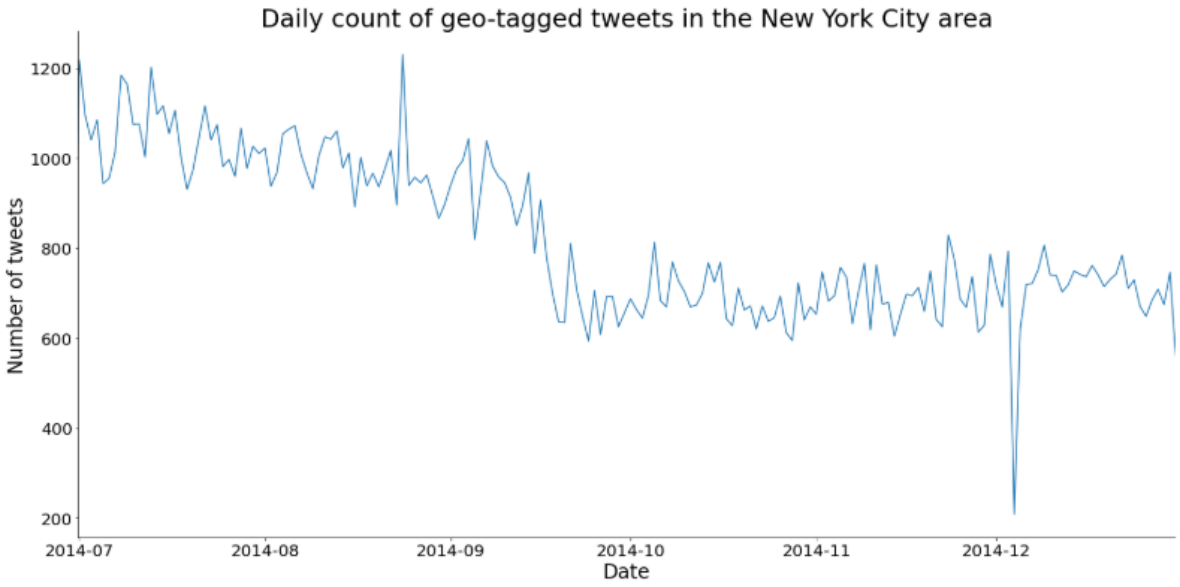
Set the time_stamp as a datetime index

Tasks

- adjust for the timezone
- get the weather data and plot this on top of the tweet data

```
# create date time index and convert to us timezone
time_index = pd.DatetimeIndex(nyc['time_stamp'])
us_index = time_index.tz_convert('US/Eastern')
time_df = nyc.set_index(us_index).sort_index()

# count the number of tweets per day
counts = time_df['id'].resample('1D').count()

# plot the counts over the timeseries
fig, ax = plt.subplots(figsize=[20, 10])
sns.lineplot(counts.index, counts.values)
plt.xlim([pd.to_datetime('2014-07-1'), pd.to_datetime('2014-12-31')])
ax.tick_params(axis='both', which='major', labelsize=16)

# formatting
axis_fs = 24
ax.set_xlabel('Date', fontsize=axis_fs)
ax.set_ylabel('Number of tweets', fontsize=axis_fs)
ax.set_title("Daily count of geo-tagged tweets in the New York City area", fontsize=30)
ax.tick_params(axis='both', which='major', labelsize=20, reset=False)
sns.despine()
plt.tight_layout()
plt.savefig('daily_counts.png')
print(f"The highest and lowest days are {counts.idxmax().strftime('%d-%m-%y')} and {counts.idxmin().strftime('%d-%m-%y')}")
```

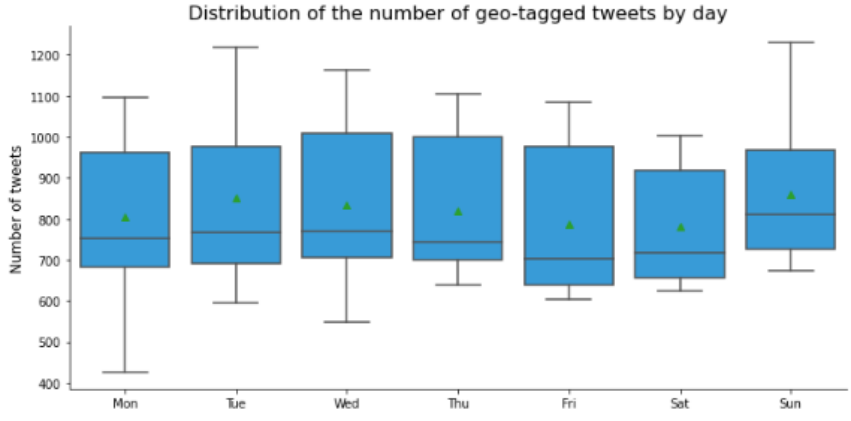
```
The highest and lowest days are 24-08-14 and 04-12-14
```



Explore the twitter and weather data

- Look at the daily distribution

```
# Lets have a look at the days for the week
cdf = counts.reset_index()
# cdf.head()
cdf['dayofweek'] = cdf['time_stamp'].dt.dayofweek
# cdf.head()

# plot the distribution of tweets over the week
fig, ax = plt.subplots(figsize=[10, 5])
sns.boxplot(x='dayofweek', y='id', data=cdf, color=twitter_blue, showfliers=False, showmeans=True)

# set the xticks to days
xticks = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
ax.set_xlabel(None)
ax.set_ylabel('Number of tweets', fontsize=12)
ax.set_title("Distribution of the number of geo-tagged tweets by day", fontsize=16)
ax.set_xticklabels(xticks)
ax.tick_params(axis='both', which='major', labelsize=10, reset=False)
sns.despine()
plt.tight_layout()
plt.savefig('box_tweets.png')
```



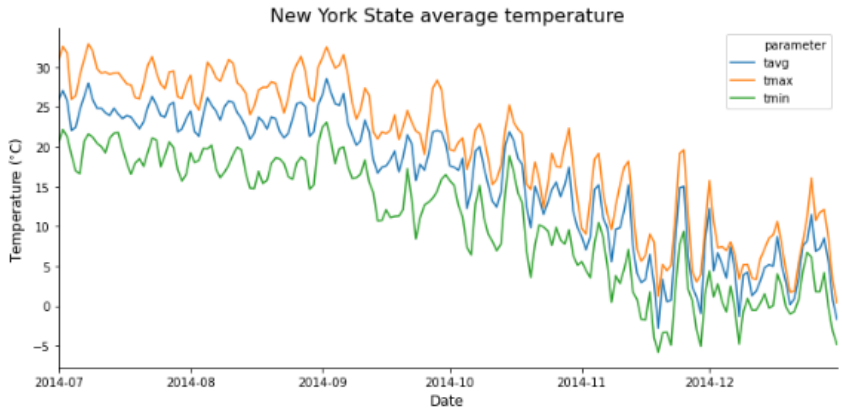
Weather data

We have the following fields:

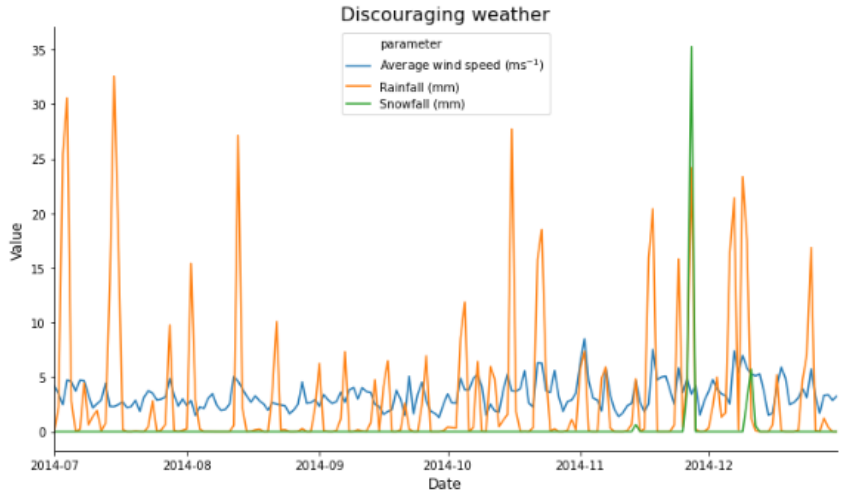
- SNOW: mm or inches of snowfall
- SNWD: mm or inches of snowdepth
- PRCP: mm or inches of rain
- AWND: average daily windspeed ms^{-1} or mph
- TMIN: min temp in f
- TAVG: average temp in f
- TMAX: max temp in f

```
# get the weather data
weather = pd.read_csv(r'datasets\weather.csv', parse_dates=['DATE'], dayfirst=True)
weather.columns = [col.lower() for col in weather.columns]
to_convert = ['tmin', 'tavg', 'tmax']

# print the number of stations
stations = weather['station'].unique()
print(f'these are the {len(stations)} stations:')
print(stations)

# convert temp function
f2c = lambda f: (f-32)*5/9

# melt the temp data for plotting
id_cols = ['station', 'name', 'date']
temps = weather.loc[weather['tmin'].notnull(), id_cols + to_convert]
temps = pd.melt(temps, id_vars=id_cols, value_vars=temps.columns[3:], value_name='value', var_name='parameter')

# group the by parameter and take the average of each parameter
temps['celsius'] = temps['value'].map(f2c)
temps = temps.groupby(['date','parameter'])['celsius'].agg(np.mean).reset_index()
temps = temps[temps['date'] >= pd.to_datetime('2014-7-1')]

# plot the results
fig, ax = plt.subplots(figsize=[10, 5])
sns.lineplot(x='date', y='celsius', hue='parameter', data=temps)
plt.title("New York State average temperature", fontsize=16)
ax.set_xlabel('Date', fontsize=12)
ax.set_ylabel('Temperature ($^{\circ}$C)', fontsize=12)
plt.xlim([pd.to_datetime('2014-07-1'), pd.to_datetime('2014-12-31')])
ax.tick_params(axis='both', which='major', labelsize=10, reset=False)
sns.despine()
plt.tight_layout()
plt.savefig('all_temps.png')
```

```
these are the 89 stations:
['US1NYWC0003' 'US1NJBG0023' 'US1NJES0011' 'USC00280907' 'US1NJES0015'
 'US1NJMD0043' 'US1NJMS0069' 'US1NJPS0005' 'US1NJBG0029' 'US1NJPS0004'
 'US1NJMN0048' 'US1NJMN0006' 'US1NJMD0045' 'US1NJUN0019' 'US1NJUN0018'
 'USW00014732' 'US1NJUN0017' 'US1NJMD0050' 'US1NJUN0016' 'USW00014734'
 'US1NJUN0014' 'US1NYNS0006' 'US1NYNS0007' 'US1NJUN0010' 'US1NYWC0009'
 'US1NJES0020' 'US1NJMD0039' 'US1NJBG0033' 'US1NJES0021' 'US1NJMD0038'
 'US1NJES0024' 'US1NJPS0019' 'US1NJBG0037' 'US1NJHD0002' 'US1NJMS0014'
 'US1NJMS0058' 'US1NJMS0059' 'US1NJPS0015' 'US1NJPS0018' 'US1NJPS0012'
 'US1NYRC0002' 'US1NJMS0011' 'US1NYRC0001' 'US1NJPS0014' 'US1NJMD0033'
 'US1NJMN0032' 'USW00094741' 'US1NJES0018' 'US1NYRL0005' 'US1NJES0019'
 'USW00094745' 'USW00094789' 'USC00289187' 'US1NJBG0030' 'US1NJBG0002'
 'US1NJBG0003' 'US1NJMS0049' 'US1NJBG0006' 'US1NJMS0047' 'US1NJMD0062'
 'US1NJMS0002' 'US1NJPS0022' 'US1NJMD0023' 'US1NJPS0025' 'US1NYKN0025'
 'US1NJMS0040' 'USC00282023' 'USC00301309' 'USC00281335' 'US1NJBG0010'
 'US1NJBG0015' 'US1NJMS0039' 'US1NJBG0017' 'US1NJMS0036' 'US1NJBG0018'
 'USC00283704' 'US1NJMD0058' 'US1NJMD0055' 'US1NJMS0075' 'USW00094728'
 'US1NJMN0051' 'US1NJMS0070' 'US1NJUN0007' 'US1NYQN0002' 'US1NJMD0060'
 'US1NJMN0010' 'US1NJMN0011' 'USW00054743' 'USW00054787']
```



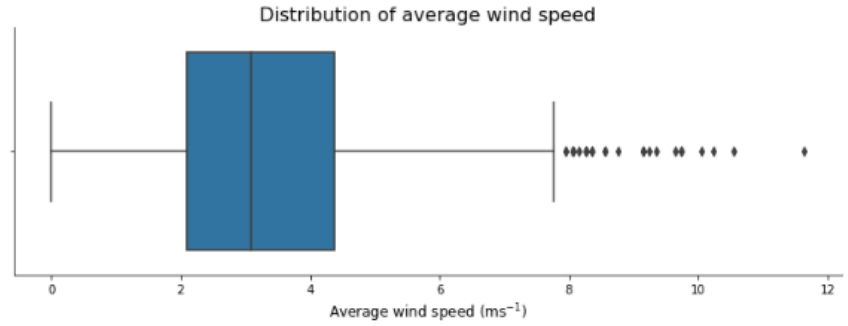
```
# Let's look at bad stuff that would keep people at home, like:
# - snow, rain and wind
bad_stuff = ['snow', 'prcp', 'awnd']
bsdf = weather.copy()

# conversions
inch2mm = lambda x: 25.4*x
mph2mps = lambda x: 1.6*x/3.6
bsdf['awnd'] = bsdf['awnd'].map(mph2mps)
to_convert = ['snow', 'snwd', 'prcp']
for col in to_convert:
    bsdf[col] = bsdf[col].map(inch2mm)

# melt columns for plotting
id_cols = ['station', 'name', 'date']
bs = bsdf.loc[:, id_cols + bad_stuff]
bs = pd.melt(bs, id_vars=id_cols, value_vars=bs.columns[3:], value_name='value', var_name='parameter').dropna()

# take the average of the values
bs = bs.groupby(['date', 'parameter'])['value'].agg(np.mean).reset_index()

# plot the results
fig, ax = plt.subplots(figsize=[10, 6])

# let's relable for the crowd
to_plot = bs.copy()
relabel = dict(awnd='Average wind speed (ms$^{-1}$)',
               prcp='Rainfall (mm)',
               snow='Snowfall (mm)')
to_plot['parameter'] = to_plot['parameter'].map(relabel)
sns.lineplot(x='date', y='value', hue='parameter', data=to_plot)

# plot formats
plt.title("Discouraging weather", fontsize=16)
ax.set_xlabel('Date', fontsize=12)
ax.set_ylabel('Value', fontsize=12)
plt.xlim([pd.to_datetime('2014-07-1'), pd.to_datetime('2014-12-31')])
ax.tick_params(axis='both', which='major', labelsize=10, reset=False)
sns.despine()
plt.tight_layout()
plt.savefig('all_bad.png')
```



Convert the bad stuff to a categorical for each day

- wind above the 75th percentile is windy
- rain above 5mm is rainy
- snow above 5mm is snowy

```
# check windspeed and take the upper percentiles as windy
fig, ax = plt.subplots(figsize=[10, 4])
sns.boxplot(bsdf['awnd'])

# use anything above the 75th percentile as windy
np.percentile(bsdf['awnd'].dropna().values, 75)

# plot formats
plt.title("Distribution of average wind speed", fontsize=16)
ax.set_xlabel('Average wind speed (ms$^{-1}$)', fontsize=12)
ax.tick_params(axis='both', which='major', labelsize=10, reset=False)
sns.despine()
plt.tight_layout()
plt.savefig('wind.png')
```



Check for relationships between the data

Use some scatterplots

```
def show_range(label, index):
    print(f"{label} (n={len(index)}): minimum date {min(counts.index).strftime('%d/%m/%y')}, maximum date {max(counts.index).str
ftime('%d/%m/%y')}")
```

```
counts.head()
```

```
time_stamp
2014-06-30 00:00:00-04:00     426
2014-07-01 00:00:00-04:00    1218
2014-07-02 00:00:00-04:00    1095
2014-07-03 00:00:00-04:00    1040
2014-07-04 00:00:00-04:00    1085
Freq: D, Name: id, dtype: int64
```

```
# check we have the same index for the weather and count values
show_range('Counts', counts.index)
show_range('Temperature', temps['date'])
show_range('Bad stuff', bs['date'])
```

```
Counts (n=185): minimum date 30/06/14, maximum date 31/12/14
Temperature (n=552): minimum date 30/06/14, maximum date 31/12/14
Bad stuff (n=552): minimum date 30/06/14, maximum date 31/12/14
```

```
for t in ['tmin', 'tavg', 'tmax']:
    show_range(t, temps[temps['parameter']==t])

for b in ['snow', 'awnd', 'prcp']:
    show_range(b, bs[bs['parameter']==b])
```

```
tmin (n=184): minimum date 30/06/14, maximum date 31/12/14
tavg (n=184): minimum date 30/06/14, maximum date 31/12/14
tmax (n=184): minimum date 30/06/14, maximum date 31/12/14
snow (n=184): minimum date 30/06/14, maximum date 31/12/14
awnd (n=184): minimum date 30/06/14, maximum date 31/12/14
prcp (n=184): minimum date 30/06/14, maximum date 31/12/14
```

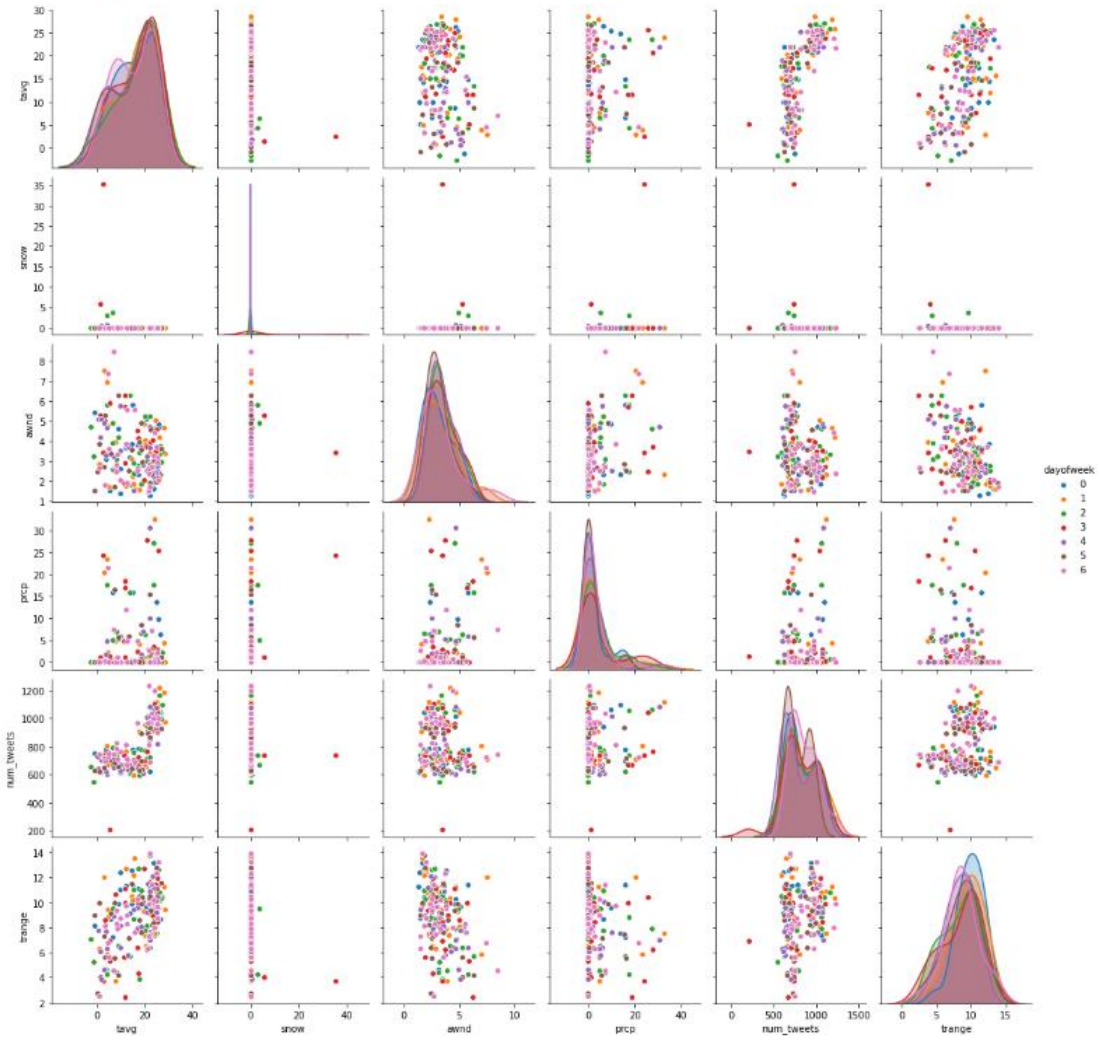
Creat a single dataset

```
def get_variable(df, param, vcol):
    return df[df['parameter']==param].set_index(pd.DatetimeIndex(df[df['parameter']==param]['date'], tz=timezone('US/Eastern')))
[vcol].rename(param)
```

```
# need to set the timezones
from pytz import timezone
variables = [get_variable(temps, 'tmin', 'celsius'),
             get_variable(temps, 'tavg', 'celsius'),
             get_variable(temps, 'tmax', 'celsius'),
             get_variable(bs, 'snow', 'value'),
             get_variable(bs, 'awnd', 'value'),
             get_variable(bs, 'prcp', 'value'),
             counts[1:]]
mdf = pd.concat(variables, axis=1).rename(columns=dict(id='num_tweets'))

# include day of the week and temp range, drop the min-max temps and the day of week
mdf['dayofweek'] = mdf.index.dayofweek
mdf['trange'] = mdf['tmax'] - mdf['tmin']
mdf = mdf.drop(['tmax', 'tmin'], axis=1)
mdf.head()
```

	tavg	snow	awnd	prcp	num_tweets	dayofweek	trange
2014-07-01 00:00:00-04:00	25.925926	0.0	4.187778	0.000000	1218	1	10.341880
2014-07-02 00:00:00-04:00	27.037037	0.0	3.454444	2.256367	1095	2	10.427350
2014-07-03 00:00:00-04:00	25.740741	0.0	2.472778	25.459765	1040	3	10.470085
2014-07-04 00:00:00-04:00	22.037037	0.0	4.710000	30.550338	1085	4	6.965812
2014-07-05 00:00:00-04:00	22.407407	0.0	4.573889	2.819400	943	5	9.401709

```
sns.pairplot(mdf, hue='dayofweek')
```



Discussion

I can't see any obvious relations with tweet volume other than the temperature and in the line plot there seems to be a shift from the warmer to colder weather

Model the response

Try predicting the response with a bunch of models

tasks:

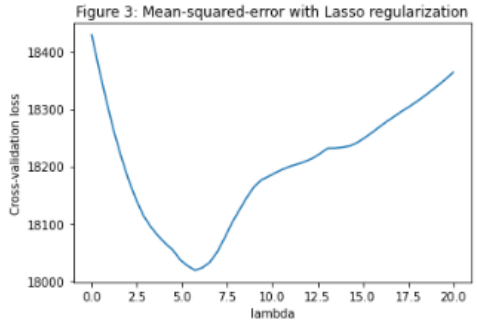
- scale the data
- use regularisation
- turn rain and snow into cats

```
# get the some modelling tools
from sklearn.linear_model import Lasso
from sklearn.model_selection import KFold
import sklearn.metrics as metrics
from sklearn import preprocessing

# lets use our work from Assingment 3 to see if any models can predict tweet volume from the data given to us?
def validate(X, y, model, verbose=False):
    """
    Calculate the ten-folder cross-validation loss
    """
    # instantiate kfold
    kf = KFold(n_splits=10)

    # split the data
    kf.get_n_splits(X)

    # loop through and calculate the loss
    losses = list()
    for train_index, test_index, in kf.split(X):
        # index the training and test sets
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # train a linear model on the test set
        model.fit(X_train, y_train)

        # make a prediction
        y_predict = model.predict(X_test)

        # score the loss
        loss = metrics.mean_squared_error(y_test, y_predict)
        losses.append(loss)

    # take the average of the losses
    mse = np.mean(losses)
    if verbose:
        print(f"Cross-validation error: {mse:0.0f}")
    return mse

def process_data(df, to_cat, response):
    '''
    categorise and scale the data
    return two arrays in a tuple (X, y)
    '''
    # get the predictors
    pred = df.copy().drop([response], axis=1)

    # categories to ones and zeros using a threshold
    for cat in to_cat:
        pred[cat[0]] = pred[cat[0]].map(lambda x: 0 if x < cat[1] else 1)

    # scale the rest
    return preprocessing.scale(pred.values), df[response].values
```

```
def lasso(X, y, lambda_lims, max_iter, verbose=False):
    """
    Find the best lambda for Lasso regularization
    """
    # get a range of lambdas
    lambdas = np.linspace(lambda_lims[0], lambda_lims[1], endpoint=True)

    # loop through the lambdas and validate each
    losses = list()
    for l in lambdas:
        # print the lambda
        if verbose:
            print(l)

        # instantiate a model
        model = Lasso(alpha=l, max_iter=max_iter)

        # validate the model
        loss = validate(X, y, model)
        losses.append(loss)

    # plot the result
    fig, ax = plt.subplots()
    plt.plot(lambdas, losses)
    plt.title("Figure 3: Mean-squared-error with Lasso regularization")
    plt.xlabel("lambda")
    plt.ylabel("Cross-validation loss")

    # print the best lambda
    best_lambda = lambdas[np.argmin(losses)]
    print(f"Applying Lasso regression with lambda of {best_lambda:0.1f} minimises the loss to {min(losses):0.0f}")
```

```
# preprocess the data
X, y = process_data(mdf, [('snow', 5), ('prcp', 10), ('awnd', 4.5)], 'num_tweets')

# have a go at Lasso regression
lasso(X, y, lambda_lims=[0.001, 20], max_iter=100000)
```

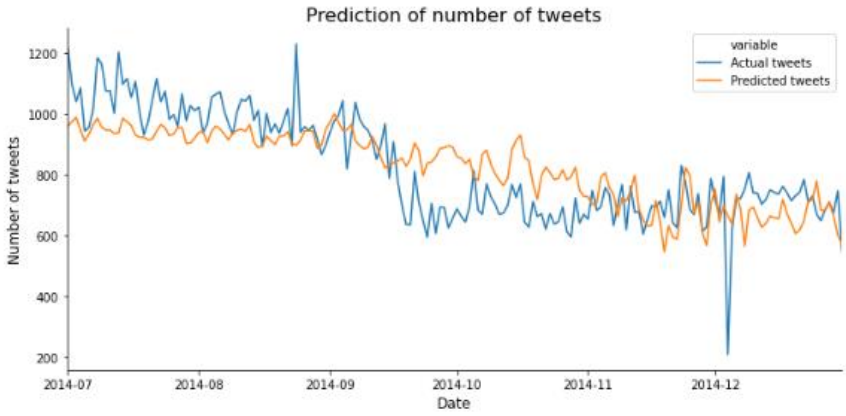
```
Applying Lasso regression with lambda of 5.7 minimises the loss to 18019
```



Lasso regression with a lambda of 5.7 works best, how does in perform?

```
model = Lasso(alpha=5.7, max_iter=1000)
model.fit(X, y)
prediction = model.predict(X)

# lets have a look at how the prediction tracks with the actual values
mdf['predict'] = prediction
to_plot = pd.melt(mdf.loc[:, ['num_tweets', 'predict']].reset_index(), id_vars=['index'], value_vars=['num_tweets', 'predict'],
value_name='count')

# plot the result
relabel = dict(num_tweets='Actual tweets',
               predict='Predicted tweets')
#               snow='Snowfall (mm)')
to_plot['variable'] = to_plot['variable'].map(relabel)
fig, ax = plt.subplots(figsize=[10, 5])
sns.lineplot(x='index', y='count', hue='variable', data=to_plot)
to_plot.head()

# plot formats
plt.title("Prediction of number of tweets", fontsize=16)
ax.set_xlabel('Date', fontsize=12)
ax.set_ylabel('Number of tweets', fontsize=12)
plt.xlim([pd.to_datetime('2014-07-1'), pd.to_datetime('2014-12-31')])
ax.tick_params(axis='both', which='major', labelsize=10, reset=False)
sns.despine()
plt.tight_layout()
plt.savefig('prediction.png')
print(f"The the highest and lowest predicted dates were {mdf['predict'].idxmax().strftime('%d-%m-%y')} and {mdf['predict'].idxmi
n().strftime('%d-%m-%y')}")
```

```
The the highest and lowest predicted dates were 02-09-14 and 19-11-14
```

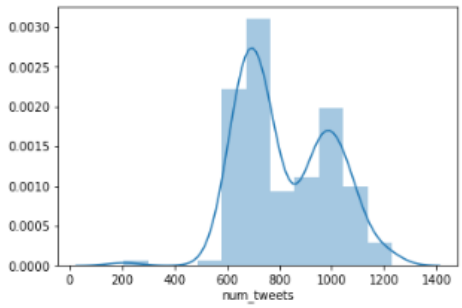



```
print(model.intercept_)
print(model.coef_)
print(model.score(X, y))
```

```
822.2391304347826
[123.67284702   4.24455132   0.          10.41053676   0.
  -8.92394831]
0.5325145141208144
```

```
residuals = mdf['predict'] = mdf['num_tweets']
sns.distplot(residuals)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20415634448>
```



Modelling

But we haven't tested it on new data - this could be a next step

What's going on on the outlier days?

Get the hashtags from our region of interest on the highest and lowest days to see what people are talking about

```
# get the hash tag data
all_data = []
for file in os.listdir('hash_tags'):
    temp = pd.read_csv(os.path.join('hash_tags', file))
    all_data.append(temp)
tags_raw = pd.concat(all_data).reset_index(drop=True)
print(len(tags_raw))
print(tags_raw.head())

# set the timezone and the locations and then filter like with the above datasets
tags_raw = tags_raw.set_index(pd.DatetimeIndex(tags_raw['time_stamp']).tz_convert('US/Eastern'), drop=True)
tags_raw['date'] = tags_raw.index.date
# tags_raw.head()

# filter for the locations
with_locs = get_locations(tags_raw.reset_index(drop=True))
filtered = with_locs[with_locs['county'].isin(ny_counties)]
filtered.head()
```

```
51793
                  id               time_stamp  \
0  483746608329146368  2014-06-30T22:59:13.000Z
1  483750991372623874  2014-06-30T23:16:38.000Z
2  483757844882530304  2014-06-30T23:43:52.000Z
3  483760449532338176  2014-06-30T23:54:13.000Z
4  483761519097032705  2014-06-30T23:58:28.000Z

                              hashtags  longitude   latitude      type
0            ({(maverickradio),(hotmess)}) -73.873522  40.728526  centroid
1                    ({(HobbyLobby)}) -73.977142  40.780881  centroid
2                ({(european),(antibes)}) -73.873522  40.728526  centroid
3                  ({(Pride),(NYCPride)}) -73.977142  40.780881  centroid
4  ({(CT),(FamousPizza),(SoNo),(Hometown)}) -73.199119  41.185674  centroid
```

	id	time_stamp	hashtags	longitude	latitude	type	date	city_lat	city_lon	city	state	county	cc
0	483746608329146368	2014-06-30T22:59:13.000Z	({(maverickradio),(hotmess)})	-73.873522	40.728526	centroid	2014-06-30	40.68149	-73.83652	Borough of Queens	New York	Queens County	US
1	483750991372623874	2014-06-30T23:16:38.000Z	({(HobbyLobby)})	-73.977142	40.780881	centroid	2014-06-30	40.78343	-73.96625	Manhattan	New York	New York County	US
2	483757844882530304	2014-06-30T23:43:52.000Z	({(european),(antibes)})	-73.873522	40.728526	centroid	2014-06-30	40.68149	-73.83652	Borough of Queens	New York	Queens County	US
3	483760449532338176	2014-06-30T23:54:13.000Z	({(Pride),(NYCPride)})	-73.977142	40.780881	centroid	2014-06-30	40.78343	-73.96625	Manhattan	New York	New York County	US
5	483771073725739008	2014-07-01T00:36:26.000Z	({(YorkRevs)})	-73.873522	40.728526	centroid	2014-06-30	40.68149	-73.83652	Borough of Queens	New York	Queens County	US

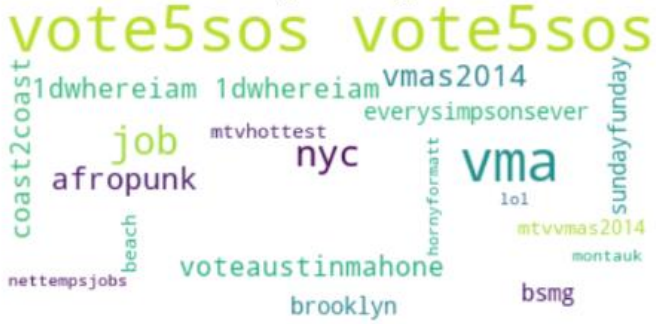
```python
def get_datewords(date, date_col, word_col, df):
    '''
    extract the outlier words from the data
    '''

    # filter by date
    by_date = df[df[date_col]==date]

    # remove the parenthesis from the words
    words = by_date[word_col].str.replace(r"[({})\s]", "").str.lower().str.split(",")

    all_words = list()
    for tags in words:
        all_words += [t.strip() for t in tags]
    return all_words
```

```python
from wordcloud import WordCloud
```

```python
# filter for the outlier dates
low_date = pd.to_datetime('2014-12-04')
high_date = pd.to_datetime('2014-08-24')
pred_low = pd.to_datetime('2014-11-19')
pred_high = pd.to_datetime('2014-09-02')
outliers = [low_date, high_date, pred_low, pred_high]
labels = ['Lowest day', 'Highest day', 'Lowest predicted day', 'Highest predicted day']
words = list()
for label, date in zip(labels, outliers):
    words = get_datewords(date, 'date', 'hashtags', filtered)
    wordcloud = WordCloud(max_font_size=50, max_words=20, background_color="white").generate(" ".join(words))
    plt.figure(figsize=[10, 6])
    plt.title(label, fontsize=24)
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.savefig(f"{label}.png")
    plt.show()
```

Lowest day

Highest day



Lowest predicted day

Highest predicted day