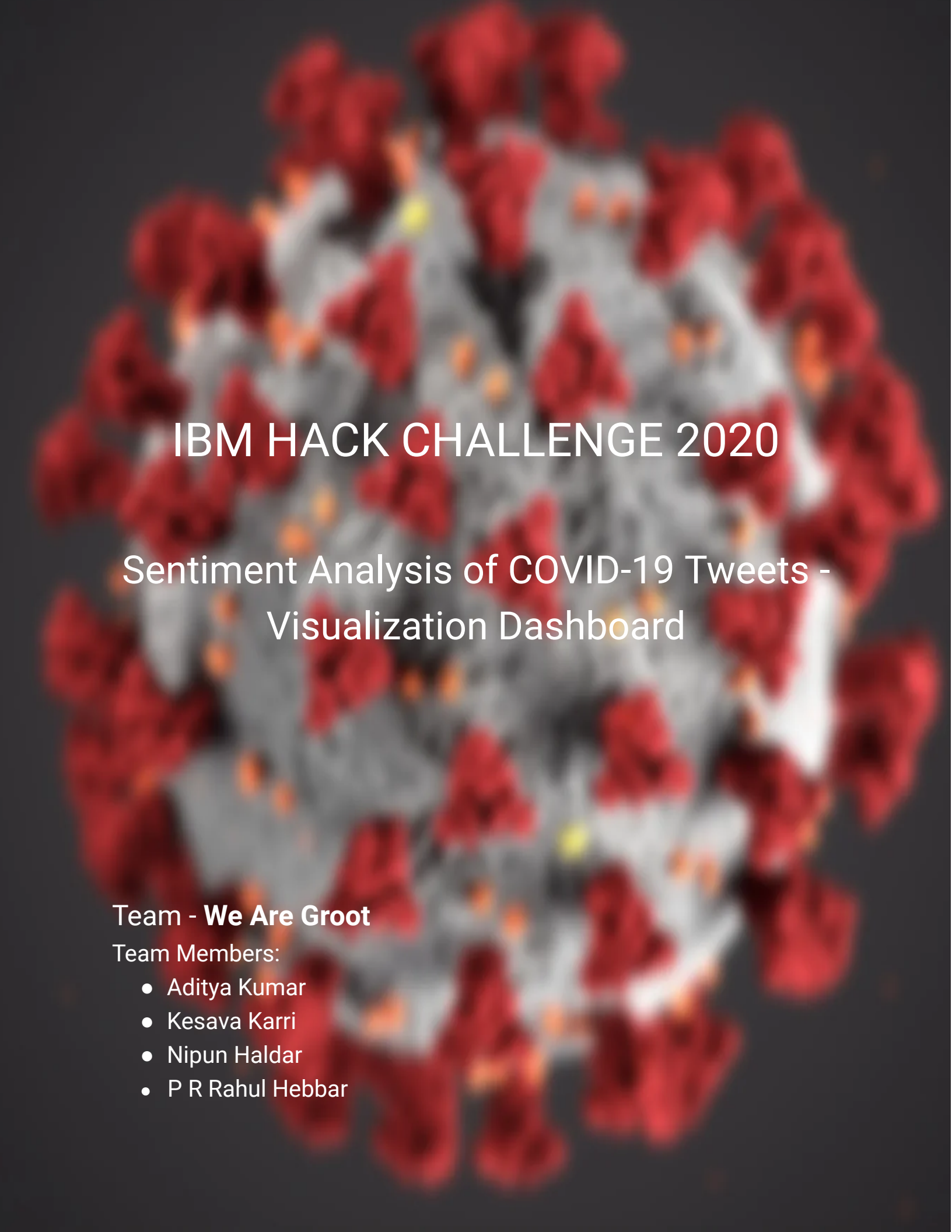


IBM HACK CHALLENGE 2020

Sentiment Analysis of COVID-19 Tweets - Visualization Dashboard

Team - **We Are Groot**

Team Members:

- Aditya Kumar
- Kesava Karri
- Nipun Haldar
- P R Rahul Hebbar

Index

1. Introduction

1.1 Overview

Twitter has always been a platform where people come together to put forth their opinions about issues. The current global concern is COVID-19 and this has led to a tremendous uproar of tweets relating to COVID-19. The large amounts of tweets produced definitely provide us the opportunity to analyze the tweets for all possible information - from people's activity and involvement to how the people are emotionally affected by the pandemic.

1.2 Purpose

With Data science and Data mining being a very growing and promising field, our goal is to exploit the tweet data and create a dashboard that will help us visualize all the analysis done on the tweet data like sentimental analysis, location heat maps, activity vs time graphs etc. This will help us gain insights regarding how well the people of our country are coping with the current pandemic. This project aims to classify tweets based on emotions, cluster tweets based on location and analyze people's activity with respect to time and then generate intuitive visual representation of the data.

2. Literature Survey

2.1 Existing Problem

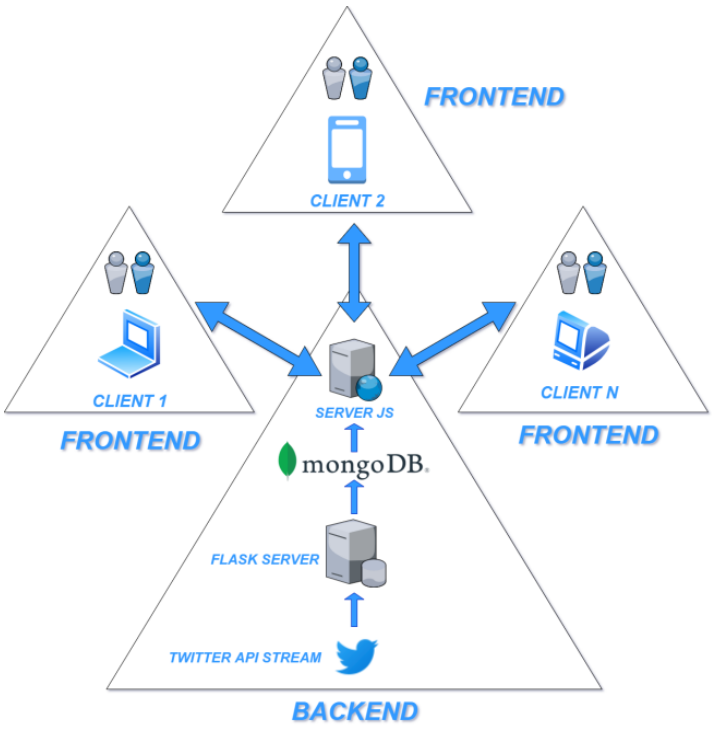
The main problems that exist in the current techniques are: inability to perform well in different domains, inadequate accuracy and performance in sentiment analysis based on insufficient labeled data, incapability to deal with complex sentences that require more than sentiment words and simple analyzing.

2.2 Proposed Solution

We will be using the twitter API to get the posts related to COVID-19 from Indian users and then we will be performing sentiment analysis on these collected posts to get the general idea of how Indians are affected by the crisis and how they are responding. We will be using technologies like Node.js , React.js, Python, Flask.

3. Theoretical Analysis

3.1 Block diagram



3.2 Software Designing

We can split the entire process needed for creating the intuitive Dashboard into smaller sections as follows.

i. Data Fetching
ii. Data Pre-processing
iii. Creating Backend
iv. Designing Frontend
v. Deployment

Data Fetching:

Here we will be running a python script that will be listening continuously to the twitter stream and fetch us the tweet along with various other parameters relating to the tweet like user info, location, date and time created and more.

Data Pre-processing:

In this section we will be using the data got from the above tweet listener and performing three actions on the tweet data them being data cleaning, sentiment

analysis and database entering. The database will be entered with all the necessary parameters we have.

Creating Backend:

We have the database with all the necessary parameters in it. We will be now reading the database here and extracting all the parameters to make cumulative calculations and then returning the refined parameters got to the frontend using socket connection.

Designing Frontend:

The Frontend designing is done using react.js and the cross platform ability supported by electron.js. We will be establishing socket connection with our backend and receiving continuous packets of data needed for visualization.

Deployment:

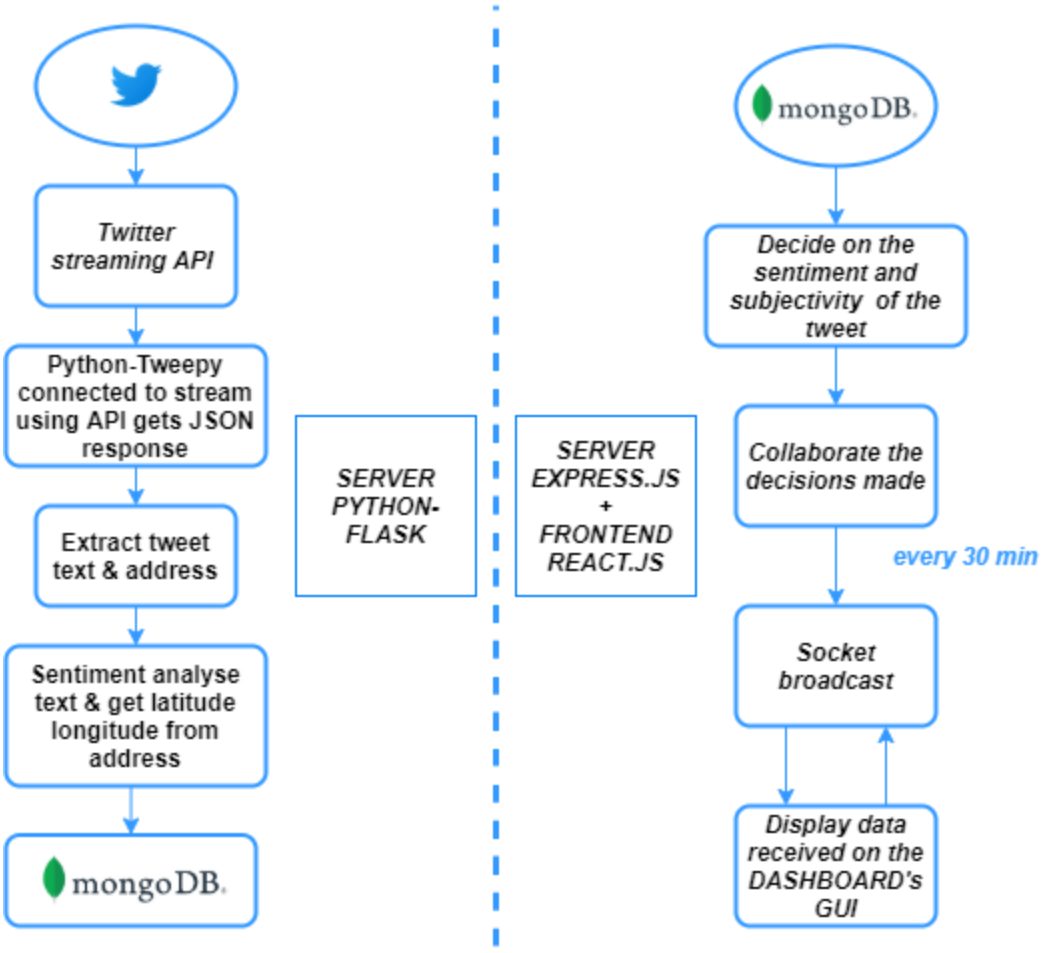
We will be deploying the Frontend and Backend on IBM cloud.

4. Experimental Investigations

At the start of the project we were thinking of using csv file for storing recent tweets and its various tweet parameters. But later we realised the inefficiency of csv - both in storing them in limited cloud memory and creating multiple read write connections. Then we thought of implementing a completely database-less dashboard by relying only on a checkpoint.json file. But this posed the biggest problem when it came to intensive memory consumption and recovery from server failure and loss of important information from tweets during the failure period. We also faced problems while running our flask stream updating server because of the necessity of either django or flask implementation for cloud foundry deployment.

But all the downfalls during the duration of the project helped us learn the various aspects of the projects, analyse the breakpoint and then finally rectify them to get the entire project work harmoniously.

5. Flowchart



5.1 Brief explanation of flowchart

Our Flowchart has two sections, the Backend Server implemented using Flask and the Backend + Frontend implemented using JavaScript. Let us look at the flow in detail

Python-Flask Server

1. We will be using the twitter's streaming API made for developers and provide filter parameters such that we get the resent COVID 19 related tweets made by Indians.
2. Python **Tweepy** library is used to connect to api twitter stream by supplying four keys - API key, API key secret, Service token and Service Token Secret to **OAuth protocol**. After OAuth get authorized we get access to live stream updates based on passed parameters. Every new tweet is given to use with lot of parameters like created at string, timestamp, unique tweet-id, user information, address etc, as a

json string.

3. The **json response** we receive has too many parameters that we don't need. So we will be using only the main and necessary parameters which include created at string, unique tweet id, tweet text and tweet location address parameter.
4. Once we have all the necessary parameters in our hand, we will perform operations on the tweet text and tweet location. We use **Unidecode** to remove emoji's and unwanted character and then use the cleaned text with **Textblob** to extract the **polarity (sentiment score)** and **subjectivity (emotional content score)**. And for forward **geo-coding** (getting latitude and longitude from address), we use the **Nominatim OpenStreetMaps** http request method to get the most probable latitude and longitude for the given address. Here it is important to note that we pass country code IN (ISO 3166-1 alpha-2 format country code for INDIA) to get only latitude and longitude of places within INDIA and for other address get null.
5. Once we process the parameters we send these refined results to **MongoDB Atlas** database as json documents containing tweets created at string, tweet id, polarity score, subjectivity score, latitude and longitude.

EXPRESS.JS Server and Frontend

1. Our server to fetch data from MongoDB and consolidate it is written in JavaScript. We are creating a backend server using **express.js** to establish a two way **socket.io websocket connection** between the backed js with the frontend dashboard.
2. We connect to our MongoDB database every 30 minutes and fetch all the new update using MongoDB find queries and a checkpoint tweet id.
3. Once we receive the new updates in the stream from the database, we run few decision instructions on polarity and subjectivity score to segregate the updates into positive, neutral or negative tweets and objective or subjective tweet.
4. After segregating the decisions, we collaborate the data in such a way that the object when sent to our frontend will get easily processed by graph library. We maintain the collaborated data in a checkpoint.json file for newly connected users and recovery from failures.
5. The socket.io by default broadcasts the new updates to all our connected clients every 30 minutes.
6. The frontend is written using **React.js** with GUI's components and styling using **Material.ui**, interactive graphs using **Nivo** library and location plot support using **Mapbox's API** for javascript.

6. Result

We have achieved the sentiment analysis of twitter COVID 19 tweets.Designed an intuitive dashboard to visualize the juice in it. We believe it is the fruit of our dedication and determination towards the project.

7. Advantages & Disadvantages

Advantages:

- User can get a clear view of the sentiments from the dashboard through various charts.
- For every interval of time we update the dashboard.
- Collective response to COVID-19 situation is shown.
- Location based activity is enabled.
- Tweets are analyzed for both emotion & sentiment content.
- Multiplatform compatibility

Disadvantages:

- The Geo-tagging of tweet if not done properly leads to errors in location based activity.
- If the tweets don't contain enough text the accuracy of the prediction reduces.

8. Applications

This can be used to fetch the insights of emotions from the tweets tweeted by the nation. We can also visualize the data for better understanding. It also plots the location from which the tweet has been made.

9. Conclusion

There's just too much data to process manually. Sentiment analysis helps businesses process huge amounts of data in an efficient and cost-effective way and we have implemented it .

10. Future Scope

The future of sentiment analysis is going to continue to dig deeper, far past the surface of the number of likes, comments and shares, and aim to reach, and truly understand, the significance of social media interactions and what they tell us about the end user. This forecast also predicts broader applications for sentiment analysis – brands will continue to leverage this tool, but so will individuals in the public eye, governments, nonprofits, education centers and many other organizations.

11. Bibliography

{https://developer.twitter.com/en/docs}
{http://docs.tweepy.org/en/latest/}
{https://pypi.org/project/Unidecode/}
{https://textblob.readthedocs.io/en/dev/}
{https://nominatim.org/release-docs/latest/}
{https://www.mongodb.com/cloud/atlas}
{https://flask.palletsprojects.com/en/1.1.x/}
{https://docs.cloudfoundry.org}
{https://developer.mongodb.com/quickstart/node-crud-tutorial#node-tutorial-read}
{https://socket.io/docs/emit-cheatsheet/}
{https://socket.io/docs/}
{https://reactjs.org/}
{https://docs.mapbox.com/mapbox.js/api/v3.3.1/}
{https://nivo.rocks/storybook/?path=/story/bar--with-marker}
{https://nivo.rocks/storybook/?path=/story/line--stacked-lines}
{https://nivo.rocks/storybook/?path=/story/pie--default}
{https://material-ui.com/getting-started/usage/}
{https://www.electronjs.org/docs}

APPENDIX

Source Code:

Python - Flask Server

```
class listener(StreamListener):
# Called initially to connect to the Streaming API
    def on_connect(self):
        print("You are now connected to the streaming API.")
# On error - if an error occurs, display the error status code
    def on_error(self, status_code):
        print('An Error has occured: ' + repr(status_code))
        return False
# Connects to your mongoDB and stores the tweet
    def on_data(self, data):
        try:
            # Decode the JSON from Twitter
            datajson = json.loads(data)

```

```
            # grap the id
            tweet_id = datajson['id_str']
             # grab the 'created_at' data from the Tweet
            created_at = datajson['created_at']
            # grab address
            address = datajson['user']['location']
            lati = None
            longi = None
            if address != None:
                address = unidecode(address)
                url = 'https://nominatim.openstreetmap.org/search/' + \
                    urllib.parse.quote(address) + \
                    '?format=json&limit=1&countrycodes=in'
                response = requests.get(url).json()
                if len(response) > 0:
                    lati = response[0]["lat"]
                    longi = response[0]["lon"]

            tweet = unidecode(datajson['text'])
            analysis = TextBlob(tweet)
            polarity = analysis.sentiment.polarity
            subjectivity = analysis.sentiment.subjectivity
            # print to the screen that we have collected a tweet
            print("Tweet collected at " + str(created_at))
            t = {
                "tweet_id": tweet_id,
                "created_at": created_at,
                "latitude": lati,
                "longitude": longi,
                "polarity": polarity,
                "subjectivity": subjectivity
            }
            print(t)
            # insert the data into the mongoDB
             db.insert_one(t)

        except Exception as e:
             print("error: ", e)
```

JavaScript - Express.js Server

```
# check for the checkpoint.json file
const updater = async (io, interval) => {
  # if file is absent create new
  if (!fs.existsSync("checkpoint.json")) {
    init(interval);
  }
  const time_str = new Date().toLocaleTimeString("en-US", {
    timeZone: "Asia/Kolkata",
    hour12: false,
  });

  # connect instance to our MongoDB atlas database
  const client = new MongoClient(uri, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  });

  # connecting to the database
  await client.connect();
  console.log("db connected");

  # reading checkpoint.json
  var data = fs.readFileSync("checkpoint.json");
  var tweet_id_fg = JSON.parse(data)[0];
  var pie = JSON.parse(data)[1];
  var line = JSON.parse(data)[2][0];
  var sub_bar = JSON.parse(data)[3][0];
  var loc = JSON.parse(data)[4];

  # cursor to fetch updated database from a previous checkpoint
  const cursor = client
    .db("sentiment")
    .collection("tweets")
    .find({ tweet_id: { $gt: tweet_id_fg } })
    .sort({ _id: 1 });

  # parsing the cursor response to array
  const result = await cursor.toArray();
  console.log(result);
  var pos_obj = 0,
    neg_obj = 0,
```

```
    neu_obj = 0;
  var factual = 0,
    emotional = 0;
  if (result.length > 0) {
    result.forEach((res) => {
      var polarity = res.polarity;
      var pol_flag = null;

      # sentiment decision making
      if (polarity > 0) {
        pie[0].value++;
        pos_obj++;
        pol_flag = 1;
      } else if (polarity < 0) {
        pie[1].value++;
        neg_obj++;
        pol_flag = -1;
      } else {
        pie[2].value++;
        neu_obj++;
        pol_flag = 0;
      }
      # emotion content decision making
      var subj = res.subjectivity;
      if (subj >= 0.5) emotional++;
      else factual++;

      # process got latitude and longitude
      var lat = res.latitude;
      var long = res.longitude;
      if ((lat !== null) & (long !== null)) {
        if (pol_flag === 1)
          loc.push({ id: "positive", latitude: lat, longitude: long });
        else if (pol_flag === -1)
          loc.push({ id: "negative", latitude: lat, longitude: long });
        else if (pol_flag === 0)
          loc.push({ id: "neutral", latitude: lat, longitude: long });
        if (loc.length > 50) loc.shift();
      }

      # get last tweet id for new checkpoint
      tweet_id_fg = res.tweet_id;
    });
  }
```

```
  # bargraph object
  sub_bar.push({
    x: time_str,
    objectivity: factual,
    subjectivity: emotional,
  });
  if (sub_bar.length > 7) sub_bar.shift();

  # line graph object
  line[0].data.push({ x: time_str, y: pos_obj });
  line[1].data.push({ x: time_str, y: neg_obj });
  line[2].data.push({ x: time_str, y: neu_obj });
  if (line[0].data.length > 10) {
    line[0].data.shift();
    line[1].data.shift();
    line[2].data.shift();
  }
  const timeperiod = interval;

  # write new checkpoint file for new users and failure handling
  fs.writeFileSync(
    "checkpoint.json",
    JSON.stringify([
      tweet_id_fg,
      pie,
      [line, timeperiod],
      [sub_bar, timeperiod],
      loc,
    ])
  );

  # broadcast to all client
  io.emit("pie", pie);
  io.emit("line", [line, interval]);
  io.emit("bar", [sub_bar, interval]);
  io.emit("location", loc);

  #close database connection
  await client.close();
 };
```

Dashboard - React.js

```
const Dashboard = (props) => {
  const classes = useStyles();

  //drawer states and functions
  const [isOpen, setDrawer] = useState(false);

  const toggleDrawer = (isOpen) => (event) => {
    if (
      event.type === "keydown" &&
      (event.key === "Tab" || event.key === "Shift")
    ) {
      return;
    }

    setDrawer(isOpen);
  };

  //Location Dialog states and functions
  const [isLocationOpen, setLocationOpen] = useState(false);

  //loading
  const [isLoading, setLoading] = useState(true);

  //piechart data
  const [pieData, setPieData] = useState(null);

  //line data
  const [lineData, setLineData] = useState(null);

  //bar data
  const [barData, setBarData] = useState(null);

  //location data
  const [locationData, setLocationData] = useState(null);

  // getting data from backend
  useEffect(() => {
    socket.emit("dashboard");
    socket.on("pie", (data) => {
      setPieData(data);
    });
    socket.on("line", (data) => {
      setLineData(data);
```

```
    });
    socket.on("bar", (data) => {
      console.log(data);
      setBarData(data);
    });
    socket.on("location", (data) => {
      setLocationData(data);
      setLoading(false);
    });
  }, []);

  if (isLoading) {
    return (
      <div>
        <CircularProgress />
      </div>
    );
  } else {
    return (
      <div style={{ height: "100vh", overflowX: "hidden" }}>
        {/* Navbar */}
          <AppBar position="static" classes={{ root: classes.appbarTheme
}}>
          <Toolbar variant="dense">
            <IconButton onClick={toggleDrawer(true)}>
              <MenuIcon />
            </IconButton>
            <Typography variant="h6">Dashboard</Typography>
          </Toolbar>
        </AppBar>

        {/* Side navigation drawer */}
        <Drawer
          anchor="left"
          open={isOpen}
          onClose={toggleDrawer(false)}
          classes={{ paper: classes.drawerTheme }}
        >
          <List style={{ width: "250px" }}>
            <ListItem button onClick={() => props.history.push("/")}>
              <ListItemIcon>
                <HomeIcon />
              </ListItemIcon>
              <ListItemText primary="Home" />
```

```
87              </ListItem>
88                <ListItem button
89                 onClick={() => props.history.push("/dashboard")}>
90                <ListItemIcon>
91                  <DashboardIcon />
92                </ListItemIcon>
93                <ListItemText primary="Dashboard" />
94              </ListItem>
95              <ListItem button onClick={() => props.history.push("/team")}>
96                <ListItemIcon>
97                  <GroupIcon />
98                </ListItemIcon>
99                <ListItemText primary="Meet the Team" />
100             </ListItem>
101           </List>
102         </Drawer>
103         {/* Graph components - tiles */}
104         <Grid
105           style={{ height: "90vh" }}
106           container
107           direction="row"
108           justify="space-evenly"
109           alignItems="center"
110         >
111           <Grid item xs={12} sm={12} lg={6}>
112          <Paper classes={{ root: classes.paperTheme }} style={tileStyle}>
113               <Typography
114                 variant="h5"
115                 align="center"
116                 style={{ color: "rgba(255,255,255,0.87)" }}
117               >
118                 Sentiment vs Time Graph
119               </Typography>
120               <div style={{ height: "90%" }}>
121                 <LineChart data={lineData} />
122               </div>
123             </Paper>
124           </Grid>
125
126           <Grid item xs={12} sm={12} lg={6}>
127             <Paper
128               classes={{ root: classes.paperTheme }}
129               style={tileStyle}
130               onClick={() => setLocationOpen(true)}
```

```
131             >
132               <Map interactive={false} data={locationData} />
133             </Paper>
134           </Grid>
135
136           <Grid item xs={12} sm={12} lg={6}>
137           <Paper classes={{ root: classes.paperTheme }} style={tileStyle}>
138               <Typography
139                 variant="h5"
140                 align="center"
141                 style={{ color: "rgba(255,255,255,0.87)" }}
142               >
143                 Subjectivity vs Time Graph
144               </Typography>
145               <div style={{ height: "90%" }}>
146                 <BarChart data={barData} />
147               </div>
148             </Paper>
149           </Grid>
150
151           <Grid item xs={12} sm={12} lg={6}>
152           <Paper classes={{ root: classes.paperTheme }} style={tileStyle}>
153               <Typography
154                 variant="h5"
155                 align="center"
156                 style={{ color: "rgba(255,255,255,0.87)" }}
157               >
158                 Polarity Distribution (All-Time)
159               </Typography>
160               <div style={{ height: "90%" }}>
161                 <PieChart data={pieData} interactive={false} />
162               </div>
163             </Paper>
164           </Grid>
165         </Grid>
166         <Location
167           open={isLocationOpen}
168           toggle={setLocationOpen}
169           data={locationData}
170         />
171       </div>
172     );
173   }
174   };
```