INSTITUTE FOR ADVANCED
COMPUTING AND
SOFTWARE
DEVELOPMENT
AKURDI, PUNE

Project on

Troll Detection, Categorization and Statistical Analysis of Tweets

Submitted By:

Group No: 5
BADAR AKHTAR 1507
KESHAV BEDRE 1509
BIBEKJYOTI NATH 1510

Mr. Prashant Karhale
Centre Coordinator

Mr. Akshay Tilekar
Project Guide

Mr. Rahul Pund (Internal Guide)
Mr. Manish (Internal Guide)

Table of Content

Title	Page Number
1. Chapter 1: Introduction	1
1.1. Abstract	1
1.2. Product Scope	1
1.3. Aims and Objective	1
2. Chapter 2: Software and Hardware Requirements	2
3. Chapter 3: Data Extraction and Wrangling	3
3.1. Connectivity with Twitter	3
3.2. Steps in Data Collection	3
3.3. Sector Chosen for Data Collection	4
3.4. Database Connectivity and Data Extraction	4
3.5. Selecting Useful Fields from Raw Database to New Database	7
3.6. Function to Load Data in Pandas Dataframe	8
3.7. Loading Training Data	11
4. Chapter 4: Data Pre-processing	13
4.1. Feature Extraction	13
4.2. Training and Testing Split	16
4.3. Data Vectorization	16
5. Chapter 5: Model Training and Model Apply	18
5.1. Model Training and Validation	18
5.2. Applying Trained Model to Extracted Data	19
6. Chapter 6: Sentiment Polarity	20
7. Chapter 7: Exploratory Data Analysis	22
7.1. List of Not Stop Words	22
7.2. List of Total Number of Words in a Tweet	23
7.3. List of Stop Words	23
7.4. Sector Wise Tweet Count and Percentage	25
7.5. Category Wise Troll / Not-Troll Distribution	27
7.6. Category Wise WordCloud	28
7.7. Relationship Between Tweet Sentiment and Tweet Type	37
7.8. User Tweet Study	38
8. Chapter 8: Inferential Statistics	40
8.1. Chi-Square Test for Association Between Sectors and Trolling	40
8.2. T-Test for Equality of Two Population Mean	41
8.3. One-Way Anova	42
8.3.1. Sector Wise Significance of User Favourite Count	42
8.3.2. Sector Wise Significance of User Follower Count	43
8.3.3. Sector Wise Significance of Retweet Count	44
8.4. Chi-Square Test for Association Between Tweet Type and Sentiment Category	45
8.5. T-Test for Equality of Sentiment Polarity for Troll and Not-Troll	46
9. Chapter 9: Conclusions	47
10. Chapter 10: Future Scope	48
11. References	49

Introduction

Abstract:

Now a day's social media plays very crucial role in formation of public opinion and Social influence theory considers how individuals influence thoughts and behaviours of others. Public opinion is sometimes driven by rumours, fake allegations. This fake, aggressive comments on social media are known as Trolls. This troll is due to some extremists who try to impose their opinion on society with the help of giant social media. These trolls must be analysed and its ramification on society must be studied as a responsible citizen of India.

The project is devoted to study the trolling behaviour of people on Twitter. Some important sectors are chosen to for study. The tweets regarding these sectors are sampled for the analysis. This is done using twitter API and performed in Python. The sampled tweets are then cleaned. These cleaned tweets are classified into Trolls and Non-Troll using a trained model. Data collected on tweets of this sectors is used to derive test whether sector wise trolling is significant. Another objective is to identify the targeted trolling.

Product Scope:

The study was conducted on live data gathered from twitter based on various categories like politics, sports personalities, Govt Institutions etc. This report focuses on the data gathered from the tweets. The Scope of the project is to classify the tweets gathered into trolls and non-trolls using NLP and other machine learning tools. It fetches live tweets from twitter-based API on various categories which we randomly selected.

Aims & Objectives:

1. To identify the trolls from tweets.
2. To test whether retweet count differ with trolls and Non-Trolls.
3. To identify commonly used keywords associated with different personalities from sampled tweets.
4. To test sector wise retweet count, follower count and favourite count differ significantly.
5. To test the dependency between Sectors and type of tweet i.e., Troll/ Non-troll.
6. To check the correlation between trolling and Sentiment

Software & Hardware Requirements

Operating System: Linux / Windows 8.1 and above / MacOS Catalina and above

System Requirements:

	Minimum Requirement	Recommended Requirement
Processor	Intel i5 5th Generation or AMD Ryzen 3	Intel i7 5th Generation or AMD Ryzen 7
RAM	8 GB	16 GB
HDD / SSD	120 GB	250 GB
Network	1 Mbps	5 Mbps

Technologies Used

1. MongoDB
2. Machine Learning

Programming Languages

1. Python
2. MongoDB

Python Libraries required

1. tweepy
2. pandas
3. numpy
4. json
5. pymongo
6. preprocessor
7. nltk
8. textblob
9. sklearn
10. vadersentiment
11. matplotlib
12. seaborn
13. collections
14. wordcloud
15. scipy

Data Extraction and Wrangling

Connectivity with Twitter:

In the project twitter API is used to extract live data from twitter. Twitter developer accounts have been created and the consumer keys and access tokens generated are used for data extraction. The tweepy library is used for extracting live data from twitter.

```
#Getting input from the user like:
#Twitter Consumer Key
consumer_key = input("Enter Consumer Key: ")
#Twitter Consumer Secret Key
consumer_secret = input("Enter Consumer Secret: ")
#Twitter Access Token
access_token = input("Enter Access Token: ")
#Twitter Access Token Secret
access_secret = input("Enter Access Secret: ")

#Authenticating the twitter consumer key and secret using tweepy library
authenticate = tpy.OAuthHandler(consumer_key, consumer_secret)
#Authenticating the twitter access token and secret using tweepy library
authenticate.set_access_token(access_token, access_secret)

#Creating the twitter API using tweepy library
api = tpy.API(authenticate)
```

Calling the above call to make the connectivity with Twitter API:

```
1 api = create_api()
```

```
Enter Consumer Key: FdZRN6aKXSW7wv8A8sHiDdNuW
Enter Consumer Secret: 17aIBSx65MUbLozxhMEEbzF4CAxi6x6g4C9CMPT91GzcYTpHtq
Enter Access Token: 312148876-IycZMfSVSKvLj5X4Tc9oY8MvwpJcOgK2WBiLWChj
Enter Access Secret: MTxLeabNLPxVbvMTq8rWpvEzaZbpaQyLRkco2kflqd42e
```

Steps in Data collection:

The training dataset with troll/non-troll labels is downloaded from DATATURKS website and sector wise data for the study is extracted using twitter API.

Collecting and cleaning data is further subdivided into the following parts:

- Scraping twitter for raw data.
- Storing in a MongoDB database.
- Transforming database: Removing unnecessary columns.
- Pre-processing of the data.

Specific keyword(s) are chosen to track/search for in the tweets.

Sectors chosen for data collection:

Political: Narendra Modi, Rahul Gandhi, etc.
Pharma: Cipla, Astrazeneca, etc.
Sports: Virat Kohli, Saina Nehwal, etc.
Automobile: Honda, BMW, etc.
IT: Infosys, TCS, etc
Celebrity: Salman Khan, Johnny Depp, etc.
Govt Institutions: RBI, IRCTC, etc.
Gaming: Cyberpunk2077, Mario, etc.
Movies and TV shows: South Park, Fast and Furious, etc.

Database Connectivity & Data Extraction:

The extracted tweets are stored in MongoDB database. MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. We have used MongoDB Compass GUI to ease out the task of importing and exporting json and viewing the extracted data.

The connectivity of MongoDB with python is done using pymongo library in python. The PyMongo distribution contains tools for interacting with MongoDB database from Python. The pymongo package is a native Python driver for MongoDB.

A function **twitter_extraction** is created to pass the keyword and tweet return count. The API that was generated needs to be passed to the function to make the connectivity with the twitter API using Tweepy library available in Python.

twitter_extraction function returns 2 parameters the **raw_tweet_collection** name which was given to the collection that we have created in MongoDB and the **mongo_clink_link** which holds the MongoDB client information.

```
#Function to extract twitter data
def twitter_extraction(api):

    #Getting the clinet link for MongoDB connection from the user
    mongo_client_link = input("Please enter the connectivity link: ")

    #creating the connection with MongoDB using the pymongo library
    #passing the information to the object
    client = pymongo.MongoClient(mongo_client_link)
    #creating the database in MongoDB to store the raw data extracted from twitter API
    raw_tweet_db = client['raw_tweet_db']
    #creating the collection to store the extracted information inside the database
    raw_tweet_collection = raw_tweet_db['raw_tweet_collection']

    #Getting the search string from the user
    tweet_search = input("Please enter the tweet you want to retrieve: ")

    #Getting the total number of records that you want retrive from twitter
    tweet_max = int(input("Please enter how many records you want to retrieve: "))

    #Getting the category name for an additional column in the database
    category_name = input("Please enter category name: ")

    #Loop where it is retrieving the data from twitter and parsing through each records to save it in MongoDB
    for tweet in tpy.Cursor(api.search, q = tweet_search, lang = 'en',
                            exclude='retweets', tweet_mode = 'extended').items(tweet_max):

        #converting the retrieve data into dictionary format and storing into variable
        raw_tweets = dict(tweet._json)

        indx = list(raw_tweets.values())[1]

        print(json.dumps(raw_tweets, indent = 3))

        #inserting each record retrieved from the Twitter API in MongoDB database
        raw_tweet_collection.insert_one(raw_tweets)
        #adding a new column in the extsting database for category
        raw_tweet_collection.update_many({'id': indx},{"$set": {'category_name': category_name}})

    return raw_tweet_collection, mongo_client_link
```

Calling of the Function and Fetching Tweets using Twitter API

```
1 rtc, mcl = twitter_extraction(api)
```

```
Please enter the connectivity link: mongodb://127.0.0.1:27017/
Please enter the tweet you want to retrieve: princeofpersia
Please enter how many records you want to retrieve: 10
Please enter category name: Game
{
   "created_at": "Wed Mar 31 05:43:18 +0000 2021",
   "id": 1377134388052299785,
   "id_str": "1377134388052299785",
   "full_text": "The first game that I played, and lasted for like months and couldn't finish it until I reached 18 and well
... Internet helped me :))\n#PrinceOfPersia: 2 #TheShadowAndTheFlame https://t.co/k4mQtQlsRj",
   "truncated": false,
   "display_text_range": [
      0,
      175
   ],
   "entities": {
      "hashtags": [
         {
            "text": "PrinceOfPersia",
```

The extracted tweets are then stored in MongoDB.

Extracted Raw Data imported in MongoDB:


```
_id: "6048ac0ebf0e1ebb8f34b3f8"
category_name: "Govt Institutions"
contributors: "null"
coordinates: "null"
created_at: "Wed Mar 10 11:22:45 +0000 2021"
display_text_range: "[15,89]"
entities: Object
    hashtags: "[]"
    symbols: "[]"
    urls: "[]"
    user_mentions: "[{"screen_name":"FTIIndia","name":"Franklin Templeton","id":552640580,..."
favorite_count: "0"
favorited: "FALSE"
full_text: "@FTIIndia @RBI EOD tomorrow please. I don't have time to follow up wit..."
geo: "null"
id: "1.36961E+18"
id_str: "1.36961E+18"
in_reply_to_screen_name: "FTIIndia"
in_reply_to_status_id: "1.36961E+18"
in_reply_to_status_id_…: "1.36961E+18"
in_reply_to_user_id: "552640580"
in_reply_to_user_id_str: "552640580"
is_quote_status: "FALSE"
lang: "en"
metadata: Object
    iso_language_code: "en"
    result_type: "recent"
retweet_count: "0"
retweeted: "FALSE"
source: "<a href="http://twitter.com/download/android" rel="nofollow">Twitter f..."
truncated: "FALSE"

user: Object
    contributors_enabled: "FALSE"
    created_at: "Sat Aug 01 06:15:30 +0000 2009"
    default_profile: "TRUE"
    default_profile_image: "FALSE"
    description: "Just Human. RT is not endorsement. Views are mine, agreement disagreem..."
  › entities: Object
    favourites_count: "2098"
    follow_request_sent: "FALSE"
    followers_count: "46"
    following: "FALSE"
    friends_count: "47"
    geo_enabled: "FALSE"
    has_extended_profile: "FALSE"
    id: "61977630"
    id_str: "61977630"
    is_translation_enabled: "FALSE"
    is_translator: "FALSE"
    lang: "null"
    listed_count: "0"
    location: "Delhi, India"
    name: "Just Human"
    notifications: "FALSE"
    profile_background_co… : "C0DEED"
    profile_background_im… : "http://abs.twimg.com/images/themes/theme1/bg.png"
    profile_background_im… : "https://abs.twimg.com/images/themes/theme1/bg.png"
    profile_background_ti… : "FALSE"
    profile_image_url: "http://pbs.twimg.com/profile_images/1305509274517471237/03ho3LEz_norma..."
    profile_image_url_htt… : "https://pbs.twimg.com/profile_images/1305509274517471237/03ho3LEz_norm..."
    profile_link_color: "1DA1F2"
    profile_sidebar_borde… : "C0DEED"

  profile_sidebar_fill_… : "DDEEF6"
  profile_text_color: "333333"
  profile_use_backgroun… : "TRUE"
  protected: "FALSE"
  screen_name: "JustHuman81"
  statuses_count: "3087"
  time_zone: "null"
  translator_type: "none"
  url: "null"
  utc_offset: "null"
  verified: "FALSE"
```

Selecting useful fields from Raw data to new Database:

Tweeter API provides us a lot of data regarding the tweets that we have gathered. In this section we are selecting only those fields that we felt is necessary for the execution of our project and saved it inside a new collection.

The function takes 2 parameters as input mentioned below:

1. **Mongo_clinet_link:** To establish the connection between the MongoDB and the program.
2. **Raw_Tweet_Collection:** The name of the collection in which we had store the raw tweets that we have fetched from Twitter.

The function returns 2 parameters mentioned below:

1. **selected_tweet_columns_db:** This provides the name of the database in which the necessary columns of the tweets are stored.
2. **selected_tweet_columns_collection:** This provides the name of the collection in which the database is present.

```
def data_selection(mcl, raw_tweet_collection):

    #creating the connection with MongoDB using the pymongo library
    #passing the information to the object
    client = pymongo.MongoClient(mcl)

    #Loop to retrieve selected columns from the raw data stored inside the database
    query = raw_tweet_collection.find({},{'_id':0, 'created_at':1, 'id':1, 'full_text':1,
                                    'entities.hashtags.text':1, 'entities.user_mentions.screen_name':1,
                                    'entities.user_mentions.id':1, 'user.id':1, 'user.name':1 , 'user.screen_name':1,
                                    'user.location':1, 'user.protected':1, 'user.followers_count':1, 'user.friends_count':1,
                                    'user.listed_count':1, 'user.created_at':1, 'user.favourites_count':1, 'user.statuses_count'
                                    'retweeted_status.created_at':1, 'retweeted_status.id':1, 'retweeted_status.full_text':1,
                                    'retweeted_status.user_mentions.screen_name':1,
                                    'retweet_count':1, 'favorite_count':1,
                                    'possibly_sensitive':1, 'lang':1, 'category_name':1})

    #creating a new database in MongoDB to store the selected data extracted from MongoDB
    selected_tweet_columns_db = client['selected_tweet_columns_db']

    #creating the collection to store the selected information inside new collection
    selected_tweet_columns_collection = selected_tweet_columns_db['selected_tweet_columns_collection']

    #Loop to parse through the selected data and store inside new collection
    for q in query:

        #printing the selected data
        print(json.dumps(q, indent = 3))

        #inserting the selected tweets inside new collection in MongoDB Database
        selected_tweet_columns_collection.insert_one(q)

    return selected_tweet_columns_db, selected_tweet_columns_collection
```

Calling of the Function to Select the Specified Columns from the Database:

```
stcd, stcc = data_selection(mcl, rtc)
```


```
{
   "category_name": "Govt Institutions",
   "created_at": "Wed Mar 10 11:22:45 +0000 2021",
   "entities": {},
   "favorite_count": "0",
   "full_text": "@FTIIndia @RBI EOD tomorrow please. I don't have time to follow up with FTI all the time.",
   "id": "1.36961E+18",
   "lang": "en",
   "retweet_count": "0",
   "user": {
      "created_at": "Sat Aug 01 06:15:30 +0000 2009",
      "favourites_count": "2098",
      "followers_count": "46",
      "friends_count": "47",
      "id": "61977630",
      "listed_count": "0",
      "location": "Delhi, India",
      "name": "Just Human",
      "protected": "FALSE",
```


Selected Tweet Data Stored in MongoDB:


```
_id: ObjectId("605d9588e11e98a0662acff4")
category_name: "Govt Institutions"
created_at: "Wed Mar 10 11:22:45 +0000 2021"
entities: Object
favorite_count: "0"
full_text: "@FTIIndia @RBI EOD tomorrow please. I don't have time to follow up wit..."
id: "1.36961E+18"
lang: "en"
retweet_count: "0"
user: Object
    created_at: "Sat Aug 01 06:15:30 +0000 2009"
    favourites_count: "2098"
    followers_count: "46"
    friends_count: "47"
    id: "61977630"
    listed_count: "0"
    location: "Delhi, India"
    name: "Just Human"
    protected: "FALSE"
    screen_name: "JustHuman81"
    statuses_count: "3087"
```


Function to load the data to pandas dataframe:

In this section we are loading the selected data in the dataframe using pandas. The function takes the below parameters as input:

1. **selected_tweet_columns_db:** This provides the name of the database in which the necessary columns of the tweets are stored.
2. **selected_tweet_columns_collection:** This provides the name of the collection in which the database is present.
3. **Mongo_clinet_link:** To establish the connection between the MongoDB and the program.

The function returns 2 parameters mentioned below:

1. **rd_tweets_df:** This returns the entire dataframe which holds all the columns that were selected for the project.
2. **new_tweet_df:** This returns only the text section/tweets which we would be pre-processing and classifying into Trolls and Non-Trolls.

During extraction we noticed that there were duplicate tweets that were getting fetched (Retweets of the original tweets without modification) so, we have removed all those tweets from the dataset before importing it in the dataframe. We have also, added a section to store the dataframe as **.csv**.

```
#Function to load the data to pandas dataframe
def load_data(selected_tweet_columns_db, selected_tweet_columns_collection, mongo_client_link):

    #creating the connection with MongoDB using the pymongo library
    #passing the information to the object
    client = pymongo.MongoClient(mongo_client_link)

    #assigining MongoDB database to a variable
    #mongo_db = selected_tweet_columns_db
    mongo_db = client['selected_tweet_columns_db']

    #assigning MongoDB database collection to a variable
    #collection = mongo_db.selected_tweet_columns_collection
    collection = mongo_db['selected_tweet_columns_collection']

    tweets_df = pd.json_normalize(collection.find({},{'_id':0}), max_level=2)

    #dropping all the duplicate rows from
    rd_tweets_df = tweets_df.drop_duplicates(subset = ['full_text'])

    #Extracting only the tweets from the exisisting dataframe and creating a new dataframe
    data = [rd_tweets_df['full_text'], rd_tweets_df['category_name']]
    header = ['content', 'category_name']
    new_tweet_df = pd.concat(data, axis = 1, keys = header)

    location = input("Please enter the path where you want to save the file: ")

    fname = input("Please enter the file name: ")

    #saving the dataframe as csv
    rd_tweets_df.to_csv(location+fname)

    return rd_tweets_df, new_tweet_df
```

Calling of the Function to Select the Specified Columns from the Database:

```
1 original_tweet_df, modified_tweet_df = load_data(stcd, stcc, mcl)
```

```
Please enter the path where you want to save the file: C:\\Users\Sagar\Project - Twitter
Please enter the file name: tweet_selected.csv
```

Dataframe with all the columns selected from the raw tweets:

```
1 original_tweet_df
```

	category_name	created_at	favorite_count	full_text	id	lang	possibly_sensitive	retweet_count	user.created_at	user.fa
0	Govt Institutions	Wed Mar 10 11:22:20 +0000 2021	0	@HugaraMaruthi @father_of_BJP @SachinP20999278...	1.36961E+18	en	FALSE	0	Fri Oct 23 23:34:39 +0000 2020	
1	Govt Institutions	Wed Mar 10 11:22:14 +0000 2021	0	@CRED_support I have done a rent payment from...	1.36961E+18	en	NaN	0	Sat Oct 28 01:24:49 +0000 2017	
2	Govt Institutions	Wed Mar 10 11:22:06 +0000 2021	0	There is a long time issue going on my credit ...	1.36961E+18	en	NaN	0	Wed Jun 15 16:45:15 +0000 2016	
3	Govt Institutions	Wed Mar 10 11:22:02 +0000 2021	0	Senior RBI officer "We have made aware to the ...	1.36961E+18	en	FALSE	0	Mon Mar 10 14:31:11 +0000 2014	

Dataframe containing the tweets and the category name:

```
1 modified_tweet_df
```

	content	category_name
0	@HugaraMaruthi @father_of_BJP @SachinP20999278...	Govt Institutions
1	@CRED_support I have done a rent payment from...	Govt Institutions
2	There is a long time issue going on my credit ...	Govt Institutions
3	Senior RBI officer "We have made aware to the ...	Govt Institutions
4	@ICICIBank @ICICIBank_Care wasn't interest not...	Govt Institutions
...	...	...
216170	I used to love playing this game! ☺ #PrinceOf...	Game
216171	Release a proper #PrinceOfPersia remake #Ubiso...	Game
216172	My workplace is ready for #PrinceOfPersia\nhtt...	Game
216173	Cosplay Character Cosplayer\n\...	Game
216174	@princeofpersia just got refunded my money hop...	Game

104706 rows × 2 columns

Loading Training Data:

The challenge to classify tweets to Troll and Not-Troll lead us to a conclusion that we would need a labelled dataset with existing classification of Troll and Not-Troll. We got a dataset from google containing 20,000 data marked as Troll and Not-Troll which was used to train our model before passing the tweets gathered using Twitter API.

The dataset was imported to MongoDB and then loaded into the program.

For the function we need to pass the **mongo_client_link** for the connectivity and the function returns a dataframe which was loaded to the program **training_df**.

```
def load_trainingData(mongo_client_link):
#Making the connection between python and MongoDB

    client = pymongo.MongoClient(mongo_client_link)

    #Connecting to datbase and collection of MongoDB
    tclass = client['troll_classification']
    tcollec = tclass['collection_troll']

    #printing the connection to the collection from MongoDB
    print(tcollec)

    #Loading the records to pandas dataframe excluding the auto-generated id by MongoDB
    training_df = pd.json_normalize(tcollec.find({},{'_id':0, 'extras':0, 'annotation.notes': 0 }))

    #creating an empty list to store the troll label that we would be extracted from the array
    troll_label = []

    for i in training_df['annotation.label']:
        for j in i:
            troll_label.append(j)
    training_df['Troll_label'] = troll_label

    training_df = training_df.drop(labels = 'annotation.label', axis=1)

    return training_df
```

Calling the Function to Load the Data from MongoDB to Pandas DataFrame:

```
1 training_df = load_trainingData(mcl)
```

```
Collection(Database(MongoClient(host=['127.0.0.1:27017'], document_class=dict, tz_aware=False, connect=True), 'troll_classification'), 'collection_troll')
```

The function displays the connection of the database.

```
1 training_df
```

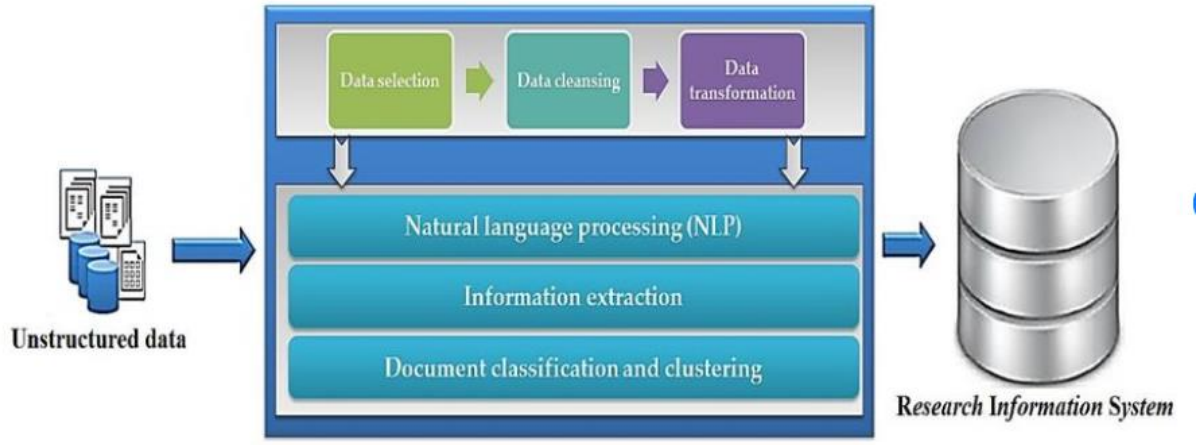
	content	Troll_label
0	She is as dirty as they come and that crook ...	1
1	why did you fuck it up. I could do it all day...	1
2	Dude they dont finish enclosing the fucking s...	1
3	WTF are you talking about Men? No men thats n...	1
4	Ill save you the trouble sister. Here comes a ...	1
...	...	...
19995	I dont. But what is complaining about it goi...	0
19996	Bahah yeah i&;m totally just gonna&; get pis...	0
19997	hahahahaha >:) im evil mwahahahahahahahahaha	0
19998	What&;s something unique about Ohio? :)	0
19999	Who is the biggest gossiper you know?	0

20000 rows × 2 columns

Data Pre-processing

The nltk package is used for Natural Language Processing (NLP).

Natural language processing (NLP) is an area of computer science and artificial intelligence concerned with the interactions between computers and human (natural) languages, how to program computers to process and analyse large amounts of natural language data. It is the branch of machine learning which is about analysing any text and handling predictive analysis.



The new research information process and lifecycle integrating new data processing layers (NLP tools, information extraction, document classification and clustering)

Feature Extraction:

Now that we have arrived at our training set, we need to extract useful features from it which can be used in the process of classification. But first we will discuss some text formatting techniques which will aid us in feature extraction:

1. **Tokenization:** It is the process of breaking a stream of text up into words, symbols and other meaningful elements called "tokens". Tokens can be separated by whitespace characters and/or punctuation characters. It is done so that we can look at tokens as individual components that make up a tweet.
2. **URL, hashtags, mentions emojis and smiley removal:** URL's and user references (identified by tokens "http" and "@") are removed if we are interested in only analysing the text of the tweet. There are few other things that are included in a tweet such as hashtags, mentions, emojis and smileys. We had removed these from the tweets to get the content of the tweet.
3. **Lowercase conversion:** Tweet may be normalized by converting it to lowercase which makes its comparison with an English dictionary easier.
4. **Abbreviation of the words:** In social media we have seen that the communication is abbreviated. This is more prominent on twitter as it has a limit that the user can type in a tweet. So, we had to create a dictionary containing few of those words that we found on daily basis such as LOL, ROFL, what's, dunno, wanna, etc.
5. **Punctuation, special characters and numbers removal:** Punctuation marks and digits/numerals may be removed if for example we wish to compare the tweet to a list of English words.

6. **Stop-words removal:** Stop words are class of some extremely common words which hold no additional information when used in a text and are thus claimed to be useless. Examples include “a”, “an”, “the”, “he”, “she”, “by”, “on”, etc. It is sometimes convenient to remove these words because they hold no additional information since they are used almost equally in all classes of text.
7. **Lemmatization:** Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item.

```
abbr_dict={"dunno": "do not know", "wanna": "want to", "what's":"what is", "what're":"what are","who's":"who is","who're":"w
    "where're":"where are","when's":"when is","when're":"when are","how's":"how is","how're":"how are",
    "i'm":"i am","we're":"we are","you're":"you are","they're":"they are","it's":"it is","he's":"he is",
    "she's":"she is","that's":"that is","there's":"there is","there're":"there are","i've":"i have","we've":"we have",
    "you've":"you have","they've":"they have","who've":"who have","would've":"would have","not've":"not have",
    "i'll":"i will","we'll":"we will","you'll":"you will","he'll":"he will","she'll":"she will",
    "it'll":"it will","they'll":"they will","I'll":"i will","isn't":"is not","wasn't":"was not","aren't":"are not","weren't"
    "can't":"can not","couldn't":"could not","don't":"do not","didn't":"did not","shouldn't":"should not",
    "wouldn't":"would not","doesn't":"does not","haven't":"have not","hasn't":"has not","hadn't":"had not",
    "won't":"will not","u":"you","ur":"your", "rolf": "rolling on floor laughing", "stfu": "shut the fuck up",
    "icymi": "in case you missed it", "tl;dr": "too long, didn’t read", "lmk": "let me know","nvm": "nevermind",
    "tgif": "thank goodness it’s Friday", "tbh": "to be honest", "tbf": "to be frank", "rn": "right now",
    "qotd": "quote of the day", "brb": "be right back", "btw": "by the way", "lol": "laugh out loud",
    "ttyl": "talk to you later", "hmu": "hit me up", "fwiw": "for what it’s worth",
    "imo": "in my opinion", "imho": "in my humble opinion", "idk": "i do not know", "tba": "to be announced",
    "tbd": "to be decided", "faq": "frequently asked question", "asap": "as soon as possible",
    "aka": "also known as", "diy": "do it yourself", "np": "o problem", "ty": "thank you", "hifw": "how i feel when",
    "bts": "behind the scenes", "cmv": "change my view", "dyk": "did you know", "eli5" : "explain it to me like i am five",
    "ftw": "for the win", "irl": "in real life", "nbd": "no big deal", "oc": "original content", "tftf": "thanks for the fol
    "tfw": "that feeling when", "tigf": "thank god it is friday", "f*ck": "fuck", "f***k": "fuck", "s**k": "suck",
    "b***h": "bitch", "b**ch": "bitch", "a**": "ass", "a**h*le": "asshole", "fu*k": "fuck", "sh*t": "shit", "s**t": "shit",
    "omg": "oh my god", "ily": "i love you", "lmao": "laughing my ass off", "wtf": "what the fuck", "ppl": "people",
    "thx": "thanks", "ffs": "for fuck's sake", "fml": "fuck my life", "stfu": "shut the fuck up", "jj": "just joking",
    "jk": "just kidding", "bff": "best friend forever", "ftw": "for the win", "txt": "text", "hbd": "happy birthday",
    "gtfo": "get the fuck out", "dgaf": "do not give a fuck", "dtf": "down to fuck", "smfh": "shaking my fucking head",
    "roflmao": "rolling on floor laughing my ass off", "ptfo": "passed the fuck out", "ttys": "talk to you soon",
    "fbo": "facebook official", "ttyn": "talk to you never", "b4": "before", "bae": "before anyone else", "btaim": "be that
    "cx": "customer experience", "dm": "direct message", "f2f": "face to face", "b2b": "business to business",
    "b2c": "business to customer", "fb": "facebook", "ftfy": "fixed that for you", "g2g": "got to go", "gr8": "great",
    "hmb": "hit me back", "hmu": "hit me up", "hth": "happy to help", "ianad": "i am not a doctor", "ianal": "i am not a law
    "idc": "i do not care", "ig": "instagram", "rss": "really simple syndication", "rt": "retweet", "motherf**ker": "motherf
    "motherfu*cker": "motherfucker", "'em": "them", "ik": "i know", "what&;s":"what is", "what&;re":"what are",
    "who&;s":"who is","who&;re":"who are","where&;s":"where is",
    "where&;re":"where are","when&;s":"when is","when&;re":"when are","how&;s":"how is","how&;re":"how are",
    "i&;m":"i am","we&;re":"we are","you&;re":"you are","they&;re":"they are","it&;s":"it is","he&;s":"he is",
    "she&;s":"she is","that&;s":"that is","there&;s":"there is","there&;re":"there are","i&;ve":"i have","we&;ve":"we have",
    "you&;ve":"you have","they&;ve":"they have","who&;ve":"who have","would&;ve":"would have","not&;ve":"not have",
    "i&;ll":"i will","we&;ll":"we will","you&;ll":"you will","he&;ll":"he will","she&;ll":"she will",
    "it&;ll":"it will","they&;ll":"they will","I&;ll":"i will","isn&;t":"is not","wasn&;t":"was not",
    "aren&;t":"are not","weren&;t":"were not",
    "can&;t":"can not","couldn&;t":"could not","don&;t":"do not","didn&;t":"did not","shouldn&;t":"should not",
    "wouldn&;t":"would not","doesn&;t":"does not","haven&;t":"have not","hasn&;t":"has not","hadn&;t":"had not",
    "won&;t":"will not", "gonna": "got to", "gotcha": "i have got you", "d": "the", "n": "and", "amp": "and"}
```

```
#function to preprocess the text data
def nlp_preprocessing(dataframe):

    #Removing of URL, Mentions, Hastages, Reserved Words (RT and FAV), Emoji, Smiley and Number
    p.set_options(p.OPT.URL, p.OPT.MENTION, p.OPT.HASHTAG, p.OPT.RESERVED, p.OPT.EMOJI, p.OPT.SMILEY, p.OPT.NUMBER)
    dataframe['content'] = dataframe['content'].apply(lambda x: " ".join(p.clean(x) for x in x.split()))

    #converting the contents in lower case
    dataframe['content'] = dataframe['content'].apply(lambda x: " ".join(x.lower() for x in x.split()))

    #normalizing short words
    dataframe['content'] = dataframe['content'].apply(lambda x: ' '.join([abbr_dict[x] if x in abbr_dict else x for x in x.s

    #removing any character which is not alphabets from the string
    dataframe['content'] = dataframe['content'].apply(lambda x: " ".join(x for x in x.split() if x.isalpha()))

    #removing the stopwords from the contents
    stop = stopwords.words('english')
    dataframe['content'] = dataframe['content'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))

    #Lemmatization of the words from the content
    dataframe['content'] = dataframe['content'].apply(lambda x: " ".join([Word(word).lemmatize() for word in x.split()]))

    return dataframe
```

The function works on pre-processing the data on the column that was mentioned on inside the function. Each tweet that are available in the dataframe are cleaned on row basis. The function takes the **dataframe** as a parameter and returns the **cleaned dataframe**.

Calling the function to clean the dataset:

```
1 clean_training_df = nlp_preprocessing(training_df)
```

```
1 clean_training_df
```

	content	Troll_label
0	dirty come crook rengel dems fucking corrupt m...	1
1	fuck could day ping later sched writing book	1
2	dude dont finish enclosing fucking hate half a...	1
3	fuck talking men thats menage	1
4	ill save trouble come big ol fuck france block...	1
...	...	...
19995	complaining going	0
19996	bahah yeah totally get pissed talking mhm that...	0
19997	hahahahaha im evil mwahahahahahahahahaha	0
19998	something unique	0
19999	biggest gossiper	0

20000 rows × 2 columns

Training and Testing Split:

The labelled dataset that was downloaded to with already defined labels for Troll and Not-Troll are split into training and validation of the model. The function takes dataframe as a parameter and returns the train_x, train_y and valid_x, valid_y

```
def train_test_split(dataframe):

    #randomizing the dataframe contents
    dataframe = dataframe.sample(frac = 1)

    #splitting the data into train and test
    train_x, valid_x, train_y, valid_y = model_selection.train_test_split(dataframe['content'],
                                                                          dataframe['Troll_label'])

    return train_x, valid_x, train_y, valid_y
```

Calling the Function to Split the Data into Training and Validation:

```
train_x, valid_x, train_y, valid_y = train_test_split(clean_training_df)
```

Data Vectorization:

TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. In **Tfidf Vectorizer** we consider **overall document weightage** of a word. It helps us in dealing with most frequent words. Using it we can penalize them. Tfidf Vectorizer weights the word counts by a measure of how often they appear in the documents.

We have used Tfidf vectorizer to convert the text into vectors so, that the machine can learn how to interpret the data and lean using it.

The function takes dataframe, **train_x and valid_x** as input. The below function is to convert the training dataset into vectorization and returns vectorized train and valid data **(xtrain_tfidf and xvalid_tfidf)**.

```
def data_vectorizing(dataframe, train_x, valid_x):

    #Vectorizing the data using TF-IDF vectorizer
    tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=10000)
    tfidf_vect.fit(dataframe['content'])
    #transforming the training and testing data
    xtrain_tfidf =  tfidf_vect.transform(train_x)
    xvalid_tfidf =  tfidf_vect.transform(valid_x)

    return xtrain_tfidf, xvalid_tfidf
```

Calling the Function to Vectorize the Text:

```
xtrain_tfidf, xvalid_tfidf = data_vectorizing(clean_training_df, train_x, valid_x)
```

Function for Vectorizing of Testing Dataset:

The function is same as the above one with few modifications which will be used for the training dataset/tweets extracted from twitter. The dataframe needs to be passed to the function upon call. The function returns the vectorized data for the machine learning model **(tweet_tfidf)**.

```
def newdata_vectorizing(dataframe):

    #Vectorizing the data using TF-IDF vectorizer
    tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=10000)
    tfidf_vect.fit(dataframe['content'])
    #transforming the dataframe
    tweet_tfidf =  tfidf_vect.transform(dataframe['content'])

    return tweet_tfidf
```

Model Training & Testing

We have used 5 different algorithms/models to train our dataset. The technique that is used is called as bagging.

Below are the models that have been used to create our own bagging model:

1. MultinomialNB
2. DecisionTreeClassifier
3. RandomForestClassifier
4. SupportVectorClassifier (SVC)
5. PassiveAggressiveClassifier

To improve the overall performance of the classification **Ensembling** method is used. It combines the decisions from multiple models to improve the overall performance. This was done by using **VotingClassifier** which gets the vote from each of the models and the highest vote is then assigned to the tweet.

xtrain_tfidf, train_y, xvalid_tfidf, valid_y needs to be passed to the function as parameters. The function returns **ensemble and ensemble_fit**.

```
def custom_model_train_test(xtrain, ytrain, xtest, ytest):

    model = []

    multinomial = Pipeline([('m', naive_bayes.MultinomialNB(alpha=0.2))])
    model.append(('multinomial', multinomial))

    decision_tree = Pipeline([('m', DecisionTreeClassifier(random_state = 2))])
    model.append(('decisiontree', decision_tree))

    random_forest = Pipeline([('m', RandomForestClassifier())])
    model.append(('randomforest', random_forest))

    svc = Pipeline([('m', svm.SVC())])
    model.append(('svc', svc))

    passive_aggressive = Pipeline([('m', linear_model.PassiveAggressiveClassifier(C = 0.5, random_state = 5))])
    model.append(('passiveaggressive', passive_aggressive))

    ensemble = VotingClassifier(estimators = model, voting = 'hard')

    ensemble_fit = ensemble.fit(xtrain, ytrain)

    predictions = ensemble.predict(xtest)

    print("Prediction: ", predictions)

    print("Model Accuracy: ", accuracy_score(ytest, predictions)*100)

    print("Confusion Matrix: \n", confusion_matrix(ytest, predictions))

    return ensemble, ensemble_fit
```

Calling the function to Train and Validate the Machine Learning Model:

```
1 ensemble, ensemble_fit = custom_model_train_test(xtrain_tfidf, train_y, xvalid_tfidf, valid_y)
```

```
Prediction:  ['1' '1' '0' ... '1' '1' '0']
Model Accuracy:  88.82
Confusion Matrix:
 [[2712  305]
 [ 254 1729]]
```

After multiple testing we have seen that the model accuracy ranges between 88.00 and 90.90.

Applying Trained Model to the Extracted Data

The trained model is then applied upon extracted data to label the tweets as Troll or Non-troll.

```
#Testing new dataset extracted from twitter

def new_data_test(ensemble, ensemble_fit, xtest):

    #model fit values
    ensemble_fit

    #predicting if the tweet is troll or not
    predictions = ensemble.predict(xtest)

    return predictions
```

Before we can test the model with the extracted tweets, we need to follow the same steps that we did for pre-processing of the training dataset. We have to follow the below steps:

1. **NLP Pre-processing:** removal of URLs, emojis, smileys, punctuation, converting to lower case, abbreviation removal, stop-word removal and lemmatization of the words by calling the appropriate function.

```
modified_clean_tweet_df = nlp_preprocessing(modified_tweet_df)
```

2. **Vectorizing:** the testing dataset using TFIDF Vectorizer, by calling the appropriate function.

```
tweet_tfidf = newdata_vectorizing(modified_clean_tweet_df)
```

Now, we can run the model to classify the tweets into Troll and Not-Troll by calling the function.

The function takes the **ensemble, ensemble_fit and vectorized data** as the parameters and returns the prediction of the data which is in the form of 0 and 1. Where **0 signifies Not-Troll** and **1 signifies Troll**.

```
1 predictions = new_data_test(ensemble, ensemble_fit, tweet_tfidf)
```

```
1 predictions
```

```
array(['0', '0', '0', ..., '0', '1', '0'], dtype=object)
```

Sentiment Polarity

sentiment score for this sector wise tweets is calculated in order to draw meaningful insights.

In this the function takes the dataframe as parameter and returns **sentiment_val and polarity_score.**

```
def sentiment_analysis(dataframe):

    #SentimentIntensityAnalyzer initialization
    analyser = SentimentIntensityAnalyzer()

    #creating an empty list to store the sentiment of the tweets
    sentiment_val = []
    polarity_score = []
    #extracting the tweets from the content column of the dataframe
    for senti in dataframe['content']:

        #passing the tweets to the model to get the polarity score of the tweet
        sentiment_dict = analyser.polarity_scores(senti)
        polarity_score.append(sentiment_dict['compound'])

        # # decide sentiment as positive, negative and neutral on the basis of compound score
        if sentiment_dict['compound'] >= 0.05 :
            val = 'Positive'
            sentiment_val.append(val)

        elif sentiment_dict['compound'] <= - 0.05 :
            val = 'Negative'
            sentiment_val.append(val)

        else :
            val = 'Neutral'
            sentiment_val.append(val)

    return sentiment_val, polarity_score
```

Calling the Function to Get Sentiment Value and Polarity Score of the Tweets:

```
sentiment_val, sentiment_polarity = sentiment_analysis(modified_clean_tweet_df)
print("Sentiment Value: ", sentiment_val)
```

```
ve', 'Positive', 'Positive', 'Positive', 'Positive', 'Positive', 'Negative', 'Positive', 'Neutral', 'Negative', 'Negative',
'Negative', 'Negative', 'Positive', 'Positive', 'Neutral', 'Negative', 'Positive', 'Positive', 'Negative', 'Positive', 'Negat
ive', 'Positive', 'Positive', 'Negative', 'Positive', 'Negative', 'Positive', 'Positive', 'Negative', 'Neutral', 'Negative',
'Negative', 'Negative', 'Neutral', 'Negative', 'Negative', 'Negative', 'Negative', 'Positive', 'Positive', 'Negative', 'Neutr
al', 'Positive', 'Positive', 'Neutral', 'Neutral', 'Positive', 'Negative', 'Neutral', 'Negative', 'Positive', 'Negative', 'Ne
utral', 'Negative', 'Negative', 'Neutral', 'Negative', 'Negative', 'Positive', 'Negative', 'Negative', 'Positive', 'Positiv
```

```
print("\nSentiment Polarity: ", sentiment_polarity)
```

```
Sentiment Polarity:  [0.3818, 0.4019, 0.3612, -0.1779, 0.91, 0.0, 0.0, -0.4404, 0.0, -0.4939, 0.0772, 0.0, 0.7003, -0.9201,
0.1027, 0.0, 0.5859, -0.6808, 0.9578, -0.4939, 0.3818, 0.743, 0.3818, 0.3818, -0.9201, 0.0, -0.7783, 0.0, 0.3818, 0.4939, 0.
0, -0.1779, 0.3818, 0.0, -0.296, -0.3818, 0.3612, 0.0, 0.5509, 0.0, -0.4215, 0.5423, -0.4404, -0.5267, 0.0, -0.4588, -0.7506,
0.4019, 0.5719, 0.2023, 0.3182, 0.6597, -0.1027, 0.0, 0.6369, 0.0, -0.128, 0.0258, 0.7906, 0.4404, -0.4215, 0.0, 0.079, 0.0,
0.0, 0.0, 0.7351, 0.0772, 0.3818, 0.0772, 0.4404, 0.0, 0.0, 0.3182, 0.0, 0.7579, 0.0, 0.3818, -0.8074, 0.6124, 0.5859, 0.757
9, -0.7003, -0.296, -0.5994, 0.0, 0.7579, 0.7351, 0.0, -0.2263, -0.4588, 0.0, 0.0, -0.4215, 0.3818, -0.5994, 0.4215, 0.0, 0.
0, 0.0, -0.2023, 0.2681, -0.0772, 0.3818, 0.0, 0.3818, 0.0, -0.8885, 0.0, -0.4767, 0.25, 0.5994, 0.5859, 0.0, -0.6249, 0.051
6, -0.6808, 0.3818, 0.0, 0.0, -0.2263, -0.7506, -0.7717, -0.4767, -0.7717, -0.7713, -0.296, 0.5994, 0.8402, 0.0, 0.3818, 0.2
5, -0.8074, 0.5859, 0.7783, -0.8655, 0.8555, -0.9423, 0.5859, 0.0, 0.6486, 0.0, 0.0, 0.5106, 0.296, -0.296, 0.4215, 0.0772,
0.296, -0.3818, -0.8689, 0.0, 0.0, 0.0, 0.296, 0.0, 0.3182, 0.0, 0.0, 0.0, 0.0, 0.2023, 0.8555, -0.0516, 0.0, 0.0, -0.128, 0.
0, -0.5859, 0.1027, -0.128, 0.0, 0.0516, 0.0, 0.5574, -0.296, 0.0772, -0.1027, 0.5994, 0.2023, 0.0, 0.296, 0.0, 0.0, -0.1027,
```

We merge the above predictions, sentiment value and the sentiment polarity with the original dataframe.

```
original_tweet_df['Troll_label'] = predictions
original_tweet_df['Text_Sentiment'] = sentiment_val
original_tweet_df['Sentiment_Polarity'] = sentiment_polarity
```

The final dataframe now contains following fields:

```
list(original_tweet_df)
```

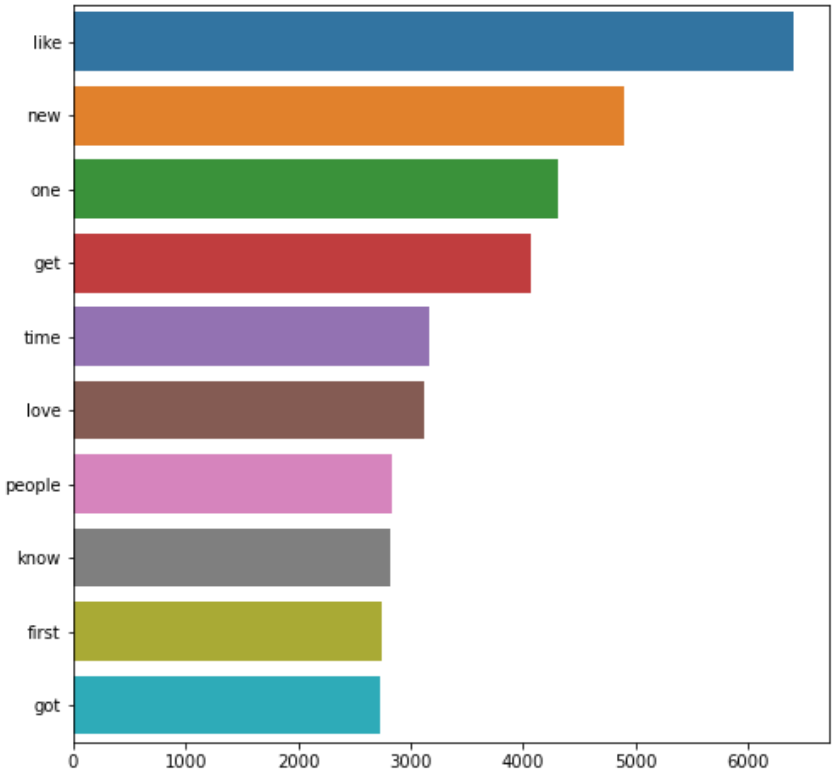
```
['category_name',
 'created_at',
 'favorite_count',
 'full_text',
 'id',
 'lang',
 'possibly_sensitive',
 'retweet_count',
 'user.created_at',
 'user.favourites_count',
 'user.followers_count',
 'user.friends_count',
 'user.id',
 'user.listed_count',
 'user.name',
 'user.protected',
 'user.screen_name',
 'user.statuses_count',
 'user.location',
 'retweeted_status.created_at',
 'retweeted_status.full_text',
 'retweeted_status.id',
 'entities.hashtags',
 'entities.user_mentions',
 'Troll_label',
 'Text_Sentiment',
 'Sentiment_Polarity']
```

EXPLORATORY DATA ANALYSIS

To list the top not stopwords in the tweets:

```
def top_not_stopwords_barplot(dataframe):

    stop = set(stopwords.words('english'))

    text = dataframe['content'].str.split()
    text = text.values.tolist()
    corpus = [word for i in text for word in i]

    counter = Counter(corpus)
    most = counter.most_common()

    words, counts = [], []

    for w,c in most[:10]:
        if (w not in stop):
            words.append(w)
            counts.append(c)

    plt.figure(figsize=(8,8))

    sns.barplot(x=counts,y=words)
```

```
top_not_stopwords_barplot(modified_clean_tweet_df)
```

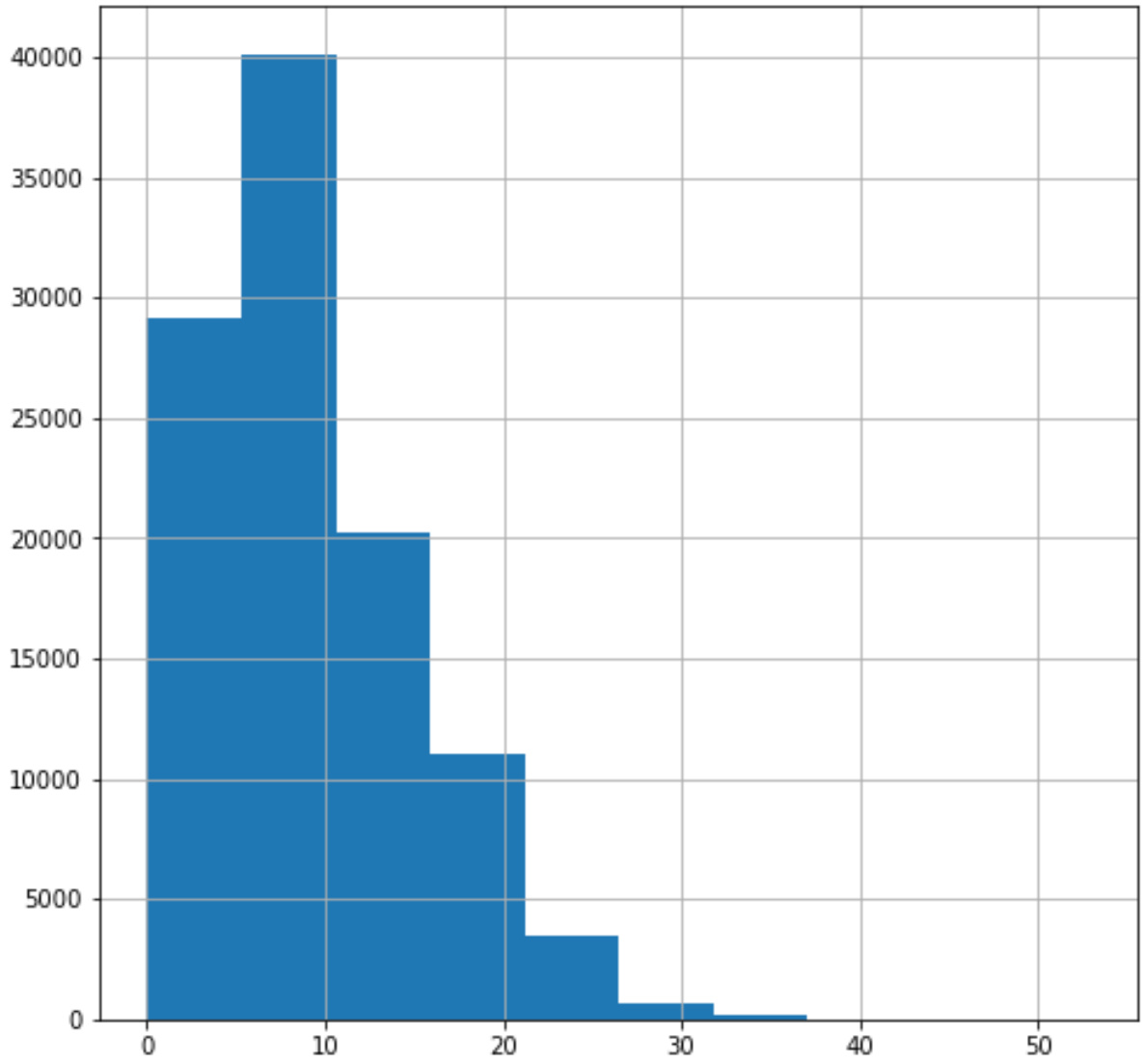


As we can see that like, new, one, get, time, etc., are the most common words.

To list the total number of words in the collected tweets

```
def word_number_histogram(dataframe):
     dataframe['content'].str.split().\
        map(lambda x: len(x)).\
        hist(figsize = (8,8))
```

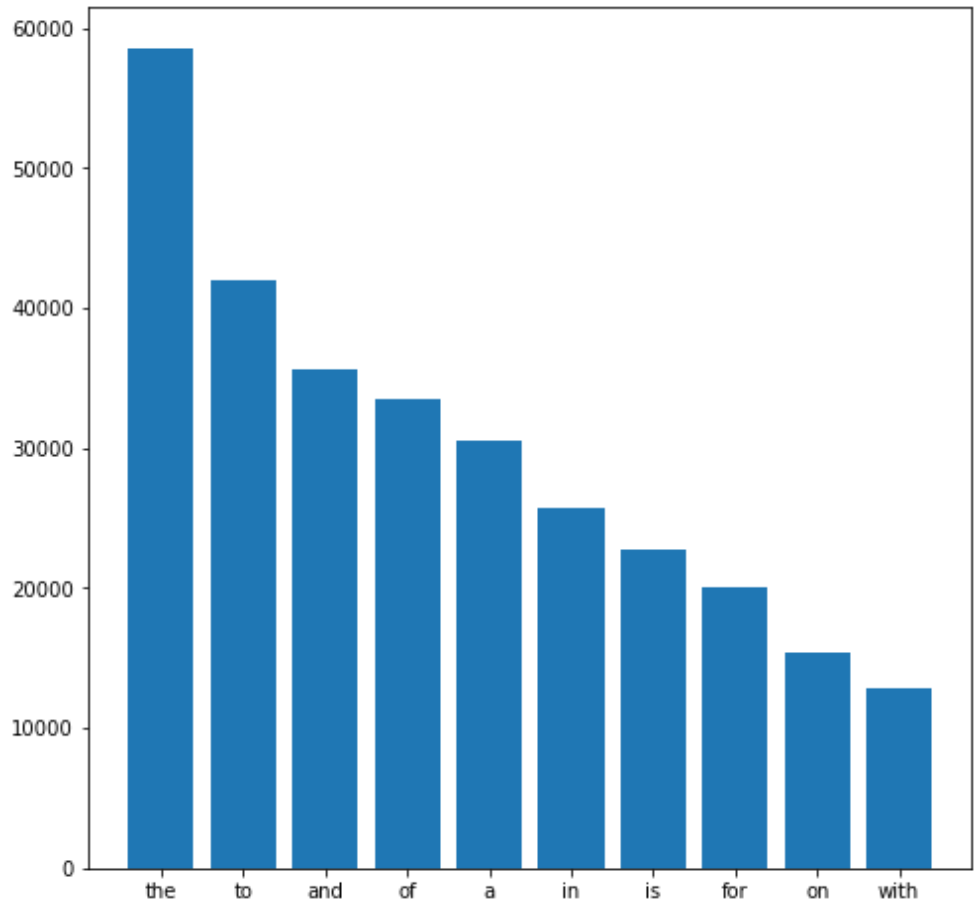
```
word_number_histogram(modified_clean_tweet_df)
#It is clear that the number of words in tweets ranges from 0 to 35
#and mostly falls between 0 to 11 words as per the below chart
```



To list the top stop-words used in the tweets:

```
def top_stopwords_barchart(dataframe):
    stopword = set(stopwords.words('english'))

    text = dataframe['full_text'].str.split()
    text = text.values.tolist()
    corpus = [word for i in text for word in i]

    from collections import defaultdict

    dic = defaultdict(int)

    for word in corpus:
        if word in stopword:
            dic[word]+=1

    top=sorted(dic.items(), key=lambda x:x[1],reverse=True)[:10]

    x,y=zip(*top)

    plt.subplots(figsize = (8,8))

    plt.bar(x,y)

    plt.show()
```

```
top_stopwords_barchart(original_tweet_df)

#As we can see from the below barplot that the most frequent 10 stopwords used are:
#"the, to, and, of, a, in, is, for, on and with" in the tweets
```



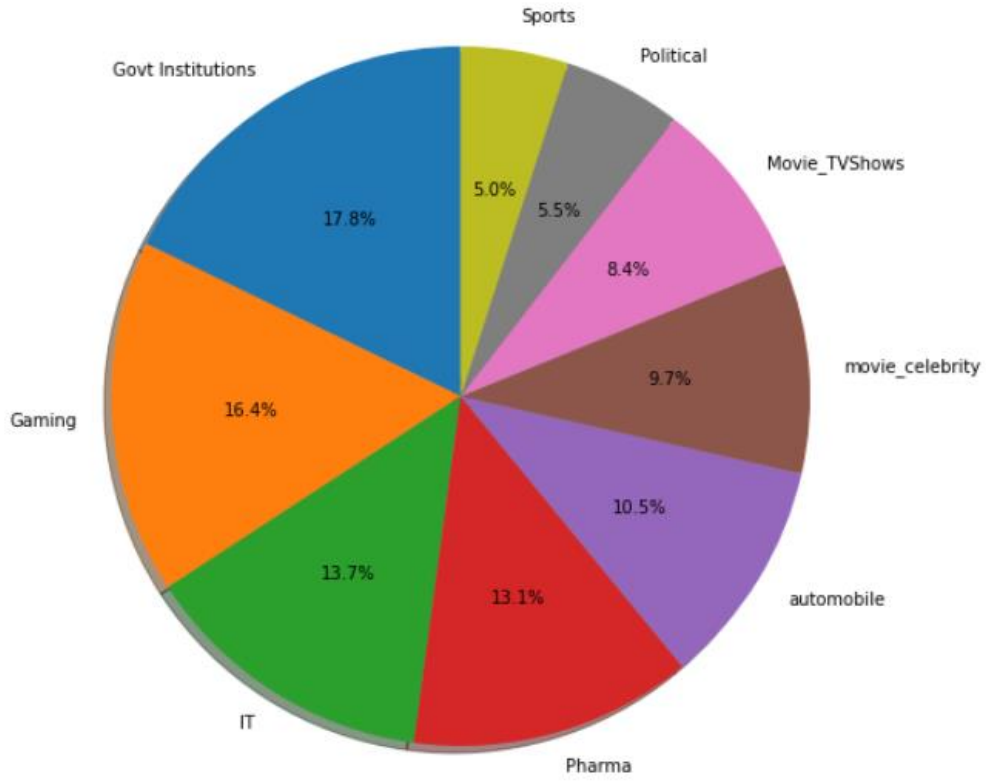
To list the sector wise tweet count:

```
# creating a dataframe to see the total number of category available
# the total number of tweets in each category

trct = original_tweet_df['category_name'].value_counts()

category_name = original_tweet_df['category_name'].unique()

records = []

for count in trct:
    records.append(count)

records

data = {'category': category_name, 'Total_Tweets': records}

trct_df = pd.DataFrame(data)

print("Total number of records in each category of tweets:\n\n", trct_df)
```

```
Total number of records in each category of tweets:

              category  Total_Tweets
0  Govt Institutions          18656
1             Gaming          17141
2                 IT          14302
3             Pharma          13698
4         automobile          10995
5    movie_celebrity          10159
6      Movie_TVShows           8783
7          Political           5755
8             Sports           5197
```

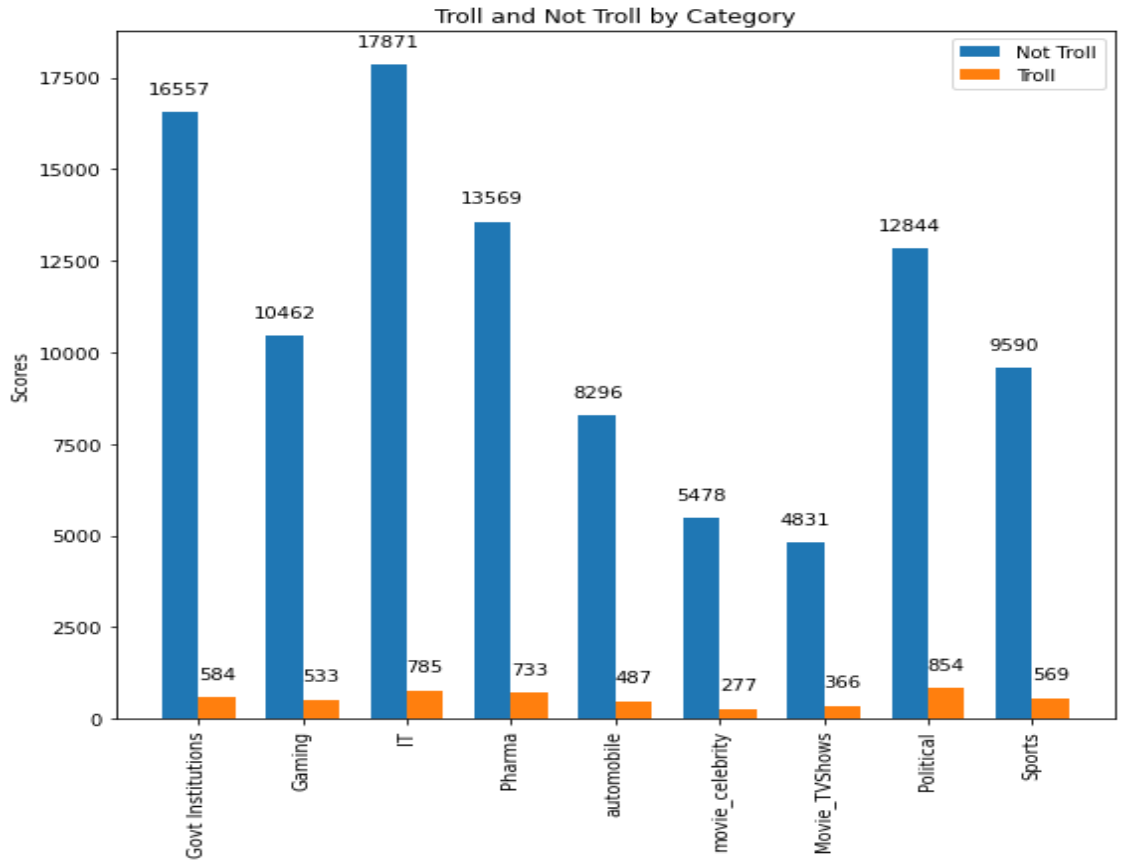
Sector wise Tweet percentage distribution:

```
fig1, ax1 = plt.subplots(figsize= (8,8))
ax1.pie(trct_df['Total_Tweets'], labels = trct_df['category'], autopct = '%1.1f%%', shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



Category wise Troll/Non-Troll distribution:

```
x = np.arange(len(tntcc['category']))

width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize = (8,8))
rects1 = ax.bar(x - width/2, tntcc['Not Troll'], width, label='Not Troll')
rects2 = ax.bar(x + width/2, tntcc['Troll'], width, label='Troll')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_title('Troll and Not Troll by Category')
plt.xticks(x, rotation=90)
ax.set_xticklabels(tntcc['category'])
ax.legend(fontsize = 10)

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 10),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


autolabel(rects1)
autolabel(rects2)

fig.tight_layout()

plt.show()
```



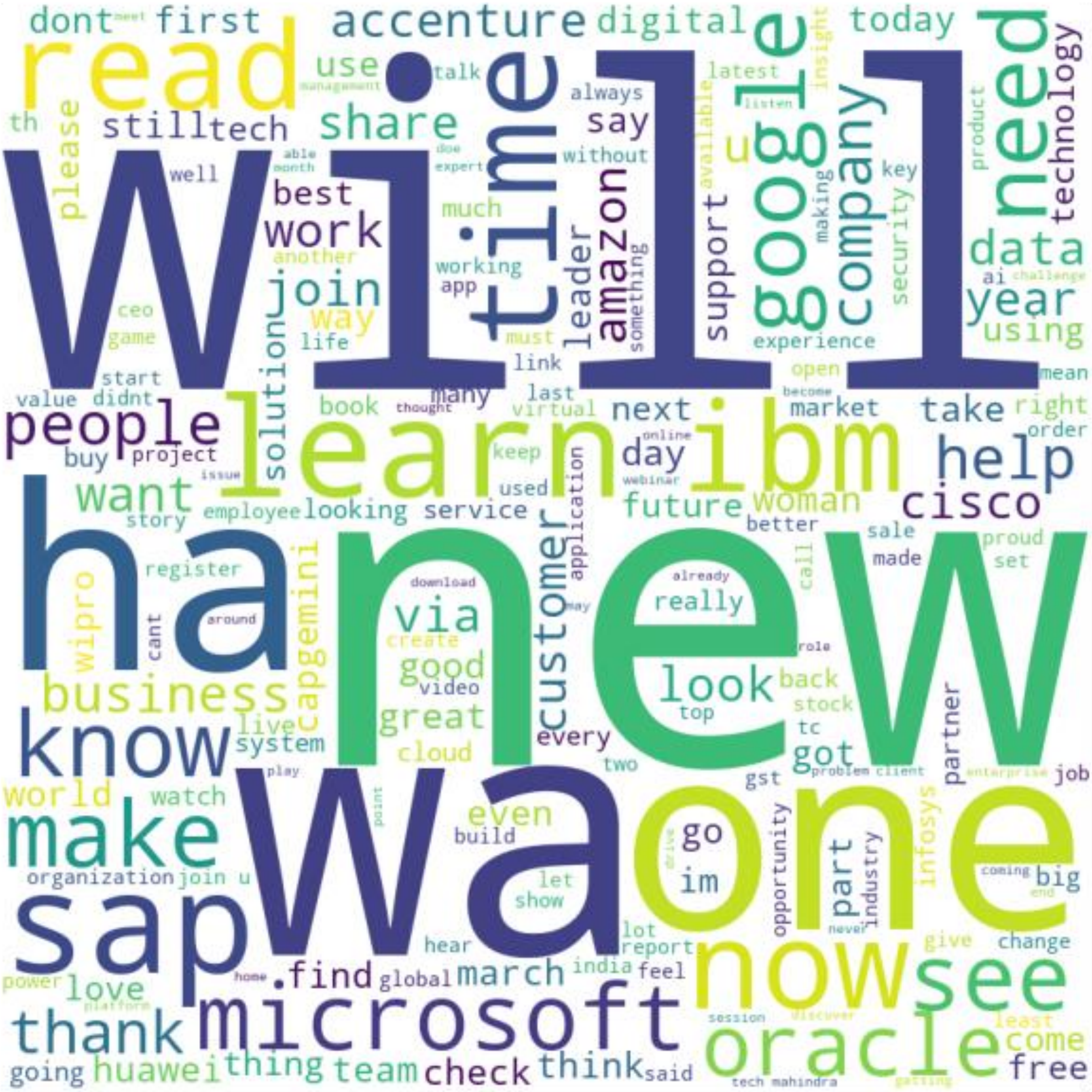
WordCloud:

From word cloud we can see which terms are most repeatedly used in a particula r sector.

1. Game

2. Govt Institutions

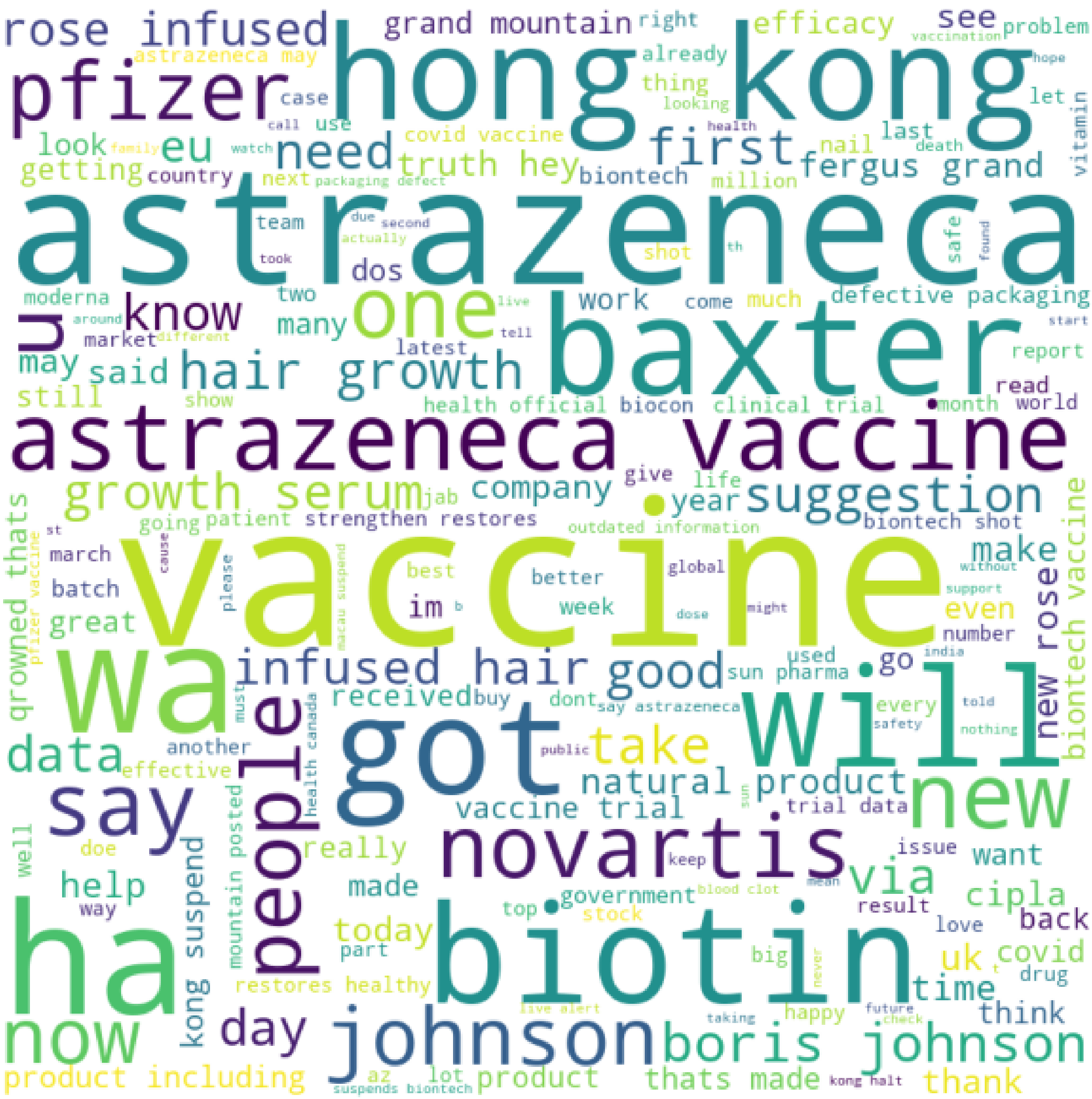
3. IT


will
new
wa
one
ha
read
learn
ibm
time
google
need
company
sap
microsoft
oracle
now
see
know
make
thank
people
business
accenture
digital
today
first
dont
share
still
tech
work
join
data
year
technology
amazon
support
customer
help
cisco
take
next
day
future
woman
look
via
good
great
cloud
capgemini
wipro
solution
want
best
say
use
always
latest
using
leader
infosys
partner
industry
part
opportunity
check
think
team
huawei
thing
love
find
march
global
going
free
come
change
big
job
world
watch
organization
system
register
employee
looking
service
market
right
order
mean
link
last
many
book
virtual
keep
used
application
really
back
stock
got
every
two
even
build
let
show
go
im
lot
report
feel
india
hear
give
said
project
buy
value
didnt
start
game
ceo
life
way
another
working
app
much
security
key
product
experience
open
please
well
th
say
without
talk
better
sale
made
proud
set
story
video
create
top
tc
gst
power
session

4. Movie_Celebrity

5. Movie_TVShows

6. Pharma


rose infused
grand mountain
right
efficacy
see
problem
pfizer
hong kong
astrazeneca may
case
already
thing
looking
let
look
eu
need
covid vaccine
first
nail
last
fergus grand
getting
country
next
truth hey
biontech
million
astrazeneca
team
dos
shot
safe
moderna
know
two
one
work
come
much
defective packaging
many
baxter
market
latest
report
may
said
hair growth
read
still
show
health official
biocon
clinical trial
month
world
astrazeneca vaccine
growth serum
jab
company
give
life
year
suggestion
going
patient
strengthen restores
outdated information
biontech shot
vaccine
march
batch
great
best
better
global
make
im
week
even
number
wa
infused hair
good
sun pharma
used
go
received
buy
dont
say astrazeneca
every
will
data
another
effective
got
take
natural product
new
say
people
vaccine trial
trial data
novartis
issue
want
help
made
government
result
cipla
back
today
part
biotin
love
ha
uk
covid
drug
restores healthy
time
think
now
day
johnson
boris johnson
happy
product including
lot
product
thats made
thank

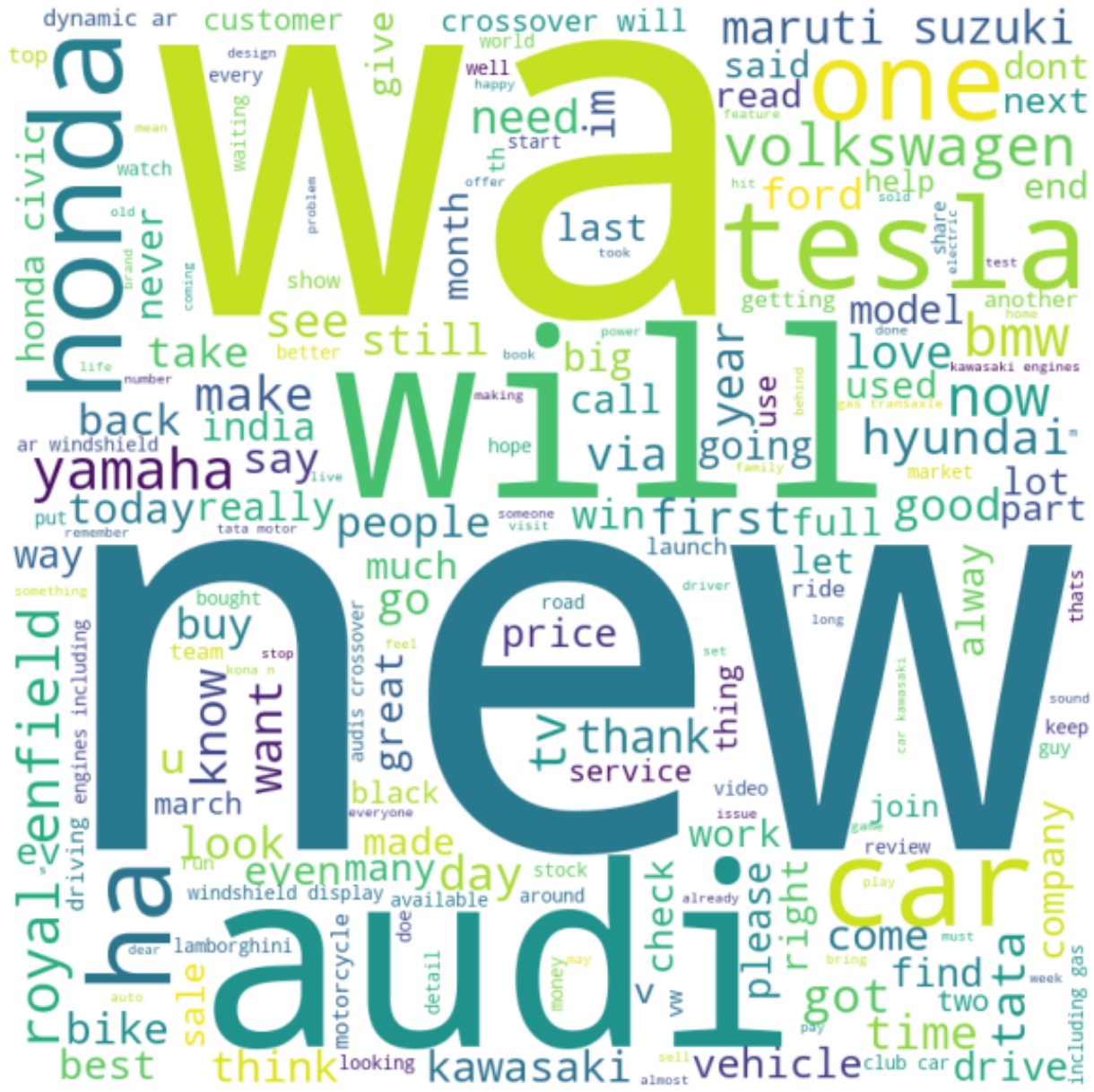
7. Political


vladimir putin
will
imran khan
rahul gandhi
boris johnson
hillary clinton
qasim dreams
nirmala sitharaman
muhammad qasim
ha
wa
say
india
hillary
now
time
attack
scott morrison
chief minister
yoshihide suga
narendra modi
joe biden
suga yoshihide
yogi adityanath
uddhav thackeray
arvind kejriwal
people
prime minister
government
finance minister
amit shah
fruitful conversation
pm modi
minister nirmala
minister imran
president vladimir
trump
first
new
want
election
party
know
cm uddhav
award go
pm
still
support
said
leader
congress
maharashtra cm
mamta banerjee
russia
trying
made
vaccine
called
got
state
perfect
world
many
woman
take
give
today
russian
actress award
going
doe
day
country
pakistan
best actor
pm narendra
rahul
every

8. Sports


novak djokovic
floyd mayweather
khabib nurmagomedov
virat kohli
lewis hamilton
saina nehwal
marc marquez
lionel messi
christiano ronaldo
happy birthday
roger federer
bernie ecclestone
dana white
rafael nadal
valentino rossi
black lives matter
parineeti chopra
logan paul
usain bolt
barcelona
miami open
rohit sharma
sania mirza
conor mcgregor
will
wa
one
floyd
world
time
india
player
back
win
match
return
england
ufc
people
make
think
virat
new
top
know
best
last
record
fight
look
tennis
year
ha
see
miami
big
beat
fan
want
got
take
next
say
race
coach
toss
end
riding
marquez will
nadal
family
better
read
day
still
via
go
going
said
single
now
game
motogp
greatest
roger
team
talk
watch
really
england open
title
set
kohli
season
good
man
retirement
indian
need
goal
won
play
even
second
join
great
final
father
honda
video
right
become
remember
career
wa offered
start
never
way

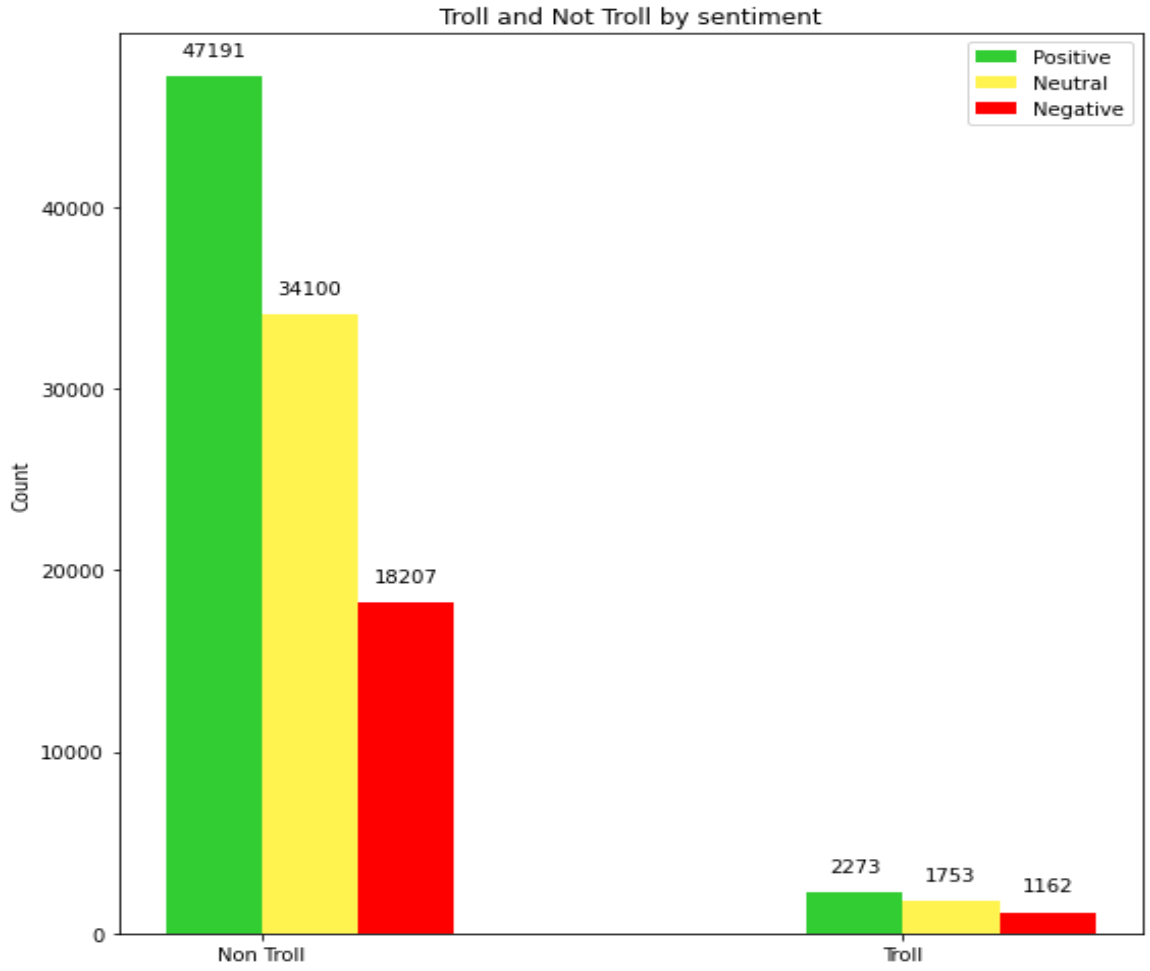
9. automobile



To investigate the relationship between tweet sentiment (Positive, Negative, Neutral) and tweet type (Troll/ Non-Troll):

```
#Checking Troll and Not Troll on the basis of Sentiment

ntts = original_tweet_df.loc[original_tweet_df['Troll_label'] == '0', 'Text_Sentiment']
tts = original_tweet_df.loc[original_tweet_df['Troll_label'] == '1', 'Text_Sentiment']

nt_pos, nt_neu, nt_neg, t_pos, t_neu, t_neg = [], [], [], [], [], []

Total_Not_Troll = tntcc['Not Troll'].sum()
Total_Troll = tntcc['Troll'].sum()

for i in ntts:
    if (i == 'Positive'):
        nt_pos.append(i)
    elif (i == 'Negative'):
        nt_neg.append(i)
    else:
        nt_neu.append(i)

for j in tts:
    if (j == 'Positive'):
        t_pos.append(j)
    elif (j == 'Negative'):
        t_neg.append(j)
    else:
        t_neu.append(j)

Troll_Pos = len(t_pos)
Troll_Neg = len(t_neg)
Troll_Neu = len(t_neu)
NTroll_Pos = len(nt_pos)
NTroll_Neg = len(nt_neg)
NTroll_Neu = len(nt_neu)

Pos = [NTroll_Pos, Troll_Pos]
Neg = [NTroll_Neg, Troll_Neg]
Neu = [NTroll_Neu, Troll_Neu]
```

```
#Plotting of graph of the above data
x = np.arange(len(Pos))

width = 0.15  # the width of the bars

fig, ax = plt.subplots(figsize = (8,8))
rects1 = ax.bar(x - width/2, Pos, width, label='Positive', color = '#32cd32') # #32cd32 - Lime Green
rects2 = ax.bar(x + width/2, Neu, width, label='Neutral', color = '#fff44f') # #fff44f - Lime Yellow
rects3 = ax.bar(x + width*3/2, Neg, width, label='Negative', color = '#FF0000') # #FF0000 - Red

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Count')
ax.set_title('Troll and Not Troll by sentiment')
plt.xticks(x)
ax.set_xticklabels(['Non Troll', 'Troll'])
ax.legend(fontsize = 10)

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 10),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


autolabel(rects1)
autolabel(rects2)
autolabel(rects3)

fig.tight_layout()

plt.show()
```



From the bar plot it is clear that, even positive sentiment tweet can be a troll and vice a versa.

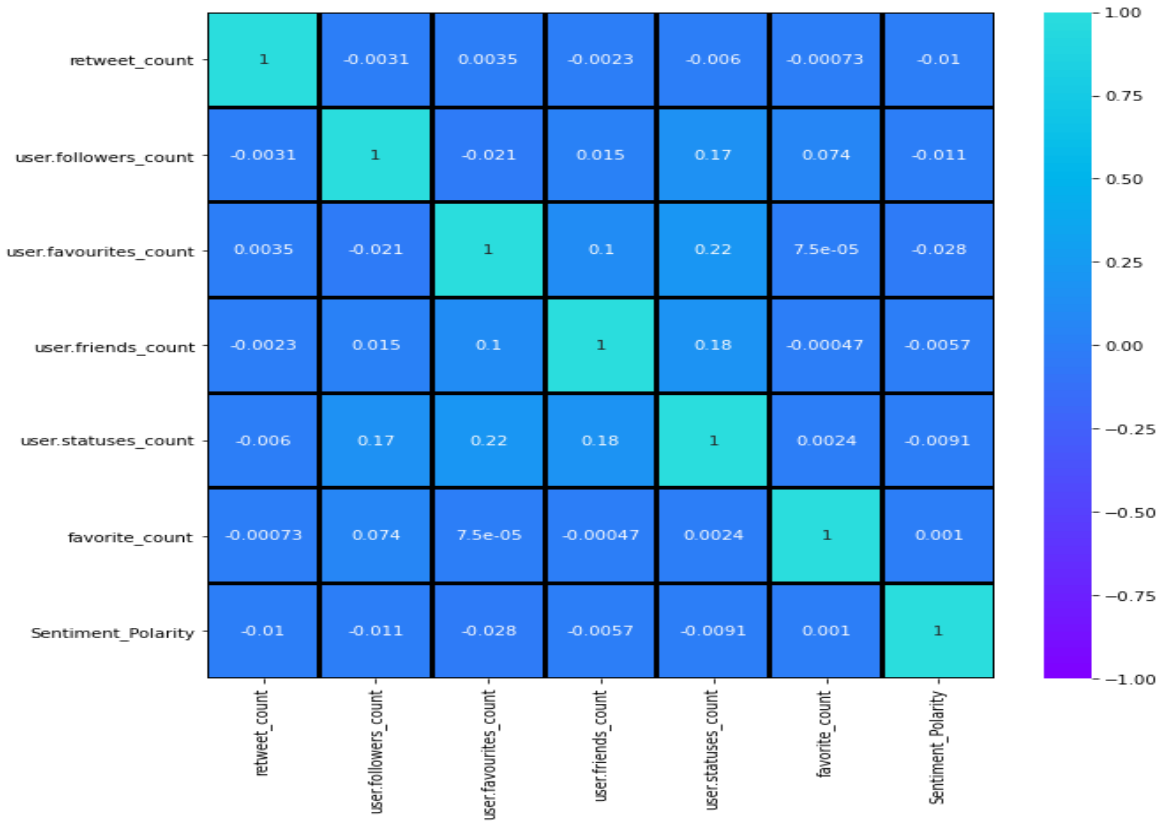
User Tweets study

```
#Getting the statistical description of the dataset
newDf1.to_csv("selected_data.csv")
newDf1.describe().round(decimals = 2)
```

	Troll_label	retweet_count	user.followers_count	user.favourites_count	user.friends_count	user.statuses_count	favorite_count	Sentiment_Polarity
count	104686.00	104686.00	104686.00	104686.00	104686.00	104686.00	104686.00	104686.00
mean	0.05	164.10	35300.52	21607.96	1361.09	51743.66	4.27	0.16
std	0.22	3468.79	483016.85	56788.53	7304.71	166673.69	77.09	0.41
min	0.00	0.00	0.00	0.00	0.00	1.00	0.00	-0.99
25%	0.00	0.00	71.00	297.00	107.00	1472.00	0.00	0.00
50%	0.00	0.00	347.00	3341.50	358.00	8079.00	0.00	0.00
75%	0.00	2.00	1552.00	17802.50	994.00	34056.00	1.00	0.49
max	1.00	594467.00	49674982.00	1377054.00	880258.00	6181855.00	9351.00	1.00

Correlation Matrix

In order to see the relationship between Sentiment of the tweet and other independent variables like retweet count, User followers count, User favourite count, User friend count, User statuses count, Favourite count pair wise correlation coefficients are calculated.



From correlation matrix we can say that there is weak correlation between the variables.

Inferential Statistics

Chi-Square Test for Association Between Sectors and Trolling

To test the association between Sectors and Trolling

The hypotheses are:

H0: There is no association between Sector and tweet type. Vs

H1: There is significant association between sectors and tweet type.

```
#Objective: Chi-Square test of association. To test whether association between trolling and categories

df_association = tntcc.iloc[:, :3]

Nt = []
T = []
for nt in df_association['Not Troll']:
    Nt.append(nt)

for t in df_association['Troll']:
    T.append(t)

NT_T_list = [Nt, T]

stat, p, dof, expected = chi2_contingency(NT_T_list)

print("Statistics: ", stat, "\np-Value: ", p, "\nDegree of Freedom: ", dof, "\nExpected Frequencies: \n", expected)
```

Output:

```
Statistics:  221.88689409346236
p-Value:  1.5375261050397052e-43
Degree of Freedom:  8
Expected Frequencies:
 [[16291.53103567 10450.11281356 17731.45108228 13593.22541696
   8347.73450127  5469.79529259  4939.44850314 13019.15828286
   9655.54307166]
 [  849.46896433   544.88718644   924.54891772   708.77458304
    435.26549873   285.20470741   257.55149686   678.84171714
    503.45692834]]
```

p-value for test of association is less than 0.05. Thus, H0 is rejected. It may conclude that there exists significant association between sectors and tweet type (Troll/ Not Troll).

T-Test for Equality of Two Population Mean:

To test whether average retweet count for Troll and Not troll tweets is different.

Test hypotheses are,

H0: The average retweet count for troll and non-troll tweets are equal. Vs

H1: The average retweet count for troll and non-troll tweets are not equal.

```
#Objective: To test whether retweet count for troll and non troll tweets are equal

# seed the random number generator
seed(1)

# generate two independent samples

data1 = newDf1.loc[newDf1['Troll_label'] == 0, 'retweet_count']

data2 = newDf1.loc[newDf1['Troll_label'] == 1, 'retweet_count']

# compare samples

stat, p = ttest_ind(data1, data2)

print('t = %.3f, p = %.3f' % (stat, p))
```

Output:

```
t = 0.303, p = 0.762
```

Since P-value is greater than 0.05. Thus, at 5% level of significance it may be concluded that the average retweet count for troll and non-troll tweets is equal.

One-way anova

Sector Wise Significance of User Favourite Count

To test whether sector wise user favourite count differ significantly.

H0: There is no significant difference among the sectors.

H1: There is significant difference among the sectors.

```
#Category wise retweet count
#Objective: To test whether category wise favourite count differ significantly

Political_RT1 = newDf1.loc[newDf1['category_name'] == 'Political', 'user.favourites_count']
GI_RT1 = newDf1.loc[newDf1['category_name'] == 'Govt Institutions', 'user.favourites_count']
Gaming_RT1 = newDf1.loc[newDf1['category_name'] == 'Gaming', 'user.favourites_count']
IT_RT1 = newDf1.loc[newDf1['category_name'] == 'IT', 'user.favourites_count']
Pharma_RT1 = newDf1.loc[newDf1['category_name'] == 'Pharma', 'user.favourites_count']
Automobile_RT1 = newDf1.loc[newDf1['category_name'] == 'automobile', 'user.favourites_count']
MC_RT1 = newDf1.loc[newDf1['category_name'] == 'movie_celebrity', 'user.favourites_count']
MTS_RT1 = newDf1.loc[newDf1['category_name'] == 'Movie_TVShows', 'user.favourites_count']
Sports_RT1 = newDf1.loc[newDf1['category_name'] == 'Sports', 'user.favourites_count']

f_oneway(Political_RT1, GI_RT1, Gaming_RT1, IT_RT1, Pharma_RT1, Automobile_RT1, MC_RT1, MTS_RT1, Sports_RT1)
```

Output:

```
F_onewayResult(statistic=113.18721162547179, pvalue=2.514078292814735e-189)
```

P value is less than 0.05. Thus, H0 is rejected at 5% level of significance. It may be concluded that the sector wise user favourite count differs significantly.

Sector Wise Significance of User Follower Count

To test whether sector wise user follower count differ significantly.

H0: There is no significant difference among the sectors in terms of follower count.

H1: There is significant difference among the sectors in terms of follower count.

```
#Category wise retweet count
#Objective: To test whether category wise followers count differ significantly

Political_RT1 = newDf1.loc[newDf1['category_name'] == 'Political', 'user.followers_count']
GI_RT1 = newDf1.loc[newDf1['category_name'] == 'Govt Institutions', 'user.followers_count']
Gaming_RT1 = newDf1.loc[newDf1['category_name'] == 'Gaming', 'user.followers_count']
IT_RT1 = newDf1.loc[newDf1['category_name'] == 'IT', 'user.followers_count']
Pharma_RT1 = newDf1.loc[newDf1['category_name'] == 'Pharma', 'user.followers_count']
Automobile_RT1 = newDf1.loc[newDf1['category_name'] == 'automobile', 'user.followers_count']
MC_RT1 = newDf1.loc[newDf1['category_name'] == 'movie_celebrity', 'user.followers_count']
MTS_RT1 = newDf1.loc[newDf1['category_name'] == 'Movie_TVShows', 'user.followers_count']
Sports_RT1 = newDf1.loc[newDf1['category_name'] == 'Sports', 'user.followers_count']

f_oneway(Political_RT1, GI_RT1, Gaming_RT1, IT_RT1, Pharma_RT1, Automobile_RT1, MC_RT1, MTS_RT1, Sports_RT1)
```

Output:

```
F_onewayResult(statistic=41.44148490983613, pvalue=1.0154988491982054e-66)
```

P value is less than 0.05. Thus, H0 is rejected at 5% level of significance. It may be concluded that the sector wise user follower count differs significantly.

Sector Wise Significance of Retweet Count

To test whether sector wise retweet count for tweet differ significantly.

H0: There is no significant difference among the sectors in terms of retweet count.

H1: There is significant difference among the sectors in terms of retweet count.

```
#Category wise retweet count
#Objective: To test whether category wise retweet count differ significantly

Political_RT = newDf1.loc[newDf1['category_name'] == 'Political', 'retweet_count']
GI_RT = newDf1.loc[newDf1['category_name'] == 'Govt Institutions', 'retweet_count']
Gaming_RT = newDf1.loc[newDf1['category_name'] == 'Gaming', 'retweet_count']
IT_RT = newDf1.loc[newDf1['category_name'] == 'IT', 'retweet_count']
Pharma_RT = newDf1.loc[newDf1['category_name'] == 'Pharma', 'retweet_count']
Automobile_RT = newDf1.loc[newDf1['category_name'] == 'automobile', 'retweet_count']
MC_RT = newDf1.loc[newDf1['category_name'] == 'movie_celebrity', 'retweet_count']
MTS_RT = newDf1.loc[newDf1['category_name'] == 'Movie_TVShows', 'retweet_count']
Sports_RT = newDf1.loc[newDf1['category_name'] == 'Sports', 'retweet_count']

f_oneway(Political_RT, GI_RT, Gaming_RT, IT_RT, Pharma_RT, Automobile_RT, MC_RT, MTS_RT, Sports_RT)
```

Output:

```
F_onewayResult(statistic=41.44148490983613, pvalue=1.0154988491982054e-66)
```

P value is less than 0.05. Thus, H0 is rejected at 5% level of significance. It may be concluded that the sector wise retweet count for tweet differs significantly.

Chi-Square Test for Association Between Tweet Type and Sentiment Category

To test whether Trolling depends on sentiment category.

H0: There is no association between tweet type and sentiment category.

H1: There is association between tweet type and sentiment category.

```
List = [Pos, Neg, Neu]

stat, p, dof, expected = chi2_contingency(List)

print("Statistics: ", stat, "\np-Value: ", p, "\nDegree of Freedom: ", dof, "\nExpected Frequencies: \n", expected)
```

Output:

```
Statistics:  58.76128746473666
p-Value:  1.7383953525575391e-13
Degree of Freedom:  2
Expected Frequencies:
 [[47012.67669029  2451.32330971]
 [18409.11642435   959.88357565]
 [34076.20688535  1776.79311465]]
```

p-value for test of independence is less than 0.05. Thus, H0 is rejected. It may conclude that there exists significant association between sectors and tweet type (Troll/ Not Troll).

T-Test for Equality of Sentiment Polarity for Troll and Not-Troll:

Sentiment Polarity gives idea about whether the tweet is Positive negative or neutral.

To test whether trolls have negative sentiment than non-troll tweets following test is performed.

H0: The average sentiment polarity for trolls and non-trolls is equal. Vs

H1: The average sentiment polarity of trolls is more than that of non-trolls.

```
# generate two independent samples
data1 = newDf1.loc[newDf1['Troll_label'] == 0, 'Sentiment_Polarity']
data2 = newDf1.loc[newDf1['Troll_label'] == 1, 'Sentiment_Polarity']



#compare samples
stat, p = ttest_ind(data1, data2, axis = 0, equal_var = False, nan_policy = 'omit', alternative = 'greater')



print('t-Test Score = %.3f, p-Value = %.3f' % (stat, p))
```

Output

```
t-Test Score = 1.051, p-Value = 0.147
```

P value is greater than 0.05. Thus, accept H0 at 5% level of significance. It may be concluded that the average sentiment for troll and non-troll is equal.

Conclusion

- Trolling is independent of sentiment; trolling can have positive sentiment.
- There is no relationship between sentiment polarity and user profile like user follower count, user favorite count.
- It is found that sector wise trolling is significant.
- The average retweets of Troll and Non-Troll are equal. It implies that retweeting has nothing to do with tweet type.
- The average sentiment polarity for Troll and Non-Troll are equal. Though, there exist a significant association between sentiment category and tweet type.
- There is no difference among the sectors in terms of user favorite count, follower count, retweet count.

Future Scope

- We can further categorize Trolls into obscene trolls, hateful trolls, thirst trolls, etc.
- Utilization of sentiment polarity along with labeled data for better classification.
- Building an UI where the user can input tweet and check if it is troll or not and classify streaming tweets into Troll and Not-Troll in realtime.

References

1. **Natural Language Processing in Action_ Understanding, analyzing, and generating text with Python.** Publication: Manning, Authors: Hobson Lane Cole Howard Hannes Max Hapke
2. **Natural Language Processing with Python**. Publication Oreilly, Authors: Steven Bird, Ewan Klein, Edward Loper
3. **21 Recipes for Mining Twitter.** Publication: Oreilly, Author: Matthew A. Russell
4. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.** Publication Oreilly, Author: Aurelien Geron
5. www.nltk.org
6. https://ieeexplore.ieee.org/document/7266641
7. https://www.mdpi.com/1999-5903/12/2/31/pdf
8. https://arxiv.org/pdf/2012.02586.pdf
9. https://pythonprogramming.net/