INDEX

PREAMBLE

Vision and Mission

Vision of the Institute

To evolve as an industry oriented, research based Institution for creative solutions in various engineering domains, with an ultimate objective of meeting technological challenges faced by the Nation and the Society.

Mission of the Institute

1. To enhance the quality of engineering education and delivery through accessible, comprehensive and research oriented teaching-learning-assessment processes in the state-of-art environment.
2. To create opportunities for students and faculty members to acquire professional knowledge and develop managerial, entrepreneurial and social attitudes with highly ethical and moral values.
3. To satisfy the ever-changing needs of the nation with respect to evolution and absorption of sustainable and environment friendly technologies for effective creation of knowledge based society in the global era.

Vision of the Department

To continually improve upon the teaching-learning processes and research with a goal to develop quality technical manpower with sound academic and practical experience, who can respond to challenges and changes happening dynamically in Computer Science and Engineering.

Mission of the Department

1. To inspire the students to work with the latest tools and to make them industry-ready.
2. To impart research-based technical knowledge.
3. To groom the department as a learning centre to inculcate advanced technologies in Computer Science and Engineering with social and environmental awareness.

Program Outcome (PO) and Program Specific Outcome (PSO)

Program Outcomes (POs)

Engineering graduates will be able to:

1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design & Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and cultural, societal, and environmental considerations.

4. Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-Long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO1: Programming skills: Apply fundamental knowledge and programming aptitude to identify, design and solve real-life problems.

PSO2: Professional skills: Students shall understand, analyze and develop software solutions to meet the requirements of industry and society.

PSO3: Competency: Students will be competent for competitive examinations for employment, higher studies and research.

PO-PSO matrix

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
Project CS892	3	3	3	1	2	1	1	3	2	2	-	1	3	3	2

Justification:

- PO1: Fundamental knowledge of mathematics and other related fields, like statistics, are important for any machine learning project.
- PO2: Substantiated conclusions were drawn on the basis of engineering and sciences, after thorough literature review.
- PO3: Solutions to problems were designed and developed, conforming to all necessary considerations.
- PO4: We have used pre-trained models, i.e. existing research-based solutions and have not developed our own research-based synthesis.
- PO5: Modern tools like Python-based Google Colab and Anvil Works have been used for prediction purposes.
- PO6: While our model solves an important modern problem of bias in public perception using engineering, it does not affect most of the mentioned fronts
- PO7: Our project has no impact on the environment and is, hence, sustainable in nature.
- PO8: We have conformed to all moral, ethical and professional values while creating this project.
- PO9: The project is a collaborative group effort of students from same discipline, with most of the development taking place online
- PO10: Societal interactions are inculcated in pre-trained models and documentation and presentation are a part of our assessment process
- PO11: No management abilities were used. No financing required as every resource is open-source or free to the general public
- PO12: Life-long learning is promoting and we will keep improving this project and others with changing time and technology
- PSO1: Fundamental knowledge applied to solve problems relevant for modern day
- PSO2: Requirements meet professional standards, with proper design, development and analysis taking place
- PSO3: Helps in research and employment prospects owing to the exponentially growing nature of our project field

ABSTRACT

Choosing between brands is always a hassle. The decision has to be a calculated one and in this process, lots of factors come in. Public perception or opinion about the brand is one such factor. Several websites show ratings and reviews related to brands. These ratings are, in many cases, biased or skewed. On the other hand, social media has become a hotspot of raw and honest opinions from circles vast and diverse, due to its open and accessible nature. Even with some form of censorship, people are more empathetic towards social media reviews than actual review websites. With one such social media giant, Twitter, as our data source, we present a solution to this dilemma – a brand comparison system, made using sentiment analysis in Python, that uses tweets to compare two (or more) brands, give us insights related to them and make a calculated decision as to which one is better and by how much using an overall rating system.

Keywords: *sentiment analysis; emotion analysis; opinion mining; Twitter; Python; Anvil; Colab; machine learning; natural language processing; NLP; digital marketing; brand image; perception; rating.*

1. Introduction

In recent times, social media has paved a new way for a demographic – critics. From boosting the stock value of cryptocurrencies [1, 2] to tarnishing the reputation of toxic personalities [3], critics have found a niche on social media to vent out their opinions. Just as people's opinions were swayed by using advertisements in newspapers and magazines, social media has become a more accessible and popular method of reaching out for a popular consensus. Hashtag activism [4] has reached heights with potential that no other means of communication can reach.

Another field, which is not very far from the technological world of social media, but is on a similar exponential rise, is machine learning. Our project combines these two fields in the beautiful world of Natural Language Processing (NLP), a subdomain of machine learning, which has been used to determine the public perception or image of a particular brand. This 'brand' can be as simple as a company or as controversial as a political figure.

Fig.1: A word cloud example

Popular review systems like the one in Google Maps and domain-specific apps and websites already exist to display their version of the public ratings. However, these places are plagued with fake reviews and ratings [5] which are designed to feed on the gullible and unaware. While social media is known to have its share of bots and biased posts, there is also a plethora of public sentiment, which has been analysed in this project to give us how people on a particular social media, like Twitter, perceive a particular brand. Aptly named sentiment analysis, this process will show us results in the form of word clouds (fig. 1) and statistics in a tussle between two brands.

1.1. Objective of the Project

The problem posed is the dilemma of choosing between two or more brands. We need to solve our problem using an unbiased overall opinion about such brands from a reliable and unadulterated source. One such place we look into when such a dilemma arises is a social media platform.

Our objective is to choose social media platforms to extract user posts and form an overall opinion about user-specified brands based on those posts using machine learning (specifically natural language processing). The platform we have chosen is Twitter, due to the wide variety of Python modules that support Tweet extraction and the policies involved with Twitter data manipulation. The biggest limitation is the problem of

bias, but, as Twitter's own blog [6] claims, its bot removal techniques are improving on a daily basis [7]. This makes our job easier and our results more accurate.

1.2. Brief Description of Project

If we want to buy a product, e.g. a soft drink, and the store has Coke and Pepsi as the two available options, we are faced with a dilemma. While advertising and other ways [8] are continuously adopted to help build a brand image they are not a great measure for judging such products, relying highly on psychological manipulation [9]. Similarly, while comparing leaders of the world, journalists will have their own opinions, which may or may not represent the sentiment of the people as a whole. An owner of a country might be thinking of handing a project to an IT Company. In such a situation, the manager would want a wider and trustworthy opinion rather than look up answers on sites like Quora [10] or reviews on Google [11]. In such situations, our project would be useful to provide some calm to this chaos. The user needs to enter the brands that need to be compared and a solution will be provided based on all the information gathered from a number of latest tweets from Twitter. The solution will contain insights and analytics based on tweets, which will be processed and normalised to be model-ready.

1.3. Tools and Platform

Python is the only programming language required for this project till now. Both the client and server have been made using tools based on Python3 and the code only features installed libraries (see tab. 1) and their modules.

1.3.1. Platforms

The client module is run on **Anvil Works**, a Python-based full stack web application builder. Here, we built a form-like web-application (in very little time) out of Anvil's components and linked it to our server, which does not run here (we can also run the server here using Anvil's paid features, but not for free). The server module is run on **Google Colaboratory**, a free Jupyter notebook environment that runs in Google Cloud Engine and stores its notebooks on Google Drive, with a lot of pre-installed data science and machine learning libraries. We had to use such cloud-based solutions due to the pandemic situation and this being a collaborative effort.

1.3.2. Libraries

Library	Purpose
anvil	Connects server module to Anvil client module
twint/tweepy	Extracts tweets using web scraping/ Extracts tweets with the help of Twitter API
nest-asyncio	Allows nested event loops on top of asyncio

pandas	Provides fast, flexible, and expressive data structures for structured data
matplotlib	Creates graphical visualisations
numpy	Provides arrays and other mathematical computations
re	Provides regular expression matching operations
nltk	NLP library, which also provides VADER sentiment analysis model
w3lib	Provides web-related functions like removal of HTML snippets
wordcloud	Generates word clouds
textblob	Provides TextBlob sentiment analysis model
flair	Provides Flair sentiment analysis model (optional)
string	Provides complete set of punctuations (optional)

Tab.1: List of essential Python libraries

1.4. Project Organization and Timeline

After our project topic was finalised in November 2020 (fig. 2), we had to look into existing and ongoing research. Since Python was not a part of our curriculum, we had to be well-versed with the language and the vast number of libraries it had to offer. Specifically, we had to learn about its applications in NLP and Sentiment Analysis, along with the relevant libraries it had to offer.

Once comfortable with the basics, we moved on to choosing a tweet extractor and after extracting the tweets using the suitable extractor, we started processing the raw data to provide a structured format that can be fed into the sentiment analysis model.

Multiple pre-trained sentiment analysis models were chosen and the processed data fed into them. The final step was to show our findings in a user-readable format and create a brand comparison system.

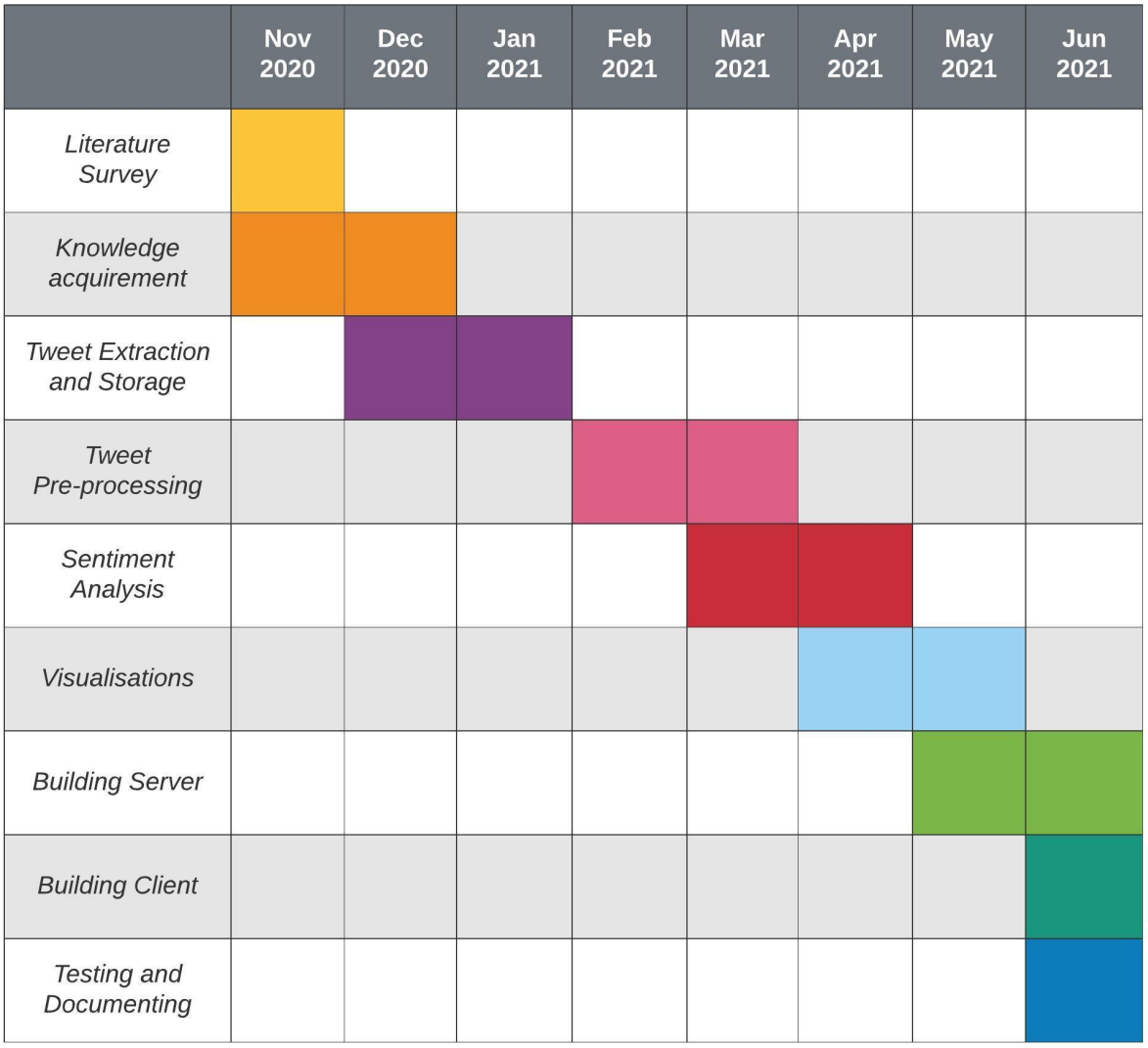

Fig.2: Gantt Chart of project planning

2. Literature Survey

There has been substantial research on social media analysis using natural language processing. Apart from papers on extracting information from images posted on social media to extract brand information [12] and text posted on social media [13], there has been research on multimodal sentiment analysis [14], which tends to use text, speech and visual processing systems to accept a wider range of information and in turn provide more accurate insights than using simply images or text. While it is the most interesting method, it is out of reach, given our knowledge base. There have been specific brands, which have been analysed, like top 500 brands in the world [15] or popular resorts [16] using sentiment analysis. There are papers which can be used by brands to track and improve their image on social media [17], but it is not from the perspective of our target – the common man.

Two research papers [18, 19] greatly helped us in our project. While the former gives us a mathematical insight to an original sentiment analyser, the latter provides us more graphical and visual tools to help us in our project.

3. Concepts and problem analysis

3.1. Python for Data Science

A survey conducted by data science giant Kaggle [20] brought out the popularity of Python as a data science language (fig. 3). Even though R and other languages were built for the purpose of building mathematical and data-driven programs, why is Python so popular?

Python, even though it is a generic programming language meant to be used in a vast variety of fields, has a plethora of libraries dealing with data science, machine learning and related fields. Coupled with the fact that it has one of the shallowest curves for a programming language, it makes Python a quick and effective tool for students, teachers and data scientists alike, most of whom are trained in other domain-specific programming languages.

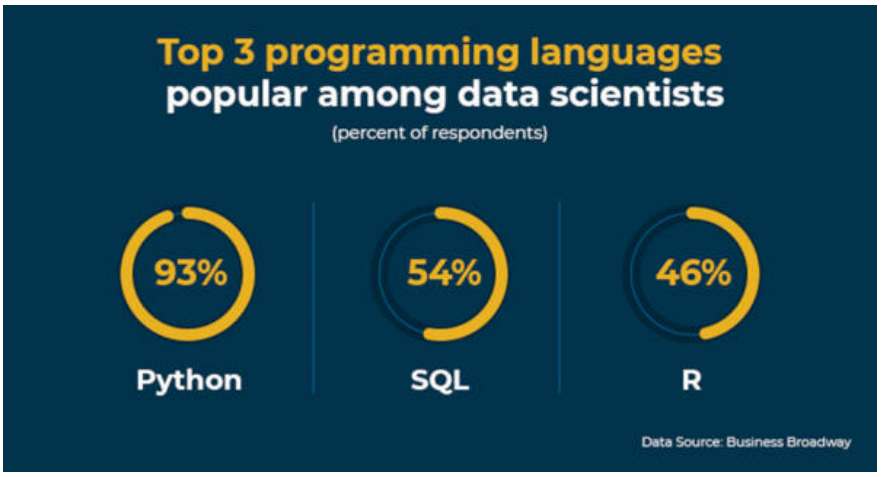

Fig.3: The data science survey result by Kaggle

Any data science problem has four important steps [21], namely data collection and preprocessing, data exploration, data modeling and finally data visualization & interpretation. Python champions each of these steps. In the matter of storing and working with data, it has powerful integration with SQL and NoSQL tools, with support for multiple file types. The second step is dealing with standardisation of data. The existence of libraries like NumPy and Pandas which can perform tasks on huge sets of data in mere seconds. This easy and efficient nature, with minimal human intervention, is what makes Python the preferred choice for the next step too, where additional libraries like Scikit-learn or SciPy step in. The final step of displaying data is done using libraries like matplotlib and Plotly, the former having strong roots in R. With the emergence of big data, cloud computing and powerful GPUs, programming has received a significant boost, with platforms like Amazon, IBM, Microsoft, Google – all building their own data science solutions. With the introduction of Jupyter Notebooks, i.e. a notebook-like file consisting of Python code, output and other documentation, Python is second to none when it comes to the first choice of an average data science enthusiast.

The Python community is also very strong and most libraries have clear and straight-forward documentation with examples, making doubts minimal (or one StackOverflow search away).

3.2. Sentiment Analysis

Sentiment Analysis is an NLP Technique where sentiment is extracted from text. A sentiment analyser specifies the overall emotional content of the passed text, while it can be further broken down and if required categorized into positive, negative or neutral. Popular sentiment analysis modules have been given in (fig. 4) below [22].

While some models rely simply on words and their frequencies, others are more context-driven and find the association between words before finalising the overall emotional content. The problem with some of the models is that they need to be trained, which would take up a considerable amount of time and effort. Our project tends to show a generalised result since other papers have already dealt with singular brands. Hence, we needed a pre-trained NLP model which is strong enough to detect context and is suitable for social media text.

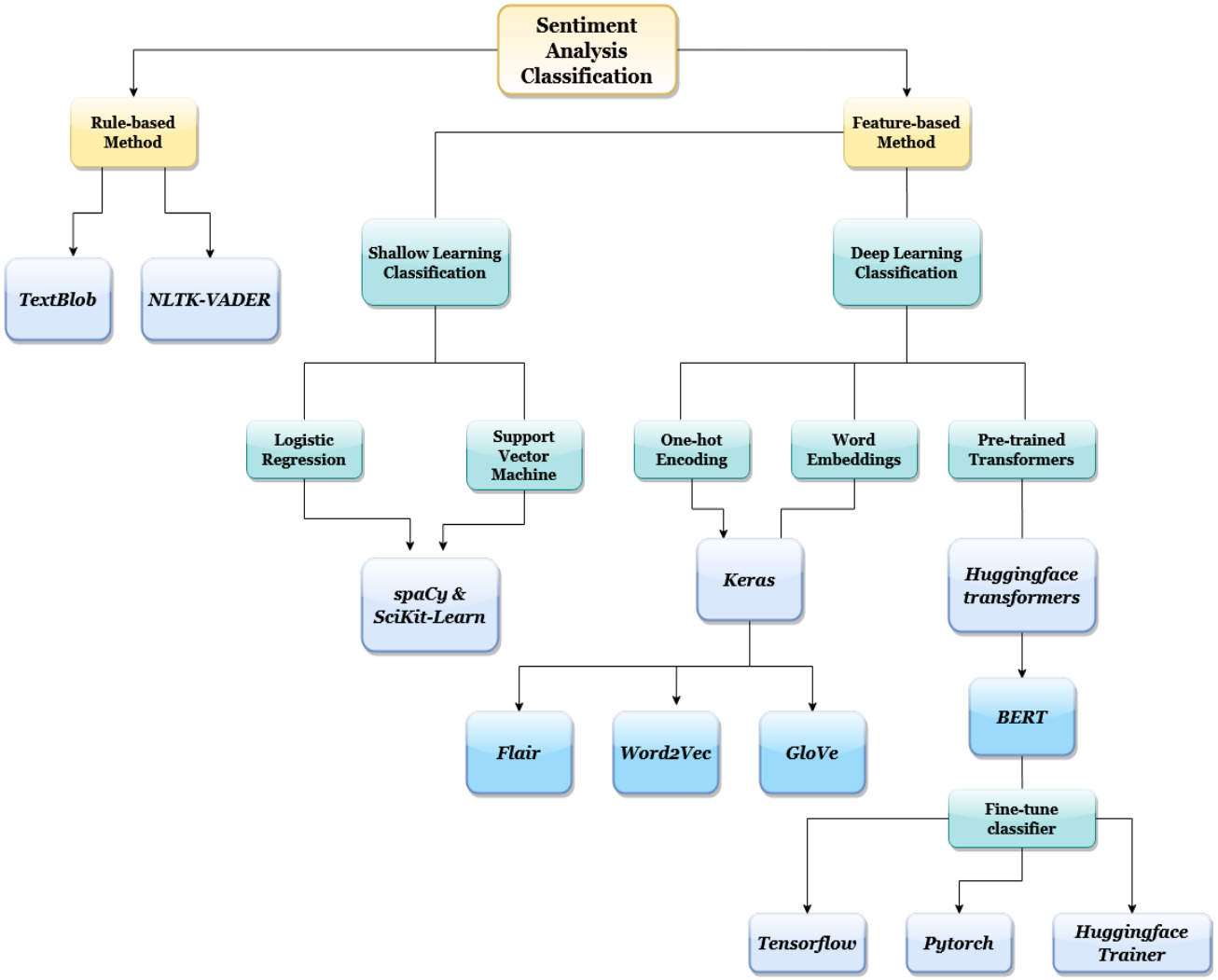

Fig.4: Popular sentiment analysis libraries in Python

We have referred to existing GitHub repositories [23] and Medium articles for looking into the feasibility of building a model from scratch, and then came to the conclusion given the speed and time constraints.

3.3. System Requirements

Our server and client are both web-based applications running on Google and Anvil clouds respectively. Hence the minimum requirement for such an application would be that of running a web browser. The recommended system requirements for some of the most popular browsers are given in Tab. 2 [24, 25].

Browser	Operating System	Requirements
Mozilla Firefox	Windows	Windows 7+ Pentium 4+, supports SSE2 512MB RAM / 2GB RAM (64-bit) 200MB drive space
	Mac	macOS 10.12+ Mac with Intel x86 or Apple silicon processor 512 MB RAM 200 MB drive space
	Linux	Depends on distro Requires dependency packages
Google Chrome	Windows	Intel Pentium 4+, supports SSE3
	Android	Android Lollipop 5.0+
	Linux	64-bit Ubuntu 14.04+, Debian 8+, openSUSE 13.3+, or Fedora Linux 24+ Intel Pentium 4+, supports SSE3

Tab.2: System requirements for popular browsers

4. Design and Methodology

4.1. Brand Perception and Comparison (Server)

The concept of brand perception was originally found in a Medium article [26] which tried to 'find image and awareness of the brand among the consumers' with the example of brand Nike. That, coupled with the visualisations shown in another article [27] gave us the idea of pitting multiple brands against each other, with the core idea of finding out their public image or perception.

Twitter has been chosen as our social media platform from which we will be extracting our data, which is a tedious process given the fact that Twitter receives numerous tweets every second from every corner of the world, making it a gigantic storehouse of data, from which only a tiny amount will be useful to us (recent tweets pertaining to a particular brand). Third-party packages like Tweepy and Twint come to our rescue. Tweepy is an open source Python package that gives a very convenient way to access the Twitter API with Python. Tweepy includes a set of classes and methods that represent Twitter's models and API endpoints. We created a Twitter Developer Account in the name of one of our members (Spandan Banerjee) to understand the OAuth process of Twitter API. The biggest limitation of Tweepy is that of the Twitter API. Twitter's API has various limits depending on our account tier and on top of that, Twitter's API limits us to a certain number of tweets in a timeline. Twint , on the other hand, is a scraping tool that is designed to grab tweets and overcome limitations of the API. Though a lot of the functionalities provided by Tweepy are missing in Twint and the documentation for Twint is subpar and outdated. However our purpose was to simply to grab a large number of tweets at a time [28], thus making Twint our chosen tweet extraction tool.

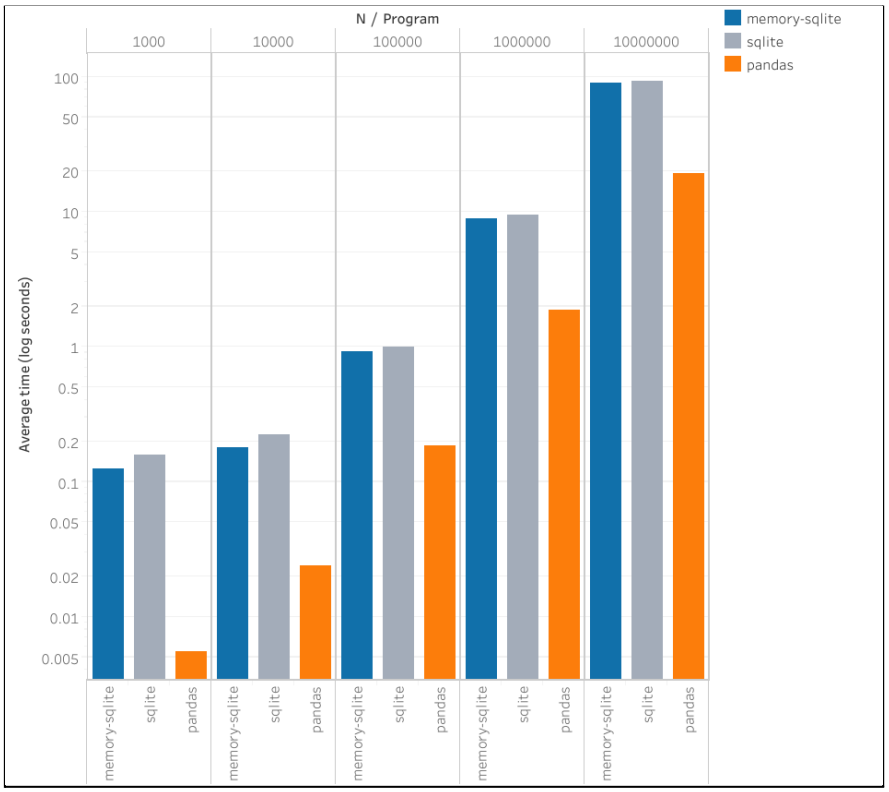

Fig.5: Comparison of average time taken for load operation by pandas vs SQLite

Now, for Tweet extraction and storage purposes, Pandas and SQL are two of the most common data manipulation tools, both of which were present in Twint. Thus, we had choices of choosing between Pandas DataFrames or SQLite. Pandas proves to be faster for grouping, loading and joining operations compared to SQLite though SQLite is faster for sorting and filtering purposes (fig. 5) [29]. Since our data does not need to be persistent for now, we use Pandas DataFrames as our data structures for storing tweet data. Now that storage is complete, we move on to processing the stored tweets to make them model-ready.

Each tweet has a lot of information as shown in (fig. 6) from which we need only the tweet field. However, there are tweets made by biased entities, e.g. fan club accounts of a particular football club, an official account filled with advertisements, etc., which will lead to a skewed result. Thus such data points are treated as outliers and are eliminated from our data to minimize bias in our result.

```
Data columns (total 38 columns):

 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              2077 non-null   object
 ...
 6   tweet           2077 non-null   object
 7   language        2077 non-null   object
 ...
 12  username        2077 non-null   object
 13  name            2077 non-null   object
 ...
 22  nlikes          2077 non-null   int64
 23  nreplies        2077 non-null   int64
 24  nretweets       2077 non-null   int64
 ...
 37  trans_dest      2077 non-null   object

dtypes: bool(1), float64(1), int64(6), object(30)
```

Fig.6: Data fields of extracted tweets (using Twint in Colab)

After the aforementioned step, we drop the username, name and other data as it will not be further needed for the NLP processes. For text normalisation purposes, we convert our processed data to lower case. Next, tweet cleansing and preprocessing is done where certain data like URLs, HTML entities (like `&gt;`), certain special characters, @ mentions, etc. are also removed as they do not contribute to the sentiment of the data. This is a generalised statement and can differ from case to case, e.g. a brand contributing `"₹0"` and that contributing `"₹10000"` offer different sentiment values and consist of special characters (₹) and numbers. However, for simplicity, we try to remove as much as we can, since most pre-trained models remove such

values anyway while processing. In (fig. 7), we see how an ideal processed tweet might look like if all the removal steps take place to their extremes. The next step would be the removal of stop words.

Fig.7: Example of an ideal pre-processed tweet about Pepsi

Stop words are common English words that contribute nothing to the final meaning of the sentence. Removing them is necessary for accurate NLP results. However, there are some words which might be classified as stop words but contain some sentiment, which will be lost on removal. Such words (or parts of words) are don't, can't, shouldn't, etc. (fig. 8). After this processing is done, the data is ready to be used by the machine learning model such that the sentiment analysis part can be conducted.

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers',
'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs',
'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being',
'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an',
'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of',
'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then',
'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will',
'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o',
're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven',
"haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't",
'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Fig.8: List of stopwords in NLTK

While choosing a sentiment analyser, we came across a lot of pre-trained models. Most of these were stated previously in (fig. 5) and classified on the basis of functioning and learning. We have chosen two of the simplest of all these models - Vader and TextBlob. However, provision has been made to incorporate more models in future (partial code for Flair has been done during the course of this project but not fully realised).

NLTK's Vader sentiment analysis tool [30] uses a bag-of-words approach (a lookup table of positive and negative words) with some simple heuristics. Due to this simple nature, literal meanings will mostly be chosen (fig. 9), which is not how people usually interact on social media. Another downside would be out-of-vocabulary (OOV) words will not be analysed. [31]

Fig.9: Vader is the simplest rule-based tool

TextBlob is built from NLTK and pattern libraries and is particularly good in social media text processing. Similar to NLTK, it uses the bag-of-words representation. It provides polarity and subjectivity (opinion, emotion, or judgement). Due to the absence of heuristics in Vader, it does not change the intensity of emotion due to the presence of words as seen in (fig. 10). Throughout the course of our project, however, we have seen TextBlob producing better results than Vader when it comes to social media content.

Fig.10: TextBlob does not take intensity into account

Flair would be the next model of choice due to it being based on deep learning framework PyTorch, taking negation and intensity into account and having excellent results with typographical errors (fig. 11), which is very common on social media. The support for multiple languages also makes it the best among the three. However, it is trained to classify text into only positive and negative tweets, making it difficult to include neutrality. Hence, we have disabled Flair in our final working model. More insight into all three can be found in *Sample Code* (Section 5).

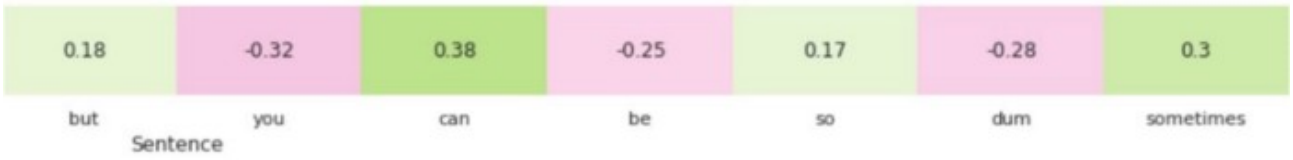

Fig.11: Flair can correct typos

We can reuse the code for a single brand's social media perception to compare two (or more) brands. For more than two brands, we can simply run a loop through all the DataFrames as given in *Conclusion and Future Work* (Section 7).

4.2. User Interface (Client)

The client module runs on Anvil Works, a full-stack web application builder based on Python. Hence, the need for creating stylesheets and maintaining repositories were eliminated. Any Python server module can be linked to the Anvil client module using `anvil-uplink` methods. We used our Colab file as the server module as given in an Anvil blog post [32]. Anvil provides a drag-and-drop solution to build the client module, whose interface is given below in (fig. 12).

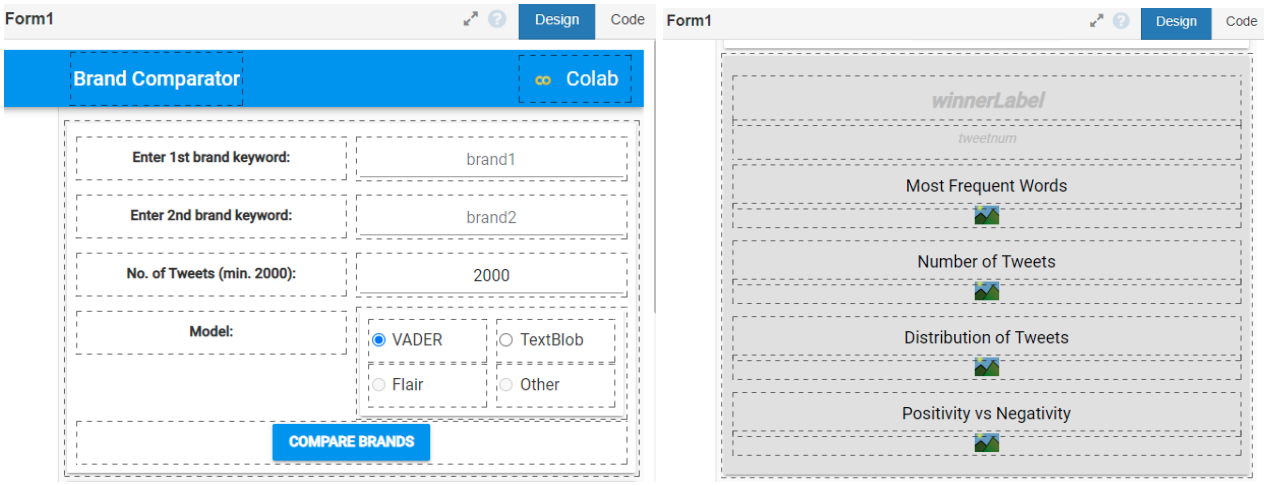

Fig.12: Design screen featuring `card_1` (left) and `card_2` (right)

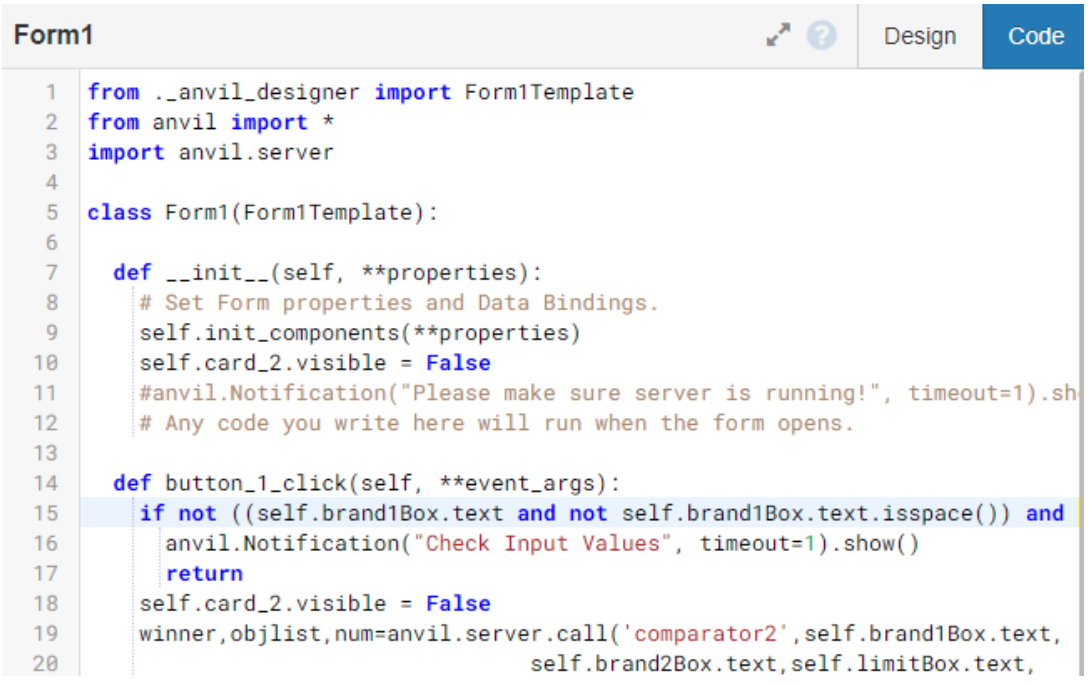

Fig. 13: Code screen for client module

In (fig. 13), we can see a glimpse of the coding screen. Important code will be discussed in *Sample Code* (Section 5). In both the Design and Code screen, we had helping tools in the form of Toolbox and Code

Snippets (fig. 14). While Toolbox provides components, their properties and other design elements, Code Snippets offer Anvil-specific code which is beginner friendly. Any challenge we faced was resolved quickly by the extensive documentation provided by Anvil Works.

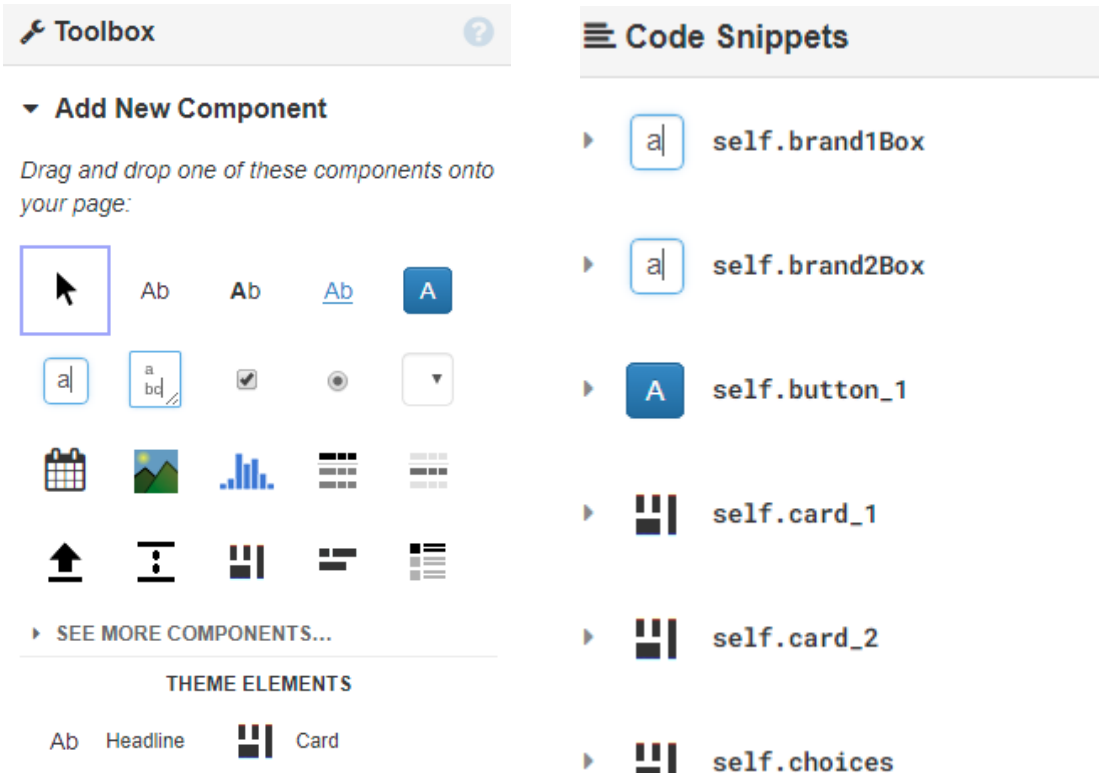

Fig. 14: Toolbox (left) and Code Snippets (right),
which help in Design and Code screens respectively

The final design is given in (fig. 15), while (fig. 16) is a proposed dark mode version of the same app (using DarkReader [33] browser extension).

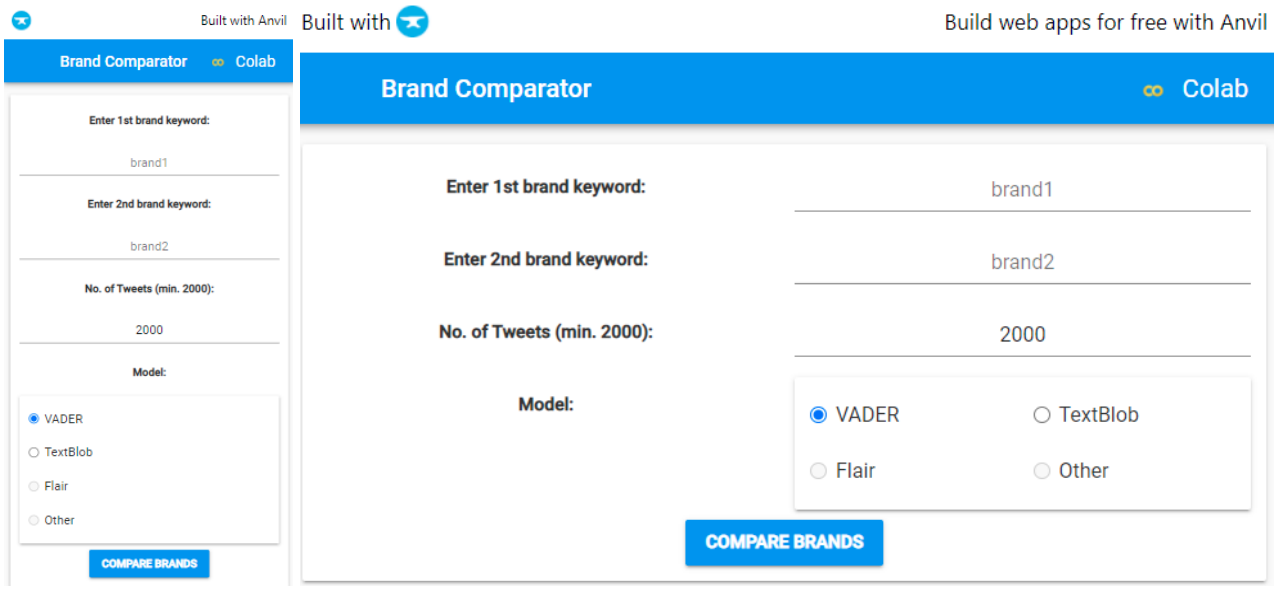

Fig.15: Client app on smartphone browser (left) and PC browser (right)

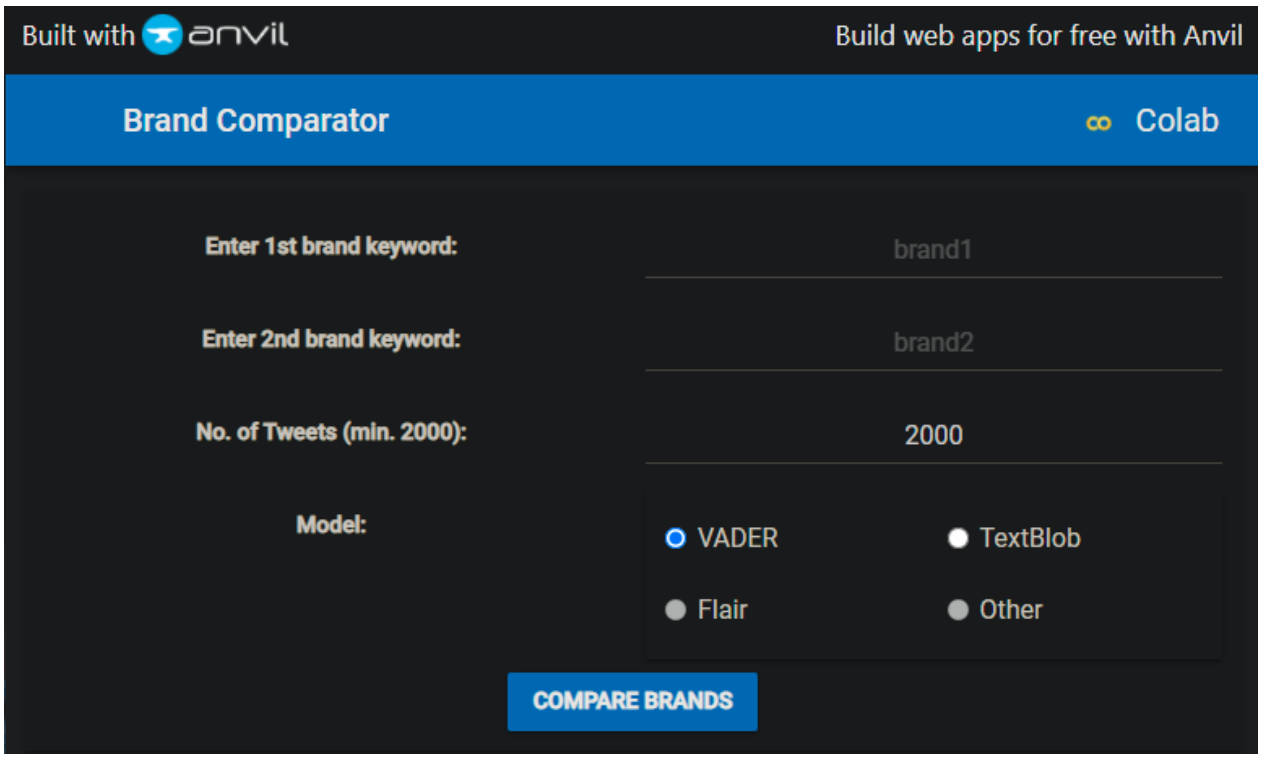

Fig.16: Dark mode of the app

5. Sample Code

5.1. Client Module

```
def button_1_click(self, **event_args):
    if not ((self.brand1Box.text and not
self.brand1Box.text.isspace()) and (self.brand2Box.text and not
self.brand2Box.text.isspace()) and (self.limitBox.text and
int(self.limitBox.text)>=2000)):
      anvil.Notification("Check Input Values", timeout=1).show()
      return
    self.card_2.visible = False

winner,objlist,num=anvil.server.call('comparator2',self.brand1Box.
text,

self.brand2Box.text,self.limitBox.text,

int(self.radio_button_1.get_group_value()))
    self.card_2.visible = True
```

```
self.image_1.source,self.image_2.source,self.image_3.source,self.i
mage_4.source = objlist
    self.winnerLabel.text = str.title(winner)+' is more positively
perceived by Twitter'
    self.tweetnum.text='(on the basis of positivity:negativity
ratio of '+str(num)+' recent tweets)'
    anvil.Notification("Execution Complete.", timeout=2).show()
    return
```

The `button_1_click()` method inside `card_1` deals with the click event of `button_1`, as given in (fig. 12). We first perform input validation, i.e. brands not entered and `limit` is a proper number above 2000. Once this is done, `card_2` (which is invisible at startup or on button click) is made visible, which contains the relevant data to be displayed. The server module sends this information from the `comparator2` method, which is a callable method (discussed later) and takes the brands, limit and chosen model as input. The output information includes the brand which is better perceived (`winner`), a list of Anvil Media objects (`objlist`) and the number of tweets considered for the comparison (`num`). These values are displayed in the appropriate container elements in the next few lines. Additionally, Anvil notifications are used to provide important messages to the user during output and wrong input.

5.2. Server Module

5.2.1. Tweet Extraction

For each brand, the tweet extraction process is as follows:

```
brand=brand.lower().strip()

c = twint.Config()
c.Search = brand
c.Lang='en'
c.Hide_output = True
c.Limit = limit
c.Filter_retweets = True
c.Pandas = True

nest_asyncio.apply()
twint.run.Search(c)
```

First we take user input (or input from client) then we deal with `c`, a Twint configuration object, whose parameters like Search, Language , Limit, etc. can be tweaked (however such filters are not very efficient as

we will see later). `nest_asyncio` is applied to the whole process to resolve nested loop errors while tweet extraction, started on `Search(c)` call.

5.2.2. Tweet Preprocessing

After each brand's tweets are obtained, the Pandas storage is cleared for the second using `twint.storage.panda.clean()`. The tweets are then processed as follows to be model-ready.

```
# Drop empty values
df = df.dropna(how = 'all')

# Drop non-English tweets
df.drop(df[df['language'] != 'en'].index, inplace = True)

# Drop duplicate tweets
df.drop_duplicates(inplace = True)

# Drop biased tweets
criterion1 = df['username'].map(lambda x: brand_name not in x)
criterion2 = df['name'].map(lambda x: brand_name not in x)
df = df[criterion1 & criterion2] # Apply both criteria
```

We drop empty, non-English and duplicate tweets using inbuilt Pandas methods. We then remove biased tweets, as discussed in *Design and Methodology* (Section 4.1), by matching `brand_name` as substring of username or name. Now, we can drop the unnecessary columns, which don't add anything to the tweet's sentiment (however verified tweets can be given more weightage and a weighted system can be adopted).

```
df.drop(columns=['id','username', 'name', 'language'], inplace =
True)
```

The `id` can be kept as an identifier, however we are using the row number (index) as one.

```
def remove_by_regex(tweets, regexp):
        tweets.loc[:, 'tweet'].replace(regexp, '', inplace=True)
        return tweets
def remove_urls(tweets):
    return remove_by_regex(tweets,
regex.compile(r"(http?\://|https?\://|www)[^\s]+[\s]?"))
def remove_usernames(tweets):
    return remove_by_regex(tweets, regex.compile(r"@\S+"))
def html_entity(tweet):
    return replace_entities(tweet)
```

```
def remove_punctuation(tweet):
    pattern = r'[^a-zA-z0-9.,!?/:;\"\'\s]'
    return regex.sub(pattern, '', tweet)
def remove_emojis(tweets):
    emoji_pattern = regex.compile("["
                                  u"\U0001F600-\U0001F64F"
# emoticons
                                  u"\U0001F300-\U0001F5FF"
# symbols & pictographs
                                  u"\U0001F680-\U0001F6FF"
# transport & map symbols
                                  u"\U0001F1E0-\U0001F1FF"
# flags (iOS)
                                  u"\U00002500-\U00002BEF"
# chinese characters
                                  u"\U00002702-\U000027B0"
                                  u"\U00002702-\U000027B0"
                                  u"\U000024C2-\U0001F251"
                                  u"\U0001f926-\U0001f937"
                                  u"\U00010000-\U0010ffff"
                                  u"\u2640-\u2642"
                                  u"\u2600-\u2B55"
                                  u"\u200d"
                                  u"\u23cf"
                                  u"\u23e9"
                                  u"\u231a"
                                  u"\ufe0f"
# dingbats
                                  u"\u3030"
                                  "]+", regex.UNICODE)
    return remove_by_regex(tweets,emoji_pattern)
```

We have defined a general method which matches regular expressions in tweets and returns the tweets after removal of such expressions. Based on that, we have other removal methods (URL, username, emojis, etc.), along with some independent ones (HTML entities like > < ≥ ≤ , etc. and punctuation).

Stop words can be removed as follows:

```
df['tweet'].apply(lambda sentence: [word
                                    for word in sentence
                                    if word not in stop])
```

5.2.3. Sentiment Analysis

We take a sample sentence (without processing) and show how all the models are written to provide its sentiment, followed by the DataFrame version.

```
sentence = 'XYZ is a bad company. But all the employees are amazing. That makes it fine, I guess?'
```

The first model, NLTK-VADER can be used to find out how the sentence's sentiment has been divided word-by-word.

```
analyzer=SentimentIntensityAnalyzer()
tokenized_sentence = nltk.word_tokenize(sentence)
```

The output of the code above is:

```
Positive: ['amazing', 'fine']
Neutral: ['XYZ', 'is', 'a', 'company', '.', 'But', 'all', 'the', 'employees', 'are', '.', 'That', 'makes', 'it', ',', 'I', 'guess', '?']
Negative: ['bad']

Scores: {'neg': 0.166, 'neu': 0.569, 'pos': 0.265, 'compound': 0.2732}
```

Hence it is clear that the data is present in a dictionary form, with the sentiment type, its values and the overall sentiment. We will make use of this information to process all the tweets as follows:

```
# data = df
data['overall']=[analyzer.polarity_scores(x)['compound'] for x in data['tweet']]

data['sentiment_type']=''
data.loc[data.overall>0,'sentiment_type']='POSITIVE'
data.loc[data.overall==0,'sentiment_type']='NEUTRAL'
data.loc[data.overall<0,'sentiment_type']='NEGATIVE'
```

For our second model, TextBlob, we use only the following:

```
TextBlob(sentence).sentiment
```

The above code returns:

```
Sentiment(polarity=0.10555555555555564, subjectivity=0.6888888888888888)
```

TextBlob therefore requires a method to extract only the polarity value of given text.

```
def get_polarity(text):
    return TextBlob(text).sentiment.polarity
```

Now we can apply this method to entire dataframe column as follows:

```
data['polarity'] = data['tweet'].apply(get_polarity)

data['sentiment_type']=''
data.loc[data.polarity>0,'sentiment_type']='POSITIVE'
data.loc[data.polarity==0,'sentiment_type']='NEUTRAL'
data.loc[data.polarity<0,'sentiment_type']='NEGATIVE'
```

The third model, Flair, is more complex than the previous ones, due to its more complex and powerful roots.

```
flair_sentiment = flair.models.TextClassifier.load('en-sentiment')

s = flair.data.Sentence(sentence)
flair_sentiment.predict(s)
total_sentiment = s.labels

total_sentiment
```

The output for the same sentence is:

```
POSITIVE (0.6922)
```

However, we have commented out all code related to Flair in regards to the neutrality issue pointed out before in *Design and Methodology* (Section 4.1). If, however, we wanted to include the code, here is how it looks like:

```
def get_sentiment(text):
    sentObj = flair.data.Sentence(text)
    flair_sentiment.predict(sentObj)
    total_sentiment = sentObj.labels
    return str(total_sentiment[0]).split()[0]
```

This method can be applied to entire DataFrame column as follows:

```
data['sentiment_type']=''
data['sentiment_type']=data['tweet'].apply(get_sentiment)
```

The tweets, along with the sentiment values and tags can be saved in multiple formats using Pandas methods. We are using one such method to save the dataframes in XLSX (Microsoft Excel) format.

```
df1.to_excel('file_name.xlsx',index=False)
```

5.2.4. Visualisations and Output

Visualisations are an important aspect of any data science or machine learning project. Apart from providing a simple solution on grounds of statistics, it is our responsibility to arm the user with tools to dig deeper. These tools come in the form of visualisations - mainly word clouds, bar charts, donut charts (a modified pie chart) and pie charts.

An example of word cloud generation code for positive tweets related to Pepsi is:

```
stopwords = set(STOPWORDS)
stopwords.update([brand_name])
pos = data.loc[data['sentiment_type'] == 'POSITIVE']
poswords=" ".join(word for word in pos.tweet)
wordcloud1 = WordCloud(stopwords=stopwords).generate(poswords)
#print(poswords)
plt.figure(figsize=(8, 8), dpi=80)
plt.imshow(wordcloud1, interpolation='bilinear')
plt.title("Most Frequent Positive Words For
"+str.title(brand_name))
plt.axis("off")
plt.savefig('pepsi_positive.png')
plt.show()
```

The output of this code is given in (fig. 17).

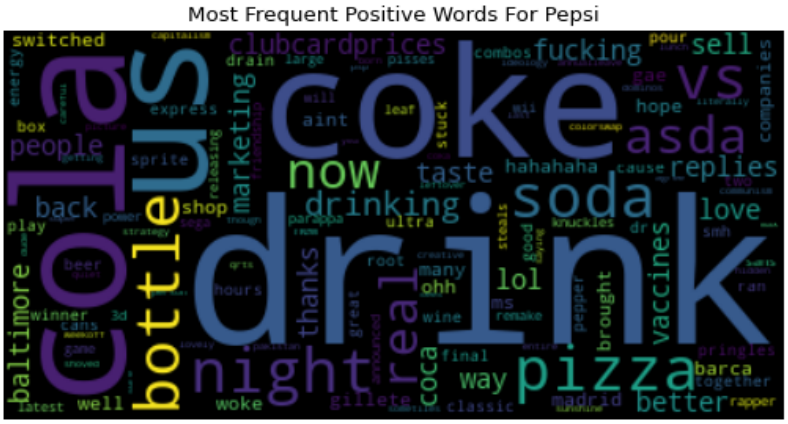

Fig.17: Word cloud (positive) for brand Pepsi

Changing the third line's `'POSITIVE'` to `'NEGATIVE'` (and other changes throughout) gives us a similar word cloud in (fig. 18).

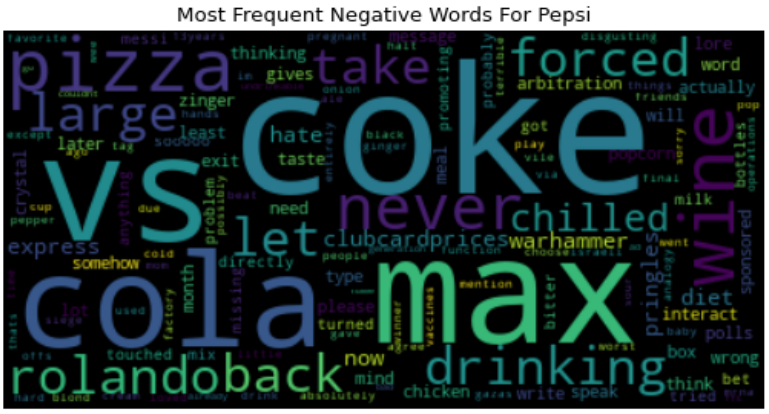

Fig.18: Word cloud (negative) for brand Pepsi

```
def wordcloud(brand1,df1,brand2,df2):
    stopwords = set(STOPWORDS)
    tags=[]
    tags.extend(brand1.split())
    tags.extend(brand2.split())

    tags.extend(subStrings("".join(brand1.split())))
    tags.extend(subStrings("".join(brand2.split())))

    stopwords.update(tags)

    plt.figure(figsize=(10, 8), dpi=150)

    plt.subplot(2, 2, 1)
    pos = df1.loc[df1['sentiment_type'] == 'POSITIVE']
    poswords=" ".join(word for word in pos.tweet if brand1 not in
word)
    wordcloud1 = WordCloud(stopwords=stopwords).generate(poswords)
    plt.imshow(wordcloud1, interpolation='bilinear')
    plt.title(str.title(brand1)+' positive')
    plt.axis("off")

    plt.subplot(2, 2, 2)
    pos = df2.loc[df2['sentiment_type'] == 'NEGATIVE']
    poswords=" ".join(word for word in pos.tweet if brand2 not in
word)
    wordcloud2 = WordCloud(stopwords=stopwords).generate(poswords)
    plt.imshow(wordcloud2, interpolation='bilinear')
```

```
    plt.title(str.title(brand2)+' negative')
    plt.axis("off")

    plt.subplot(2, 2, 3)
    pos = df1.loc[df1['sentiment_type'] == 'NEGATIVE']
    poswords=" ".join(word for word in pos.tweet if brand1 not in
word)
    wordcloud3 = WordCloud(stopwords=stopwords).generate(poswords)
    plt.imshow(wordcloud3, interpolation='bilinear')
    plt.title(str.title(brand1)+' negative')
    plt.axis("off")

    plt.subplot(2, 2, 4)
    pos = df2.loc[df2['sentiment_type'] == 'POSITIVE']
    poswords=" ".join(word for word in pos.tweet if brand2 not in
word)
    wordcloud4 = WordCloud(stopwords=stopwords).generate(poswords)
    plt.imshow(wordcloud4, interpolation='bilinear')
    plt.title(str.title(brand2)+' positive')
    plt.axis("off")
    plt.savefig('cloud.png')
    # Converting to Anvil MediaObject
    mediaobj = anvil.media.from_file('cloud.png')
    return mediaobj
```

In the first few lines of the brand comparator's word cloud code, we see an update of `stopwords`. This is different from the stopwords used in pre-processing in *Sample Code* (Section 5.2.2) and uses various alterations and typographical errors of the brand in the word clouds. The `split()` and `subStrings()` methods allow us to achieve such a result. Why this is done is explained in *Testing, Results and Discussion* (Section 6).

```
def subStrings(brand):
    temp=[]
    for i in range(len(brand)):
        for length in range(i+1,len(brand)+1):
            temp.append(brand[i: length]);
    return temp
```

The bar chart shows us the number of tweets for each brand and their sentiment.

```
def bargraph(brand1,df1,brand2,df2):
```

```
    plt.figure(figsize=(15, 8), dpi=100)
    plt.subplot(1, 2, 1)

df1.sentiment_type.value_counts().plot(kind='bar',title=str.title(
brand1)+' tweets')
    plt.subplot(1, 2, 2)

df2.sentiment_type.value_counts().plot(color='brown',kind='bar',ti
tle=str.title(brand2)+' tweets')
    plt.savefig('bar.png')
    mediaobj = anvil.media.from_file('bar.png')
    return mediaobj
```

The donut chart can be built using pie charts or bar charts. We have used the pie chart method provided in matplotlib documentation.

```
def donutchart(brand1,df1,brand2,df2):
    fig, ax = plt.subplots()
    size = 0.3

val1=df1.sentiment_type.value_counts().to_numpy(dtype='float32')

val2=df2.sentiment_type.value_counts().to_numpy(dtype='float32')
    labels=['Positive','Neutral','Negative']
    ax.pie(val1, labels=labels, radius=1,
wedgeprops=dict(width=size,
edgecolor='w'),colors=['darkgreen','darkblue','darkred'])
    ax.pie(val2, radius=1-size, wedgeprops=dict(width=size,
edgecolor='w'),colors=['green','blue','firebrick'])
    ax.set(aspect="equal", title=str.title(brand1)+' (Outer) vs
'+str.title(brand2)+' (Inner)')
    fig = plt.gcf()
    fig.set_size_inches(8,8)
    fig.dpi=150
    plt.savefig('donut.png')
    mediaobj = anvil.media.from_file('donut.png')
    return mediaobj
```

For each brand, we calculate the values to be passed for building the pie chart.

```
pos = df.sentiment_type.value_counts()['POSITIVE']
neg = df.sentiment_type.value_counts()['NEGATIVE']
```

```
neu = df.sentiment_type.value_counts()['NEUTRAL']
total = pos + neg + neu

posval = round(pos * 100 / total, 2)
negval = round(neg * 100 / total, 2)

def piechart(brand1, brand2, posval1, negval1, posval2, negval2,
pos1, neg1, pos2, neg2):
    plt.subplot(1, 2, 1)
    labels = ['Positive ['+str(posval1)+'%]' , 'Negative
['+str(negval1)+'%]']
    sizes = [pos1, neg1]
    colors = ['chartreuse','red']
    patches, texts = plt.pie(sizes,colors=colors, startangle=90)
    plt.style.use('default')
    plt.legend(labels)
    plt.title(str.title(brand1)+" Positive vs Negative")
    plt.axis('equal')
    plt.subplot(1, 2, 2)
    labels = ['Positive ['+str(posval2)+'%]' , 'Negative
['+str(negval2)+'%]']
    sizes = [pos2, neg2]
    colors = ['chartreuse','red']
    patches, texts = plt.pie(sizes,colors=colors, startangle=90)
    plt.style.use('default')
    plt.legend(labels)
    plt.title(str.title(brand2)+" Positive vs Negative")
    plt.axis('equal')
    fig = plt.gcf()
    fig.set_size_inches(10,5)
    plt.savefig('pie.png')
    mediaobj = anvil.media.from_file('pie.png')
    return mediaobj
```

The pie chart is the best indicator of our overall result, based on the positivity:negativity ratio, which is calculated for each brand as follows:

```
ratio1=round(posval/negval, 4)
```

The brand for which this ratio is higher is more positively perceived, even though this is a very simple non-weighted mathematical way of approximating such a complicated result.

6. Testing, Results and Discussion

The client-server model works as expected, with a sample output of client side given in (fig. 19-23) below using 1st brand as 'adidas' representing 'Adidas' and 2nd brand keyword as 'nike' representing Nike.

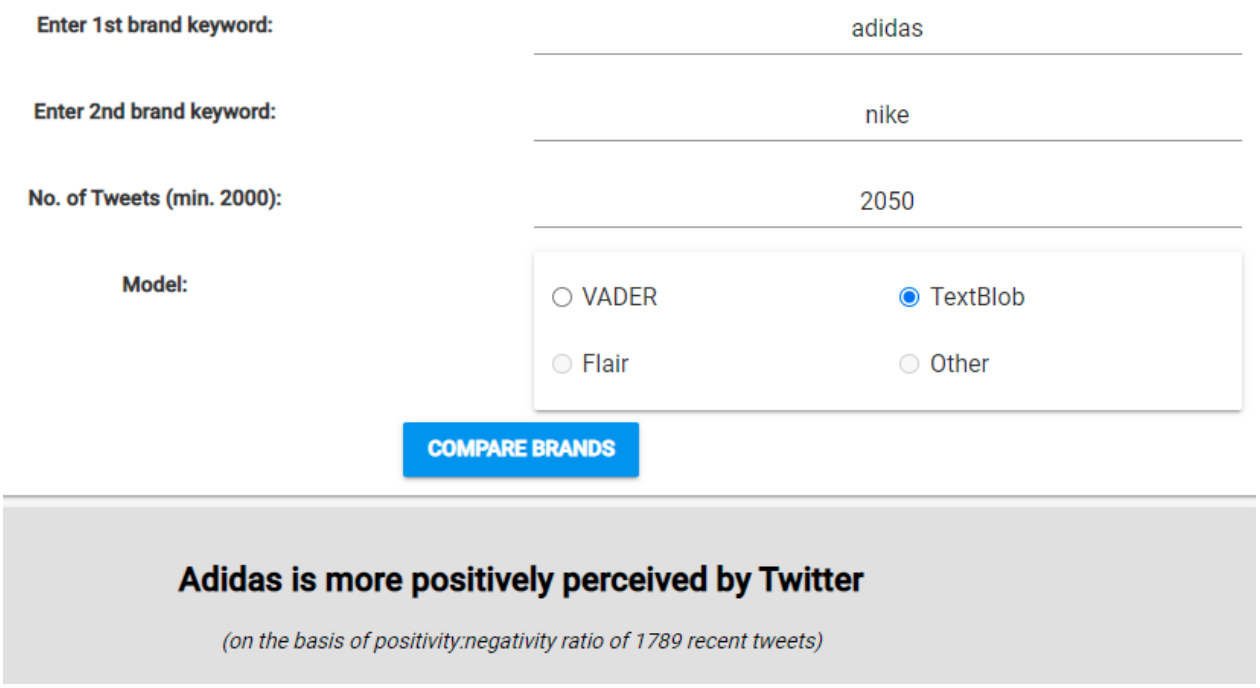

Fig.19: Comparison of Adidas vs Nike
with `limit=2050` and `TextBlob` as analyser

Fig.20: Adidas vs Nike word cloud output

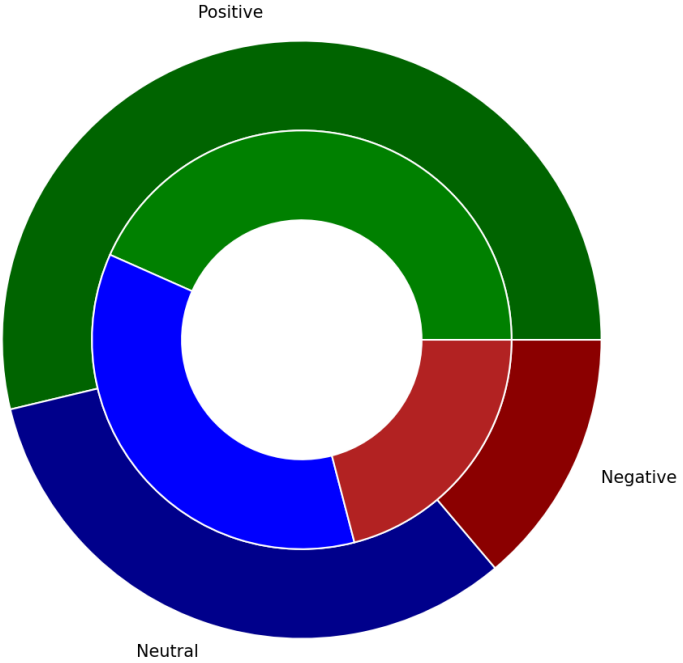

Fig.21: Donut chart for Adidas (outer ring) vs Nike (inner ring)

A donut chart, as shown above, is an important tool for checking sentiment distribution. The starting axes for positive and negative rings are aligned, making it easy to understand the difference. A bar chart, however, is essential to understand the number of tweets that have made it to the sentiment analyser. The bar chart provides a numerical value using its Y-axis which the donut chart does not.

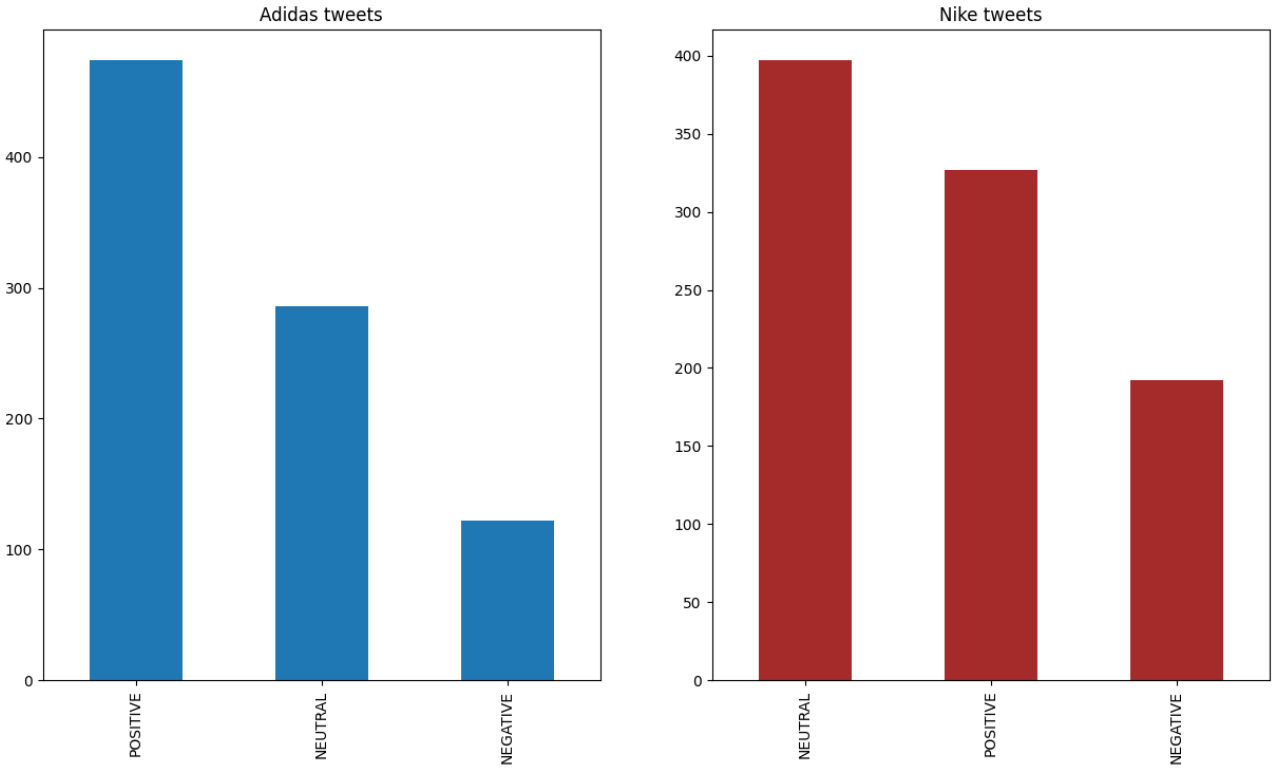

Fig.22: Bar chart for showing number of tweets for each brand

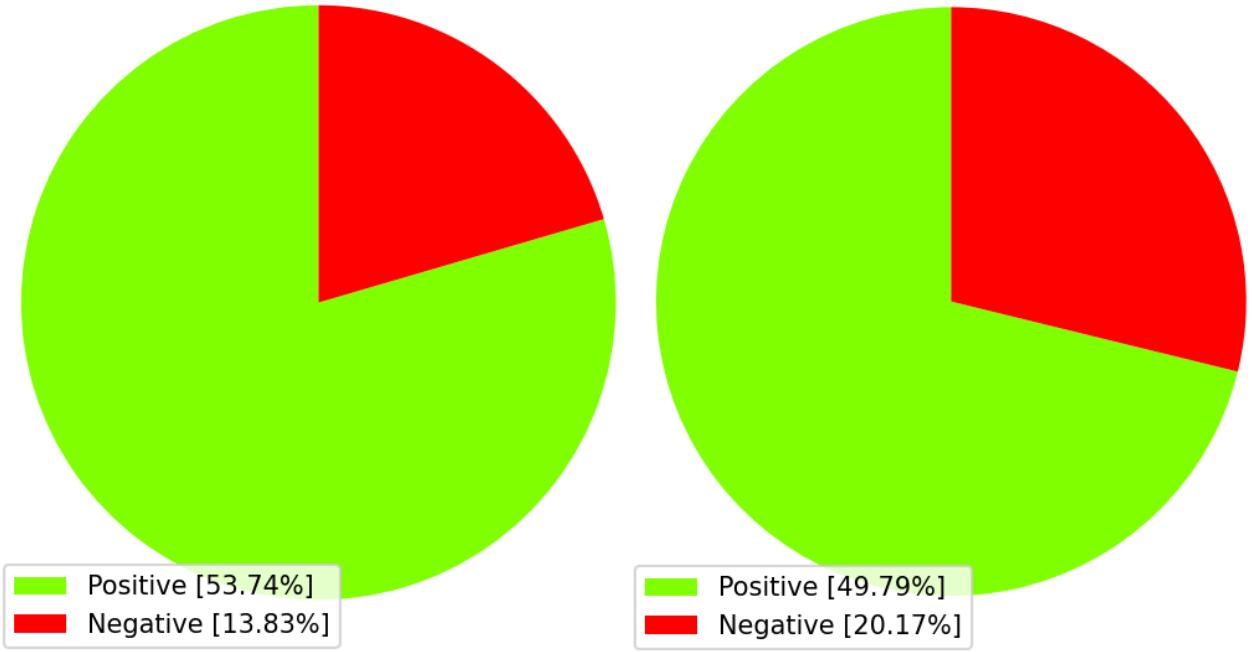

Fig.23: Pie chart showing positive to negative tweets for Adidas (left) and Nike (right)

Let us now look into the Excel files where the Adidas and Nike tweets have been stored (`tweets1.xlsx` and `tweets2.xlsx` respectively) as given in (fig. 24, 25) (screenshots taken on Microsoft Excel 2010). The tweets are mostly given the correct tags, with most misinterpreted tweets being sarcastic or using specific lingo which an NLP model cannot process easily.

the state 7on7 tournament returns today	0.6652	POSITIVE
mocinky adidas!	0	NEUTRAL
hi i loaded the adidas app but the 20 pro	-0.123	NEGATIVE

Fig.24: A section of the Adidas tweet file

need another pair of those nike hippi	0.5106	POSITIVE
nike	0	NEUTRAL
university blue nike dunk low up for g	-0.3382	NEGATIVE

Fig.25: A section of the Nike tweet file

The `nest_asyncio` library is another source of runtime errors. Mostly these errors are obtained when the same instance of client is used for another comparison or when multiple instances are running. Let us go through such errors:

```
ClientOSError: [Errno 104] Connection reset by peer
at ...
```

This error occurs when there is some socket error. This is related to internal sleep and wake functions and total handling of nested loops by `nest_asyncio`.

```
ClientPayloadError: Response payload is not completed
at ...
```

This error was obtained while running multiple instances or obtaining tweets one after the other. This is a combined result of mishandling of both Colab and Anvil instances.

The stop words being updated in the word cloud code was done when we pitted two technological giants against each other (using their full name and not keyword) - Tata Consultancy Services and Cognizant Technology Solutions. In an ideal scenario using 'TCS' and 'Cognizant' respectively would give us better results. However, for the sake of testing, we went ahead with such terms and noticed the words coming up in the word cloud in various forms (substrings, misspellings, whole name without spaces). This can be seen in (fig. 26). Hence, we have used splitting and substring methods to find and eliminate such cases.

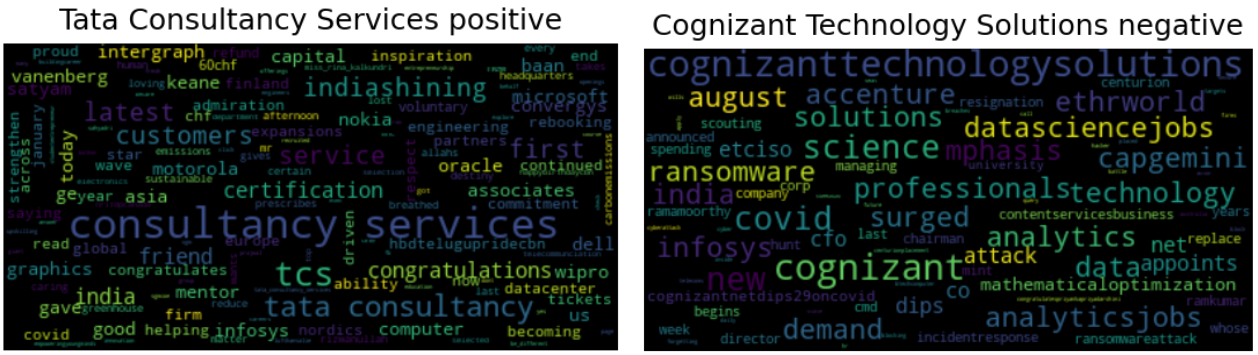

Fig.26: Word clouds with brand name appearing in them

On adding the stop word updating lines, this redundancy was significantly reduced as seen in (fig. 27) below.

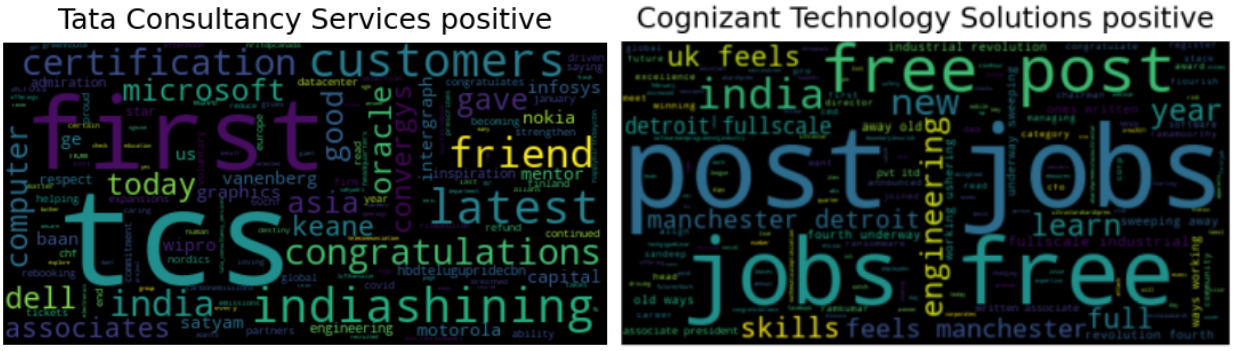

Fig.27: Word clouds with stop words code incorporated

It should be noted that using 'TCS' instead of the whole name also gives us a proper word cloud. We have also seen this while using 'Coke' instead of 'Coca Cola' or 'Coca-Cola'. This can be seen in (fig. 28).

Fig.28: Using keywords or abbreviations over whole brand names provides better results

Let us see what effects typographical errors and alterations of brand names have on the tweet extraction and overall accuracy. In all the cases that follow, `limit = 2000`. We start with an example where we pit 'TCS' against 'Cognizant Technology Solutions' (CTS). The server side summary was:

```
TCS: 886 tweets
Cognizant Technology Solutions: 1889 tweets
```

The client side output was:

```
Cognizant Technology Solutions ... 2775 recent tweets
```

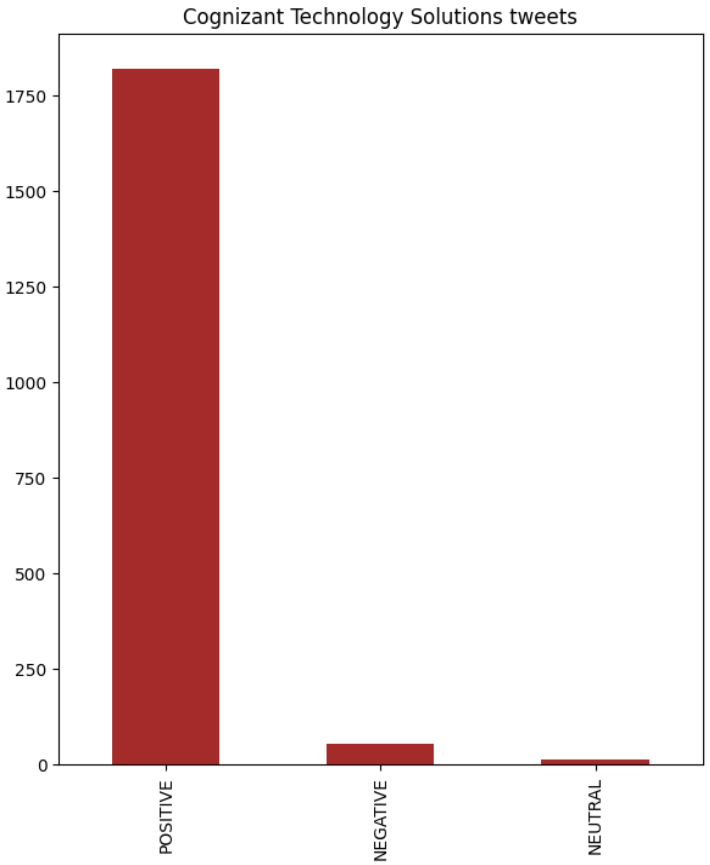

Fig.29: A skewed bar chart for CTS

This is far away from the truth, and this is where visualisations come into play. If we look at the bar graph for this case in (fig. 29), we will see a huge difference in the number of tweets (886 for TCS and 1889 for CTS) and the distribution in sentiment (low number for TCS coupled with a high number of positive tweets in CTS). Even though our total process takes the total number of tweets into account, these are certain cases where the user must use visualisations to enhance the decision-making process.

In the case of typos, we purposefully altered the second words of both brands. 'Consultancy' was misspelt as 'Consulatancy' and 'Technology' was altered as ' Technological'.

```
tata consulatancy services vs cognizant technological solutions started
[!] No more data! Scraping will stop now.
found 0 deleted tweets in this search.
Size of dataset for
1st brand: 1850
2nd brand: 35
```

The altered spelling of CTS cost more tweets than the misspelt word in TCS. However, this changes on a case-to-case basis. When both words are written properly and proper back-end processes take place, the server output is as follows:

```
tata consultancy services vs cognizant technology solutions started
Size of dataset for
1st brand: 1844
2nd brand: 1889
```

However the best results, as seen in the case of other brands too, is using the most preferred brand name (i.e. TCS is used more commonly than its full unabbreviated form). Such an output is given below:

```
tcs vs cognizant started
Size of dataset for
1st brand: 1898
2nd brand: 1902
```

Let us now move onto the performance of our two models. For this purpose, we will be concentrating on a single brand's perception, with that being Pepsi. We have already shown the word clouds for this case in (fig. 17, 18) of *Sample Code* (Section 5.2.4). We will concentrate on the number (bar chart) and distribution (pie chart) of tweets.

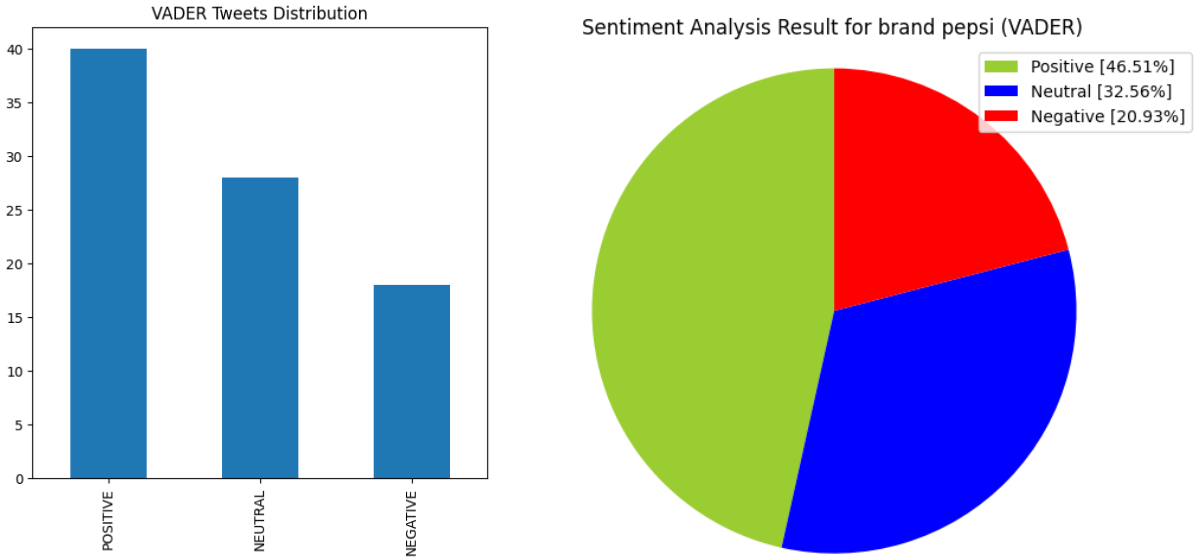

Fig.30: NLTK-VADER results for brand keyword 'pepsi'

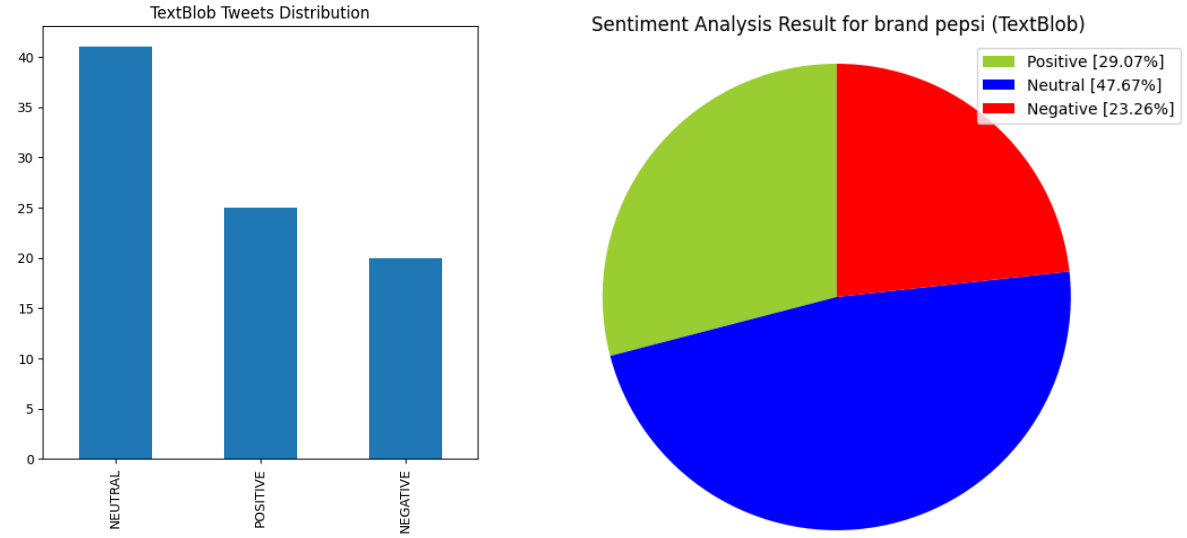

Fig.31: TextBlob results for brand keyword 'pepsi'

(Fig. 30, 31) contain the output of Vader and TextBlob respectively, with Vader having a maximum of positive tweets and TextBlob having a maximum of neutral tweets. This might be due to the coupled action of Vader's lexicon-based rules and TextBlob's contextual preferences. However, overall, there is rarely a case where the two contradict each other in terms of overall result.

Flair was left out as a model due to the facts presented in *Design and Methodology* (Section 4.1). However, we had tested it out, as can be seen in (fig. 32), and found out a lot of code rewriting was necessary to implement it before finally disabling its radio button in the client module. Debugging on Anvil was of great help due to the presence of the 'Output' panel, which can be seen on the extreme left of the same figure.

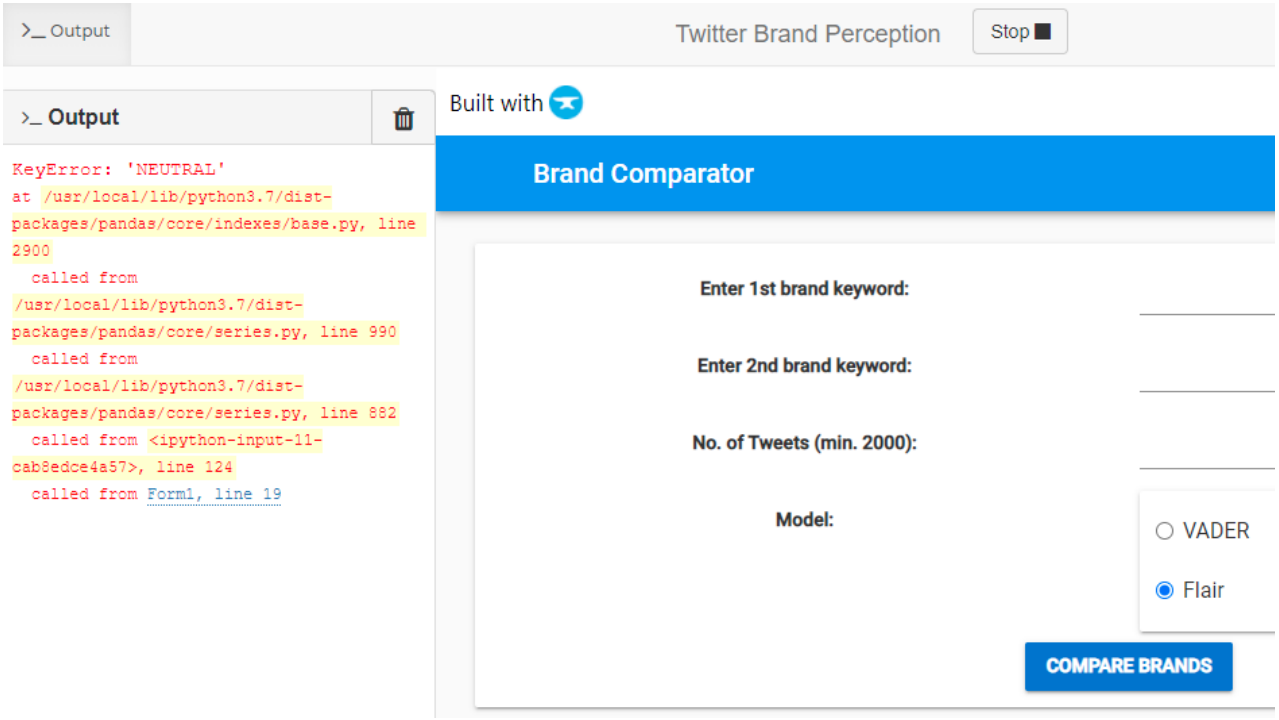

Fig.32: Sentiment analysis model Flair has no 'NEUTRAL' tag due to the nature of its training dataset and overall purpose

7. Conclusion and Future Work

While our working model provides a rough output of what we initially set out to do, it is far from being perfect and accurate. While simple rule-based models like Vader and TextBlob work for such small-scale processing, larger datasets would require more sophisticated models based on deep learning. The failure of Flair can be rectified by replacing the built in word-embedding and tagging in this line:

```
flair_sentiment = flair.models.TextClassifier.load('en-sentiment')
```

by our own word embeddings file as:

```
flair_sentiment = flair.models.TextClassifier.load('our-model')
```

Similarly, other models in (fig. 4) and more processing of data can lead to better, more accurate and near-perfect results. At the end of the day, understanding human sentiment, including sarcasm and satire, are

still a long way away for simple machine learning models like Vader and TextBlob. Regardless of what approach we choose, our goal is to provide insights and analytics based on the text, so that the user can choose a clear winner between two brands. For future work, the following can be taken into account:

1. Problems with Twint:
    a. The language filter (`c.Lang='en'`) does not work as expected. The GitHub features many issues, some solutions involve manually editing the JSON files after cloning. If this problem is solved, an extra step in tweet filtering can be eliminated.
    b. `c.Limit` of configuration object can be understood better and implemented more precisely. This gives rise to unbalanced datasets for the two brands, since no particular number of tweets are being extracted. Due to lack of proper documentation and knowledge about JSON files, it could not be fully realised.

    These problems can be removed if we build our own web scraper or use Tweepy.
2. We can include non-English tweets by using Google Translate or other APIs to convert to English and then use the tweet or pre-trained models (like Flair offers language options)
3. We can add more pre-trained sentiment analysis models and use ensemble methods to come up with the best model/average of all models/weighted average,etc. Creating our own deep learning framework, training, testing on it and using metrics to determine precision and accuracy would be the ideal scenario in such social media sentiment analysis problems.
4. While we have created a client-server model that acts as front and back ends of our project, it would be better to build an actual standalone application that does not depend on Google Colab. Web-based frameworks like Flask and deploying platforms like Heroku (or Anvil paid version) can be used for such purposes.
5. Support for multiple brand comparisons. This can be done by simply running a loop as:

```
df_list = [df1, df2, ... ]
for df in df_list:
      # operations
```

    The front-end can be made by using a single text-box (area-like) to take comma separated values. `split()` method can be used for this purpose.
6. Emojis and other non-alphanumeric data can be included and analysed either using libraries in Python or manually using unicode. The former would be feasible.
7. Including more types of visualisations and weighted statistical methods, taking into account retweets, verified tweets, bot tweets, etc.
8. Grammatical and syntactical variations to be taken into account. '`I had a bad day`' is less negative than '`I HAD A BAD DAY`' which is, in turn, less negative than '`I HAD A BAD DAY!!!`'.

BIBLIOGRAPHY

1. R. Hart, "'It's Doge Time': Dogecoin Surges As Reddit Traders Push To Make It The Crypto GameStop," Forbes, 01-Feb-2021. [Online]. Available: https://www.forbes.com/sites/roberthart/2021/01/28/its-doge-time-dogecoin-surges-as-reddit-traders-push-to-make-it-the-crypto-gamestop/.
2. Y. Peterseil, J. Ossinger, "Bitcoin surges above $38,000 after Elon Musk's Twitter bio update," ThePrint, 29-Jan-2021. [Online]. Available: https://theprint.in/economy/bitcoin-surges-above-38000-after-elon-musks-twitter-bio-update/594871.
3. I. Aravind, "#MeToo movement: Women call out their past tormentors on social media," The Economic Times, 13-Oct-2018. [Online]. Available: https://economictimes.indiatimes.com/news/politics-and-nation/metoo-movement-women-call-out-their-past-tormentors-on-social-media/articleshow/66198396.cms.
4. M. Farnsworth, "For 10 years, the Twitter hashtag has fueled both social activism and dad jokes," Vox, 23-Aug-2017. [Online]. Available: https://www.vox.com/2017/8/23/16180180/ten-year-anniversary-birthday-twitter-hashtag-started.
5. D. Proserpio, B. Hollenbeck, S. He, "How Fake Customer Reviews Do - and Don't - Work," Harvard Business Review, 24-Nov-2020. [Online]. Available: https://hbr.org/2020/11/how-fake-customer-reviews-do-and-dont-work.
6. Y. Roth, N. Pickles, "Bot or not? The facts about platform manipulation on Twitter," Twitter Blog. [Online]. Available: https://blog.twitter.com/en_us/topics/company/2020/bot-or-not.html.
7. A. Hutchinson, "Twitter Says That it's getting Better at Detecting and Removing Bots, Outlines Common Misinterpretations," Social Media Today, 19-May-2020. [Online]. Available: https://www.socialmediatoday.com/news/twitter-says-that-its-getting-better-at-detecting-and-removing-bots-outlin/578272/.
8. Ken C., "Coke vs. Pepsi: The Story Behind the Biggest Marketing Rivalry in History," ContentWriters Blog, 24-Jun-2019. [Online]. Available: https://contentwriters.com/blog/coke-vs-pepsi-the-story-behind-the-biggest-rivalry-in-history/.
9. A. V. Riabchyk, "Psychological Influencing Methods In Advertising," Efektyvna ekonomika, no. 11, 2018.
10. J. Evans, "Q: What's Wrong With Quora?," TechCrunch, 08-Dec-2012. [Online]. Available: https://techcrunch.com/2012/12/08/the-underachiever/.
11. D. Nguyen, "How Reliable Are Google Reviews? Can We Trust Reviews?," Review Doctor. [Online]. Available: https://www.reviewdoctor.co/post/how-reliable-are-google-reviews-can-we-trust-reviews.
12. M. Paolanti, C. Kaiser, R. Schallner, E. Frontoni, P. Zingaretti, "Visual and Textual Sentiment Analysis of Brand-Related Social Media Pictures Using Deep Convolutional Neural Networks,"

Image Analysis and Processing - ICIAP 2017 Lecture Notes in Computer Science, pp. 402–413, 2017.

13. S. A. A. Hridoy, M. T. Ekram, M. S. Islam, F. Ahmed, R. M. Rahman, "Localized twitter opinion mining using sentiment analysis," Decision Analytics, vol. 2, no. 1, 2015.
14. M. Soleymani, D. Garcia, B. Jou, B. Schuller, S.-F. Chang, M. Pantic, "A survey of multimodal sentiment analysis," Image and Vision Computing, vol. 65, pp. 3–14, 2017.
15. M. M. Mostafa, "More than words: Social networks' text mining for consumer brand sentiments," Expert Systems with Applications, vol. 40, no. 10, pp. 4241–4251, 2013.
16. K. Philander, Y. Zhong, "Twitter sentiment analysis: Capturing sentiment from integrated resort tweets," International Journal of Hospitality Management, vol. 55, pp. 16–24, 2016.
17. D. Al-Hajjar, A. Z. Syed, "Applying sentiment and emotion analysis on brand tweets for digital marketing," 2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT), 2015.
18. M. Ghiassi, J. Skinner, D. Zimbra, "Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network," Expert Systems with Applications, vol. 40, no. 16, pp. 6266–6282, 2013.
19. A. Baj-Rogowska, "Sentiment Analysis of Facebook Posts: The Uber Case," SSRN Electronic Journal, 2017.
20. B. Hayes, "Programming Languages Most Used and Recommended by Data Scientists," Business Broadway, 13-Jan-2019. [Online]. Available: https://businessoverbroadway.com/2019/01/13/programming-languages-most-used-and-recommended-by-data-scientists/.
21. P. D. Bhavsar, "Why Python is One of the Most Preferred Languages for Data Science?," KDnuggets. [Online]. Available: https://www.kdnuggets.com/2020/01/python-preferred-languages-data-science.html.
22. H. Naushan, "Sentiment Analysis of Social Media with Python," Medium, 02-Oct-2020. [Online]. Available: https://towardsdatascience.com/sentiment-analysis-of-social-media-with-python-45268dc8f23f.
23. M. Zabłocki, "ml-twitter-sentiment-analysis," GitHub, 13-Dec-2016. [Online]. Available: https://github.com/marrrcin/ml-twitter-sentiment-analysis/blob/develop/twitter_sentiment_analysis.ipynb.
24. "Firefox 89.0.2 System Requirements," Mozilla. [Online]. Available: https://www.mozilla.org/en-US/firefox/89.0.2/system-requirements/.
25. "Chrome Browser system requirements," Google Chrome Enterprise Help. [Online]. Available: https://support.google.com/chrome/a/answer/7100626?hl=en.
26. M. Mathur, "Brand Perception Using Twitter Sentiment Analysis," Medium, 07-Aug-2019. [Online]. Available:

https://towardsdatascience.com/with-the-emergence-of-social-media-high-quality-of-structured-and-unstructured-information-shared-b16103f8bb2e.

27. Y. Yener, "Step by Step: Twitter Sentiment Analysis in Python," Medium, 07-Nov-2020. [Online]. Available: https://towardsdatascience.com/step-by-step-twitter-sentiment-analysis-in-python-d6f650ade58d.
28. J.P Hwang, "What Python package is best for getting data from Twitter? Comparing Tweepy and Twint", Medium, 29-June-2020.[Online]. Available : https://towardsdatascience.com/what-python-package-is-best-for-getting-data-from-twitter-comparing-tweepy-and-twint-f481005eccc9
29. P. Paczuski, "SQLite vs Pandas: Performance Benchmarks", The Data Incubator, 23-May-2018 [Online]. Available : https://blog.thedataincubator.com/2018/05/sqlite-vs-pandas-performance-benchmarks.
30. C.J. Hutto, E. Gilbert, "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text", Eighth International AAAI Conference on Weblogs and Social Media, 2014.
31. M. Terry-Jack, "NLP: Pre-trained Sentiment Analysis," Medium, 02-May-2019. [Online]. Available: https://medium.com/@b.terryjack/nlp-pre-trained-sentiment-analysis-1eb52a9d742c.
32. R. Britnell, "Turning a Google Colab notebook into a web app," Anvil. [Online]. Available: https://anvil.works/learn/tutorials/google-colab-to-web-app.
33. Darkreader, "darkreader," GitHub. [Online]. Available: https://github.com/darkreader/darkreader.