Analysing and Benchmarking Customer Service performance Indices from Twitter data

[Vivekanand Praturi]
[John]

Professor: Dr. Jaeki. Song

This proposal is submitted in partial fulfilment for the degree of Master of Science in Data Science.

TEXAS TECH UNIVERSITY
Rawls College *of* Business

Executive Summary

With more than 500 million tweets, over two billion searches a day we have an active audience to engage with. In a recent study, it is estimated that over 34% of the companies have dedicated twitter account for customer service. Low Customer satisfaction in turn halts the business growth and gives the competitors a chance to increase their presence. To optimize Customer Support team's performance, data analysis is required to find out how the support team's performance. As competition grows in a flat-lining global economy, information will be power to understand to help management make changes to their core business.

This project can be divided into three phases. The first phase involved collecting data in real time and storing it. The second phase involved processing, analysis of data, which can helps us to mine interesting patterns that can evaluate current performance and drive the future of business. The final phase is visualizing interesting trends and patterns from the data. We tried to incorporate features that can export graphs and the data of the graph in different formats, which help further investigation.

Table of contents

List of Figures:

1. INTRODUCTION

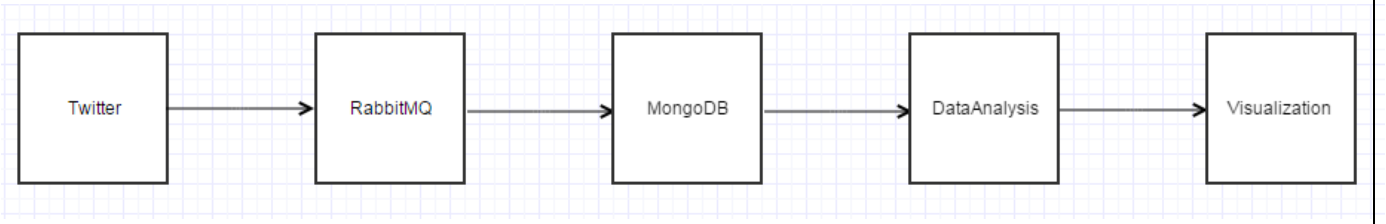
Twitter has become major platform for information sharing. Applications which use twitter data to mine interesting patterns can be help us in driving and evaluating business decisions based on public opinion. Customer Service through social media has become relevant these days with the advent of social networking sites like Twitter and Facebook. In a recent study it is estimated that over 34% of the companies have dedicated twitter account for customer service.

The objective of the project is to analyze and quantify the performance of customer service teams in providing support to customer. Evaluating and optimizing performance of Customer Service is essential for the growth of any business. Evaluating performance requires relevant metrics which can give some useful insights to take correct decisions which benefit the business. Our project aims in analyzing the data from twitter and to get relevant metrics to evaluate and benchmark the performance of Customer Service by providing access to a web application which is a dashboard with visualizations.

2. OVERVIEW OR BACKGROUND

This project helps in collecting data in real time and in understanding and visualizing interesting trends and patterns from the data, which can potentially help in evaluating and driving new business strategies. As competition grows in a flat-lining global economy, information will be power to understand to help management make changes to their core business. To optimize Customer Service, data analysis is required to find out relevant metrics like daily customer request trend and the average response time in providing service to the customer.

3. ARCHITECTURE



4. DATA SOURCE AND DATA STORAGE

Twitter has provided APIs for developers to stream data. The streams of Json responses we receive must be parsed for required information and stored in the data base for further analysis. The rate at which the response is received is not always equal to the insert rate in the database. So overcome this problem we have used a Publish Subscribe model to store the real time data.

RABBITMQ:

We used RabbitMQ which is an open source message broker middleware that implements the Advanced Message Queuing Protocol (AMQP). The principal idea is pretty simple: it accepts and forwards messages. Producing means nothing more than sending. A program that sends messages

is a producer. A queue stores the messages. Although messages flow through RabbitMQ and its applications, they can be stored only inside a queue.

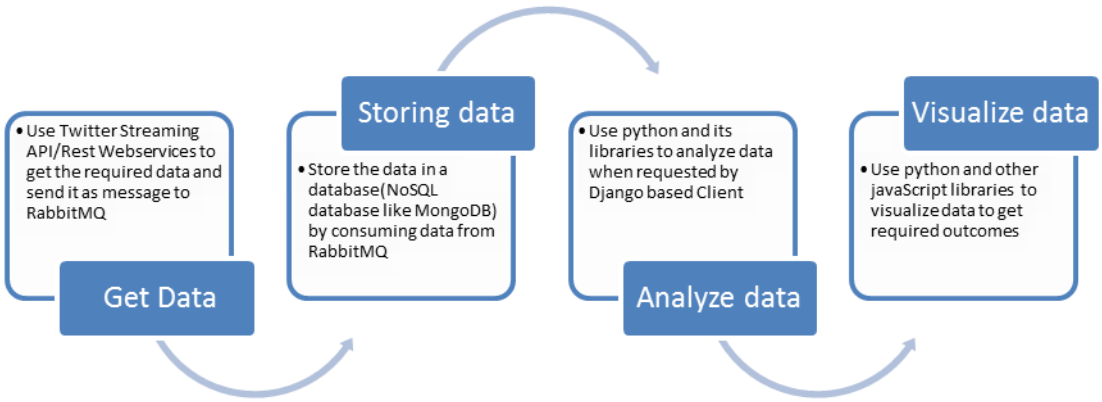
A queue is not bound by any limits, it can store as many messages as you like – it's essentially an infinite buffer. Many producers can send messages that go to one queue, many consumers can try to receive data from one queue. Consuming has a similar meaning to receiving. A consumer is a program that mostly waits to receive messages.

Persistent NoSQL Data Store:

MongoDB (from humongous) is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster.

4.1 IMPLEMENTATION STEPS

FIGURE 1: FLOW DIAGRAM



Step 1 :

The first step is the registration of the application with twitter. Once its done we receive a consumer key and aconsumer secret: these are application settings that should always be kept private. From the configuration page of your app, you can also require an access token and an access token secret. Similarly to the consumer keys, these strings must also be kept private: they provide the application access to Twitter on behalf of your account.

Using tweepy, a python package, we get the tweets from twitter as a Json response. The response has all the tweet related data and metadata. We should parse the Json and store the fields that really matter to us. So we post the Json to RabbitMQ queue which is configured in the python script (using Pika package to connect to RabbitMQ server).

Step 2:

A consumer (python script) reads the messages from the queue and using Pymongo package it connects to the MongoDB server and stores the tweets in the collections in real time. Once the data gets stored in real time in the NoSQL datastore we can take advantage of the data, analyse it, and display the findings in real time.

Step 3:

The Django web application hosted on webserver and it gives an interface to the user to log in and access the application to view the analytics dashboard. Django is a free and open source web application framework, written in Python, which follows the model–view–controller (MVC) architectural pattern.

Model - The lowest level of the pattern which is responsible for maintaining data. View - This is responsible for displaying all or a portion of the data to the user. Controller - Software Code that controls the interactions between the Model and View

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response.

Step 4:

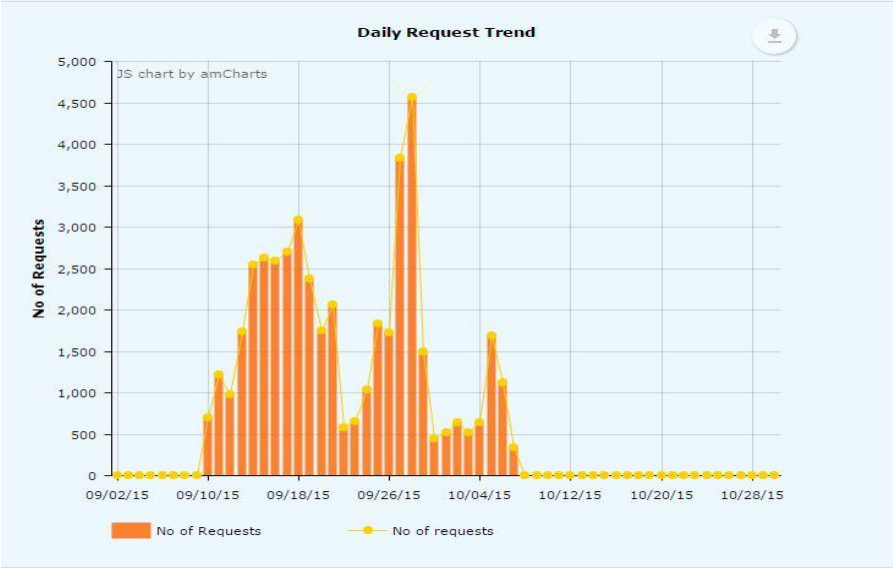
Visualizations:

1) **Daily Customer Service Request Trend** :

Use: This Plot shows the request trend per day. We find peaks on certain days that can account to some technical problems faced on particular day. This gives a heads up to start investigating upon issues on specific days.

The plot shows the daily customer service request trend of Delta Assist service handle.

Figure 2: Daily Request Trend Chart

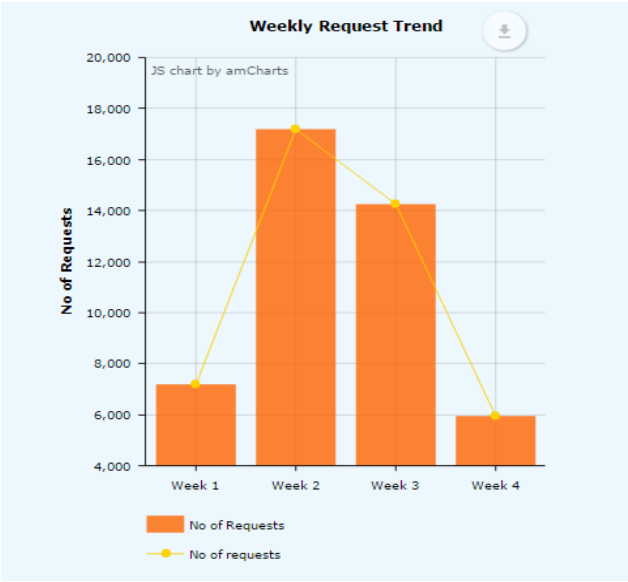


2) **Weekly Request Trend** :

Use: This Plot shows the request trend per week. We get trend over a month on weekly basis to see if there is a similar pattern over different months.

The plot shows the weekly customer service request trend of Delta Assist service handle.

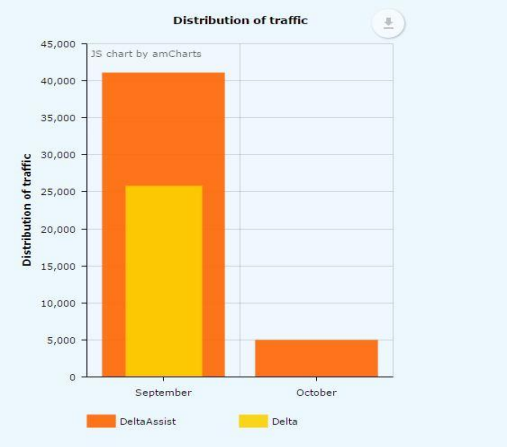
Figure 3: Weekly Request Trend



3) **Distribuiton of Traffic to different Customer Service handles:**

Use: Almost every product has more than one twitter customer Service handle. By measuring customer service activity across multiple accounts, we can determine the number of requests per handle and the resource allocation for support teams.

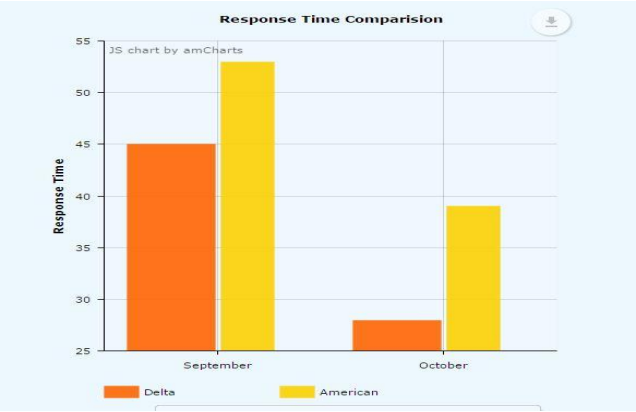
Figure 4: Distribution of Traffic



4) **Response time**: Response time is Time of @Reply to inbound - Tweet Time of inbound Tweet.

Use: The response time metric is useful to benchmark the time taken by the customer support team to respond. Quick response times and a high response rate are key to optimizing performance. Brands can identify a target response time and set targets for improvement by monitoring their competitors.

Figure 5: Response Time



5) **Customer Satisfaction Index**:

Use:

- Customer Satisfaction is something that is very important for any business. Constant monitoring is required and appropriate action needs to be taken to improve the customer experience.

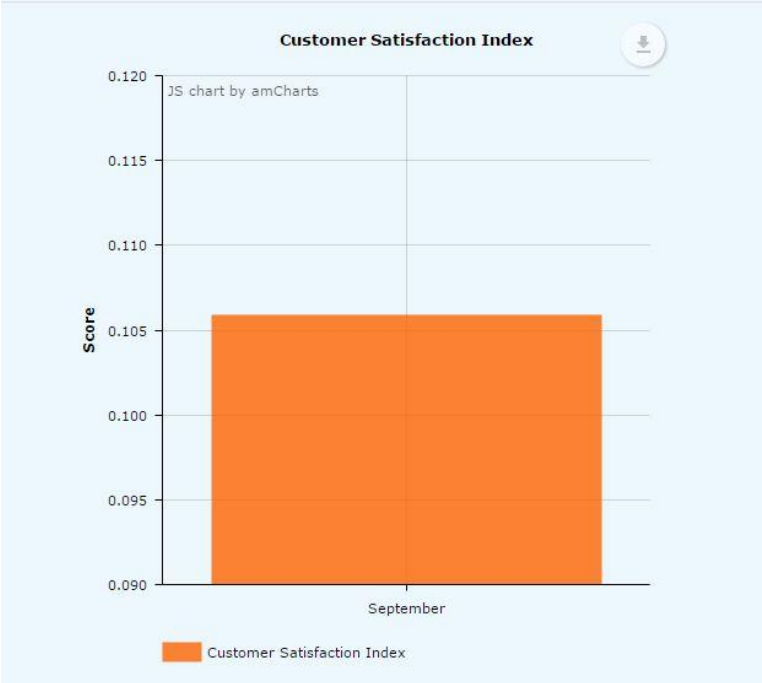
- To calculate the customer satisfaction index, the text from the tweet is extracted and then analysed tested against a set of test data which are categorised based on the word into Good, Bad etc., the range of scores is from -1 to 1.

Here is an example of the sentiment analysis:

```
>>> testimonial = TextBlob("Textblob is amazingly simple to use. What great fun!")
>>> testimonial.sentiment.polarity
0.39166666666666666
```

The polarity score is a float value that helps measure the negativity vs positivity and fall at -1 for a perfectly negative phrase and a perfectly positive phrase value of 1.

Figure 6: Customer Satisfaction Index



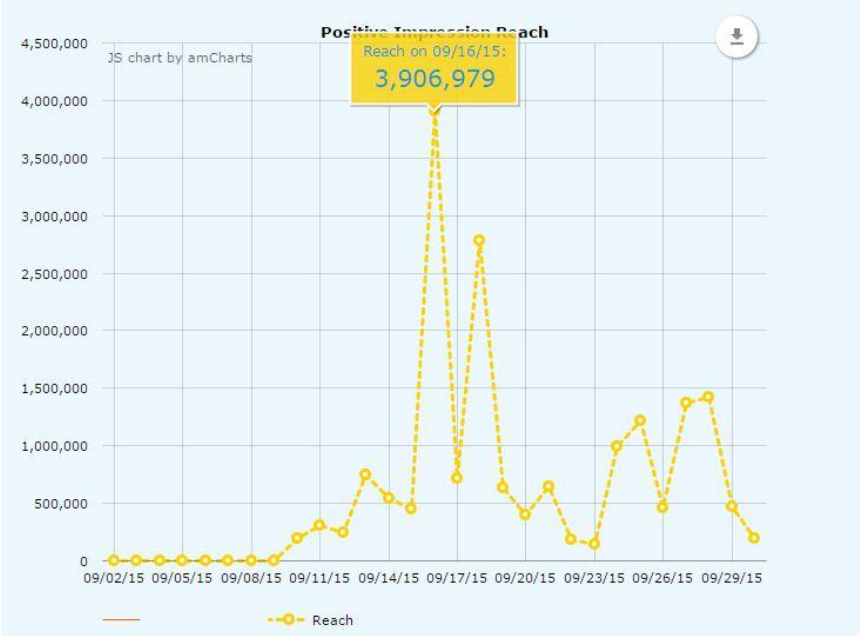
From the graph we can infer that there is a slight positive Average score for the month of September.This score can be evaluated over a period of time and when this is used with some other parameters we can gain some useful insights.

6) Positive Impression Reach:

Use: Every company is curious to understand the reach of positive impression about their product or brand among the people. Due to the spread of positive impression can bring new customers and retain old customers.

Positive Impression reach is calculated by getting the total count of all the users on twitter who can potentially see a positive tweet about Delta Airlines.

Figure 7: Positive Impression Reach



From the graph we can see that the postive impression about Delta Airlines reached to almost 4 million customers on 16th September.The zero values are not due absense of data for those days which can be ignored in this plot.

Other Features:

- We can download the graph in Jpg, Png, Svg, Pdf formats.
- We can save the data of the graph in Csv, Xlxs, Json formats
- We can print the individual graphs from the menu shown below.
- We can annotate the graphs and point out the findings using pencil and draw shapes on the graphs as shown.

Figure 7A. Positive Impression Reach with Export Feature

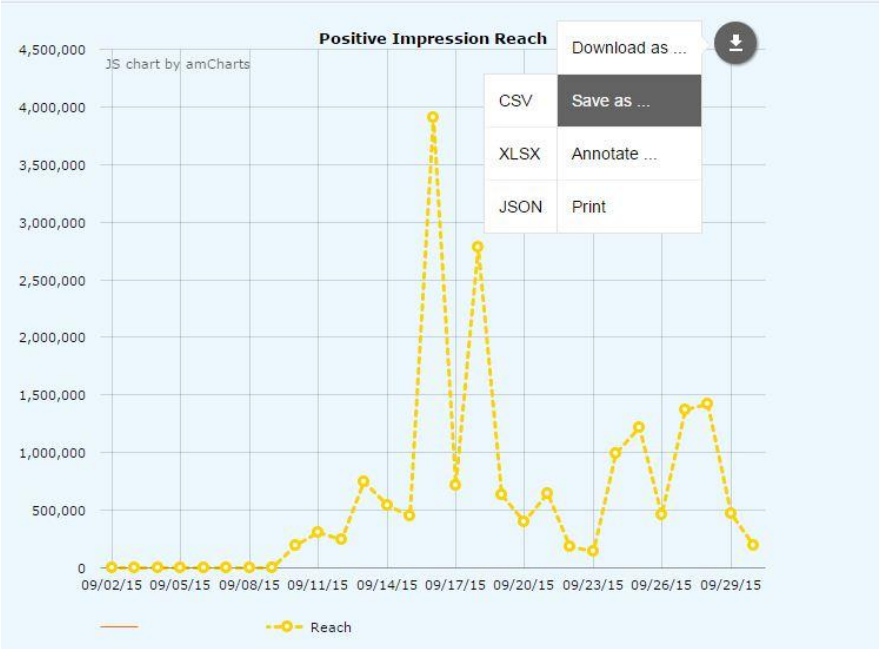
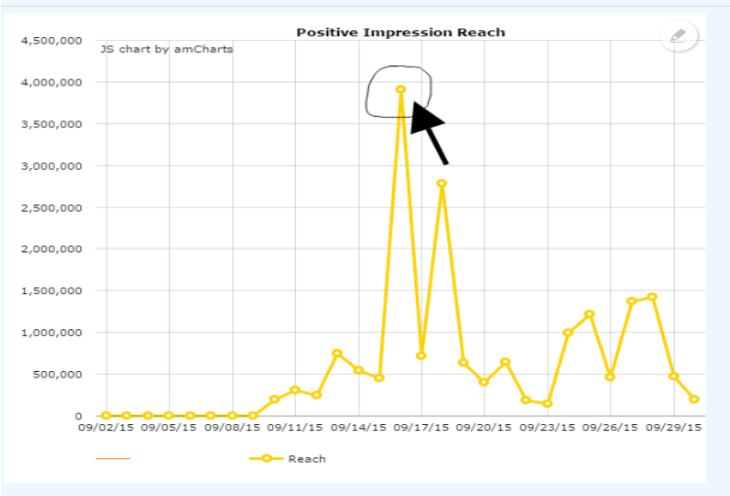


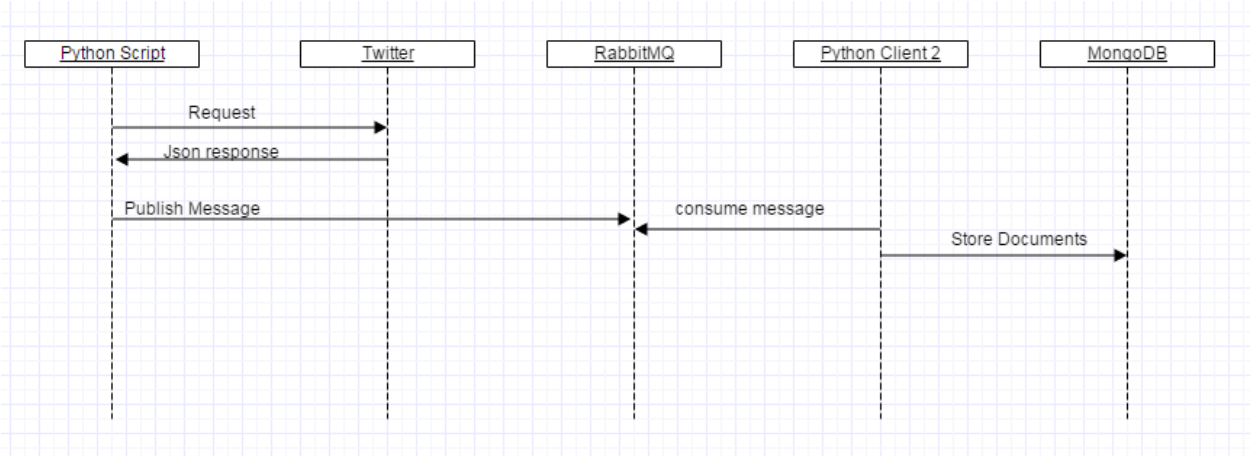
Figure 7B. Positive Impression Reach with Annotate Feature



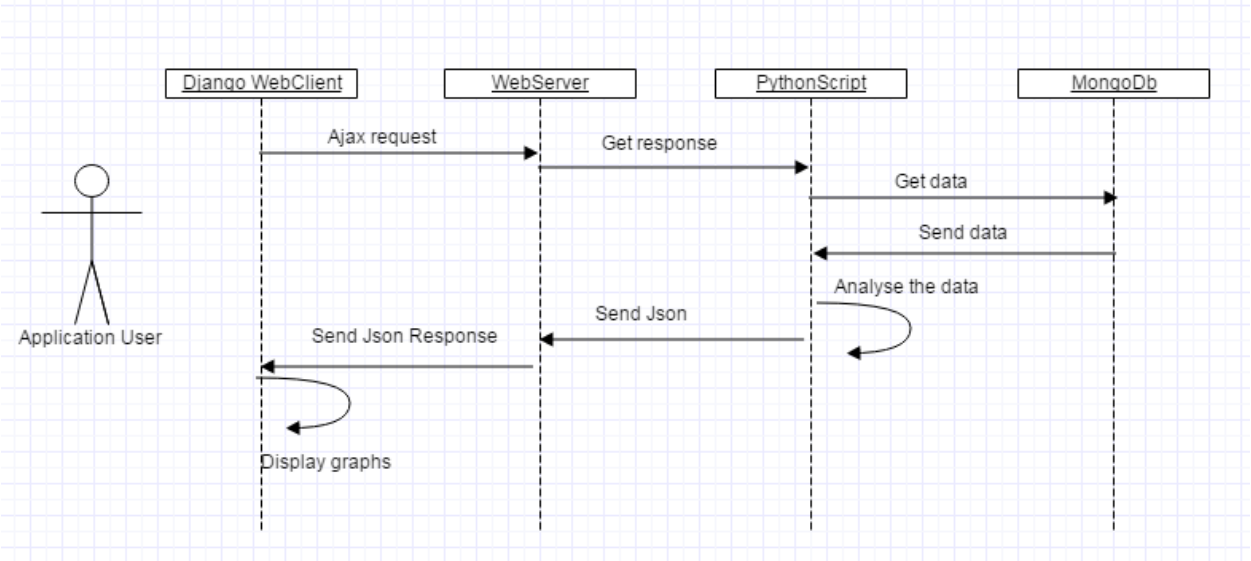
APPENDIX A

A.1 **SEQUENCE DIAGRAMS**

SEQUENCE DIAGRAM FOR STORING DATA IN REALTIME

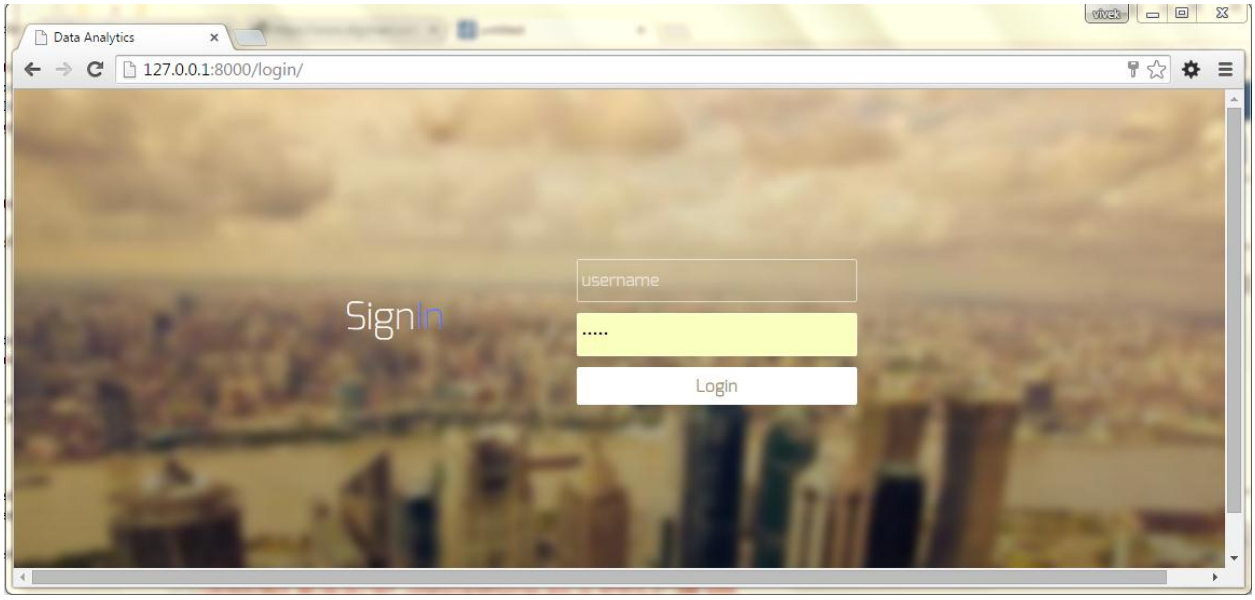


SEQUENCE DIAGRAM TO DISPLAY GRAPHS ON DASHBOARD



A.2 GUIDELINES TO USE THE PROJECT

LOGIN PAGE:



Once the login is sucessful the dashboard pages is loaded with the visualizations.

A.3 STEPS TO RUN THE PROJECT

Installation:

Install MongoDB from below link:

http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/

Install RabbitMQ server from following link: https://www.rabbitmq.com/download.html

Steps to run the project:

Run the manage.py file with the run server argument to start the server with the project deployed.

Once the project is deployed, a link to the application will be displayed on the console (Ex: http://127.0.0.1:8000/login)

A.4 Explanation of code in a nutshell

1.URL.py

URL.py defines the Uri mappings as shown below.

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^vis/', 'Visualization.views.dashboard'),
    url(r'^impressionpage/', 'Visualization.views.impressionpage'),
    url(r'^accounts/login/', 'Visualization.views.login'),
    url(r'^login/', 'Visualization.views.login'),
    url(r'^csreqperday', 'Visualization.views.csreqperday'),
    url(r'^csreqperweek', 'Visualization.views.csreqperweek'),
    url(r'^impression', 'Visualization.views.impression'),
    url(r'^negimpression', 'Visualization.views.impression'),
     url(r'^$/', 'Visualization.views.login'),
     url(r'^percentageofcs/', 'Visualization.views.percentageofcs'),
     url(r'^realtime/', 'Visualization.views.realTimeTweetSentiment'),
     url(r'^aveScore/', 'Visualization.views.Avescore'),
     url(r'^aveScoreperday/', 'Visualization.views.Avescoreperday'),
     url(r'^realtimescore/', 'Visualization.realtimeviews.score'),
      url(r'^responsetime/', 'Visualization.views.responsetime'),
]
```

2.Views.py

This script calls the appropriate business logic and returns a response to the request from the web browser.

```
def csreqperday(request):
    '''
        :param request:
        '''
    obj = CsRequestPerDay.getcsreq(1)
    data = obj.getCsreq()
    return HttpResponse(data, content_type='application/json')
def csreqperweek(request):
    obj = CsRequestPerDay.getcsreq(7)
    data = obj.getCsreq()
    return HttpResponse(data, content_type='application/json')
def impression(request):
    obj = CsRequestPerDay.getcsreq(7)
    data = obj.get_pos_imp()
    return HttpResponse(data, content_type='application/json')
def negimpression(request):
    obj = CsRequestPerDay.getcsreq(7)
    data = obj.get_neg_imp()
    return HttpResponse(data, content_type='application/json')
def Avescore(request):
    data = get_art()
    return HttpResponse(data, content_type='application/json')
def responsetime(request):
    print "inresponsetime"
    dataR = []
    dataR = ({"Handle":'September', "Cstweets":'45', "dtweets":'53'}, {"Handle":'October', "Cstweets":'28', "dtweets":'3
    data = json.dumps(dataR)
    return HttpResponse(data, content_type='application/json')
def Avescoreperday():
    data = get_art()
    return HttpResponse(data, content_type='application/json')
```

3.Settings.py

This script has all the setting of the Django project.The static files folder the applications and the templates are registered here

4.CalArt.py

This script gets the requests for Customer service and the replies for the requests and finds out the response time using the unique id of the tweet and the reply_to tweet id from mongodb

```
    for d in posts.find({"created_at": {"$gte": date1, "$lt": date2}}):
        one = d['id_str']  # tweet id from a cs request to deltaAssist
        complaint_id[one] = d['created_at']
        # print('afterfirstfor')

    for d in posts1.find():
        # print('in 2 for')
        two = d['in_reply_to_status_id_str']
        reply_id[two] = d['created_at']

    f = open('art.txt', 'w')
    for key_comp_id in complaint_id:

        for key_rep_id in reply_id:
            if(key_comp_id == key_rep_id):
                print("got one match")
                print(str(reply_id[key_rep_id] - complaint_id[key_comp_id]))
                f.write(str(reply_id[key_rep_id] - complaint_id[key_comp_id]))
                # print(complaint_id)
get_response_time()
```

5.CsRequestPerDay.py

This script queries the mongodb to get the tweets over a range of dates and also has functions to calculate the positive reach of the tweets and negative reach of tweets.

We used textblob python package to get tweet sentiment polarity. Based on the sentiment polarity score we process the data.

6.Percent_of_Cstweets.py

This script gets the distribution of tweets to various service handles

```
while date1 < date2:
    month_val = date1.strftime("%B")

    if(date1.strftime("%B") == month_val):
        date1 += dayDelta
        customer_service_count = posts1.find({"created_at": {"$gte": date1, "$lt": date1 + dayDelta }}).count()
        # print customer_service_count,'is the count on',date1
        temp = customer_service_count + temp
        delta_tweets_count = posts2.find({"created_at": {"$gte": date1, "$lt": date1 + dayDelta }}).count()
        temp2 = delta_tweets_count + temp2
        # print month_val,temp,temp2
        if(date1.strftime("%B") != month_val) :
            dataR.append({"Handle": month_val, "Cstweets": temp, "dtweets":temp2})
            temp = 0
            temp2 = 0
            print dataR

json_counts = json.dumps(dataR)
    # print json_counts
return json_counts
```

7. AverageScorePerDay.py

- This script gets the tweets from the MongoDb using the MongoClient from the Pymongo package by querying the database based upon certain criterion(Range of Dates).
- Once we get the tweet we extract the text from the tweet and use the TextBlob package to get the sentiment score of the tweet and then compute the average per day.
- Once the average is computed we send the result as a Json response over HTTP to the client which has sent the request to plot the graph for visualization.

```
startDate = dt.strptime("1/09/15 ", "%d/%m/%y ")
endDate = dt.strptime("30/09/15 ", "%d/%m/%y ")
dayDelta = datetime.timedelta(days=1)
while startDate < endDate:
    startDate += dayDelta
    for d in posts.find({"created_at":{"$gte": startDate, "$lt": startDate + dayDelta }}):
        tweet = TextBlob(d["text"])
        # print(tweet.sentiment.polarity)
        created_at.append(tweet.sentiment.polarity)
    #  print(tweet)
        score = sum(created_at) / (created_at.__len__())
        dataR.append({
        'date': startDate.day,
        'count': score
        })
    # counts_req[startDate.day]=count
```

8.ProduceRealtimeTweet.py

This script connects to twitter and posts the Json to the RabbitMQ queue.

```
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
channel = connection.channel()
    # set max queue size
args = {"x-max-length": 2000}
client = MongoClient('localhost', 27017)
db = client.twitterdata
channel.queue_declare(queue='twitter_topic_feed')

for tweet in tweepy.Cursor(api.search, q="@Delta", since="2015-01-09", result_type='recent', lang="en").items():
    data = {}
    posts = db.posts10
    data['created_at'] = tweet.created_at
    data['in_reply_to_status_id'] = tweet.in_reply_to_status_id
    data['text'] = tweet.text
    data['id_str'] = tweet.id_str
    data['in_reply_to_status_id'] = tweet.in_reply_to_status_id
    data['geo'] = tweet.geo
    data['friends_count'] = tweet.user.friends_count
    data['in_reply_to_screen_name'] = tweet.in_reply_to_screen_name
    data['in_reply_to_user_id'] = tweet.in_reply_to_user_id
    data = json.dumps(str(data))
```

9.ConsumeRealtimeTweet.py

This script connects to RabbitMQ server and consumes the tweets from the configured queue and inserts into MongoDB.

```
def consumers():

    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()
    channel.basic_consume(callback, queue='twitter_topic_feed', no_ack=True)
    print ' [*] Waiting for messages. To exit press CTRL+C'
    channel.start_consuming()

    # print(sum(created_at)/(created_at.__len__()))
def callback(ch, method, properties, body):
    print " [x] Received %r" % (body,)
    posts.insert(body)
    return body

client = MongoClient('localhost', 27017)
db = client.twitterdata
posts = db.posts10
consumers()
```

10.Dashboard.html

The code in the file makes calls to the service hosted by the django application and receives the Json response over HTTP.The json response is then parsed and plotted using javascript.