KING SAUD UNIVERSITY

College of Computer & Information Sciences

Software Engineering Department

COLLEGE OF COMPUTER AND INFORMATION SCIENCES

DEPARTMENT OF SOFTWARE ENGINEERING

SWE 486 – Cloud Computing and Big Data

Section: 66346

Group: 6

Prepared By:

#	Name	ID
1	Jouri Mufadhi Alanazi	441200780
2	Monerah Faris Alsubaie	441200880
3	Renad Nigr Alsubaie	441200868
4	Mais Ashraf Alkhatib	441203863
5	Abeer Sami Alshaya	441201201

Supervised By: Dr. Afshan Jafri

Table of Contents

1. Project Description

1.1 Introduction

Threads is an online social media and social networking service operated by Meta Platforms. The app offers users the ability to post and share text, images, and videos, as well as interact with other user's posts through replies, reposts, and likes. Closely linked to Meta platform Instagram and additionally requiring users to both have an Instagram account and use Threads under the same Instagram handle, the functionality of Threads is similar to Twitter. The application is available on iOS and Android devices, while a web version offers limited functionality. It is the fastest-growing consumer software application in history, gaining over 100 million users in its first five days, surpassing the record previously set by ChatGPT.

In October 2019, Threads was introduced as a separate app available for Android and iOS. This app's functionality resembled that of Snapchat, allowing users to communicate through messaging and video chats. It was integrated with Instagram's "Close friends" feature, so that users could send images, photos, and texts privately to others, and was embedded with Instagram's photo editing system. Instagram discontinued this version of Threads in December 2021, mainly due to most of its features being rolled out on Instagram, as well as low usage compared to other social media applications. Approximately 220,000 users globally downloaded the original Threads app, less than 0.1% of Instagram's monthly active users.

Overall, threads aim to provide users with a similar experience to Twitter. Threads prioritizes public dialogues over private communications, commonly known as microblogging, and is closely linked to Instagram, its sister social networking service. Accompanied by the launch of Threads, Meta announced their vision for the app to be a "positive and creative space to express your ideas". Users can create posts consisting of up to 500 characters of text or 5

minutes of video content (compared to Twitter's 280 characters of text and 2 minutes 20 seconds of video content for non-paying users). However, Threads lacks other common social media features, such as hashtags, trending stories, and direct messaging.
In response to user feedback, Threads will introduce a new home feed for posts along with several updates to the social media app. These changes will include the capability to edit posts, translation into multiple languages, and an improved user interface for switching between different Threads accounts.

Threads has also introduced a reposting feature which is visible on each user's profile tab and in their following feed, Threads has the same community guidelines as Instagram.*[1]*

In this project, we will obtain our reviews about Threads Application on Google Play Store & App Store using Python and its libraries, and we will conduct a Sentiment analysis on these users' reviews and experiences regarding the Threads application.

1.2 Goal

Our primary goal is to evaluate the level of satisfaction among users of Threads Application, extract valuable information about their preferences and opinions, and identify any challenges they may be encountering. The outcome of this analysis can greatly improve the overall experience of Threads Application Users.

1.3 Initial Hypothesis

The initial hypothesis for analyzing the Threads, an Instagram App Reviews dataset could be as follows:

- H1: By analyzing user feedback, ratings, and sentiment analysis, we aim to compare user satisfaction levels between Twitter and the Threads app.
- H2: Through data analysis of customer feedback, reviews, and ratings, we seek to measure and evaluate user satisfaction with the Threads app.
- H3: Conducting a comprehensive analysis, we aim to compare user engagement, satisfaction, feature preferences, content relevance, user experience, and performance between Twitter and the Threads app.

By utilizing the dataset from the Threads app reviews, researchers from the Department of Software Engineering at King Saud University, College of Computer and Information Sciences aim to analyze and validate these hypotheses. The dataset provides a comprehensive collection of user reviews from both the Google Play Store and App Store, allowing for a thorough understanding of user satisfaction, evaluation of app performance, and identification of emerging patterns.

1.4 Objective

Throughout this project we aim to understand how to apply data analysis techniques to achieve several objectives, such as:

Measure User Satisfaction: Through data analysis, compare user satisfaction levels between Twitter and the Threads app. By analyzing user feedback, ratings, and sentiment analysis.

Measure User Satisfaction: Utilize data analysis techniques to measure and evaluate user satisfaction with the Threads app. By analyzing customer feedback, reviews, and ratings

Overall, the objective is to compare user engagement, satisfaction, feature preferences, content relevance, user experience, and performance between Twitter and the Threads app. By conducting a comprehensive analysis.

1.5 Analysis Plan

Threads App

We will undertake four key phases in our project to analyze the Threads app:

Phase 1: Discovery

In this phase, we have completed the Discovery stage by selecting the Threads app as our subject for analysis. We have clearly defined our project goal and objectives. To guide our analysis, we have formulated initial hypotheses based on our understanding of the app industry and the Threads app's positioning within it.

Additionally, we have outlined the objective of our analytical solution, which is to gain insights into, comparing twitter and Threads app. These insights will help to detect which app is more popular and favorite in users.

Furthermore, we have set up our development environment by downloading the necessary frameworks and ensuring all required packages are available. This ensures that we have the required tools and resources to conduct our analysis effectively.

The completion of the Discovery phase sets the foundation for the subsequent phases of our analysis plan on the Threads app.

Phase 2: Data Preprocessing

In this phase, we will thoroughly examine the data and explore it, assess its quality, and address any issues. We will utilize frameworks/libraries like Pandas and NumPy. Our steps include evaluating missing values, inconsistencies, and outliers. We will tidy the data by restructuring and standardizing it. The cleaning process involves imputing missing values, correcting

inconsistencies, and removing outliers. We will document each step with screenshots in our report.

Phase 3: Model Planning and Building

In this phase, we create datasets for training, testing, and production. We then build and execute models based on the previously planned approach. Using the specified libraries and tools, we evaluate if the models align with the dataset. This step is crucial for assessing their effectiveness. The insights gained inform further refinements and improvements to optimize model performance.

Phase 4: Communicate Results

In this phase, we will evaluate the outcomes of the model to determine the criteria for success or failure. We will carefully analyze the results and identify the key findings of our analysis. To effectively summarize our work and convey the results, we will create a narrative that captures the essence of our findings.
Additionally, we will create visualizations that highlight the discoveries related to the initial hypotheses. These visualizations will provide a clear and concise representation of the key insights derived from our analysis.

2. Development Environment

2.1 Language And IDE

- **Python**

Python is a highly versatile and efficient high-level programming language that has gained immense popularity. It is widely known for its ability to handle a wide range of tasks and its extensive library ecosystem. These libraries provide numerous functionalities and tools that greatly assist in completing projects efficiently and effectively. Python's power and speed make it an excellent choice for tackling diverse programming challenges. Its popularity continues to grow due to its flexibility and the robust support it offers through its rich library ecosystem.

- **Google colab**

Colab, a browser-based platform, enables writing and running Python code effortlessly. It is particularly advantageous for tasks involving machine learning, data analysis, and education. Serving as a hosted Jupyter notebook service, Colab eliminates the need for complex setup procedures. This convenience saves valuable time and enhances the overall user experience. With Colab, users can seamlessly engage in coding activities without worrying about infrastructure configuration.

2.2 List of used Frameworks/Libraries:

Description	Library Name
NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices, and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.*[2]*	**NumPy**
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.*[3]*	**Matplotlib**
Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.*[4]*	**Pandas**
NLTK is a toolkit build for working with NLP in Python. It provides us various text-processing libraries with a lot of test datasets. A variety of tasks can be performed using NLTK such as tokenizing, parse tree visualization.*[5]*	**NLTK**

potly is an open-source module of Python that is used for data visualization and supports various graphs like line charts, scatter plots, bar charts, histograms, area plots, etc. Plotly produces interactive graphs, can be embedded on websites, and provides a wide variety of complex plotting options.*[6]*	**plotly**
Google Play Scraper is a tool or library that allows developers and researchers to extract data from the Google Play Store. It automates the process of collecting information about mobile applications available on the Google Play platform. With Google Play Scraper, users can retrieve various details about apps, including app names, descriptions, ratings, reviews, download counts, categories, and other relevant metadata.*[7]*	**Google-Play-Scraper**
An app store scraper is a tool or library that extracts data from mobile app stores, such as the Apple App Store or Google Play Store. It automates the process of gathering information about mobile applications, including details like app names, descriptions, ratings, reviews, download counts, and other relevant metadata.*[8]*	**App-Store-Scraper**

Table 1 List of used Frameworks/Libraries

3 Data Collection

In this section we will describe how we collected our data from both google play and app store and what are the resources that we have used. Unfortunately, getting an API to collect data from these two platforms requires subscription fees with many restrictions that limited us from getting a good amount of data (200 only) which will affect our results.

Luckily, we found a dataset that matches with our idea from Kaggle, which is a popular online

platform for data science and machine learning that contains many valuable resources for data scientists. In this project we will use this dataset as our resource of information since it contains more than 32 thousand of data [click here to go to this dataset].*[9]*

Although our dataset is from an online website, we will show how to collect and scrape data from google play and app store in python (if we had that subscription).*[10]*

Step1: Importing the necessary libraries that we will use throughout our data analysis journey. To upload our data in csv format we used google.colab files library. note that in the last two lines (which are highlighted with the green #) are the libraries which requires subscription.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import nltk
import plotly
from google.colab import files
from google_play_scraper import app, Sort, reviews_all #*
from app_store_scraper import AppStore #*
```

Figure 1 Import libraries.

Step2: retrieving the data from google play and app store.

```
g_reviews = reviews_all(
        "com.instagram.barcelona&hl",
        sleep_milliseconds=0, # defaults to 0
        lang='en', # defaults to 'en'
        country='us', # defaults to 'us'
        sort=Sort.NEWEST, # defaults to Sort.MOST_RELEVANT
    )

a_reviews = AppStore('sa', 'threads-an-instagram-app', '1138075747')
a_reviews.review()
```

Figure 2 retrieves data.

Step3: Importing the dataset and saving it in a lists.

```
uploaded = files.upload()
dataset = pd.read_csv('threads_reviews.csv')
appName = dataset.iloc[:,0].values
reviewList = dataset.iloc[:,1].values
rateList = dataset.iloc[:,2].values
print(appName[:10])
```

Figure 3 Import the dataset and save it.

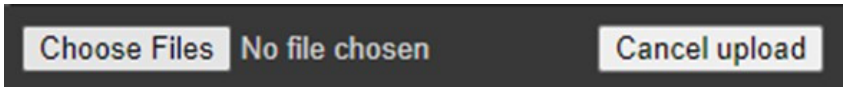


Figure 4 Import dataset file.

Data Description:

We have collected 32911 review from both google play and app store about the new application "Threads" which is powered by Meta Platforms, Inc. , our dataset contains 4 columns which are app name, review as a text, rate out of 5 and the date of the review, we note that X app (its old name was Twitter) had been mentioned in many review, so we found an importance to compare between these two platform in the next phases. Below we show a sample of the reviews:

	A	B	C	D
15915	Google Play	Worst app waste of time	1	2023-07-10 18:40:34
15916	Google Play	I'm really happy with this app	5	2023-07-12 21:54:50
15917	Google Play	This is a new app I hope it will be good	2	2023-07-09 18:48:46
15918	Google Play	Needs a delete function	1	2023-07-08 18:23:30
15919	Google Play	For some reason says account suspended soon I set it up	1	2023-07-08 5:44:40
15920	Google Play	I dlwd the app it is owesome	5	2023-07-07 14:49:57
15921	Google Play	Cheap twitter knockoff	1	2023-07-06 15:02:52
15922	Google Play	Good working app	5	2023-07-05 23:16:19
15923	Google Play	Not as good as twitter	1	2023-07-22 14:21:15
15924	Google Play	Can this App boost my followers on Instagram?	4	2023-07-06 13:18:18
15925	Google Play	It's not good as Facebook	1	2023-07-15 4:37:10
15926	Google Play	Nice data collecting. Monopolists	1	2023-07-07 8:43:09
15927	Google Play	Great better than Twitter I guess 😍	5	2023-07-07 10:44:22
15928	Google Play	Worst app I've ever used , not recommend	1	2023-07-10 10:10:06
15929	Google Play	This app didn't works	1	2023-07-19 7:30:04
15930	Google Play	Its like Twitter but made in china	1	2023-07-16 14:25:18
15931	Google Play	Love this app 😘😘 Better than Twitter 👠👠	5	2023-07-07 12:43:01
15932	Google Play	I didnt try it yet i just wanted to be the first comment	4	2023-07-06 13:04:17
15933	Google Play	Twitter is much better than threads	1	2023-07-13 6:46:58

Figure 5 Collected Reviews.

Data File:

File	Description
threads_reviews.csv	This file contains 32911 review about threads app from both google play and app store in a CSV file format.

Table 2 data file.

Phase 2

1. Data Exploration

After collecting the reviews data about threads platform from google play and app store in the previous phase, we started exploring and understanding our data, the following shows several information about it:

1- Total number of reviews in the dataset:

```
len(dataset)

32910
```

Figure 6 number of Reviews.

2- Number of rows and columns:

```
dataset.shape

(32910, 4)
```

Figure 7 number of rows and columns.

3- Names of columns in the dataset:

```
dataset.columns

Index(['source', 'review_description', 'rating', 'review_date'], dtype='object')
```

Figure 8 names of columns.

4- Types of data in each column it the dataset:

```
[40] dataset.columns
     print("the types of the above columns are:")
     dataset.dtypes

     the types of the above columns are:
     source               object
     review_description   object
     rating                int64
     review_date          object
     dtype: object
```

Figure 9 types of data in columns.

5- General information about the dataset:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32910 entries, 0 to 32909
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   source              32910 non-null  object
 1   review_description  32910 non-null  object
 2   rating              32910 non-null  int64 
 3   review_date         32910 non-null  object
dtypes: int64(1), object(3)
memory usage: 1.0+ MB
```

Figure 10 general information.

2. Data Preprocessing

Upon exploring and reviewing the collected data, we were able to identify several issues that our dataset may have. These issues include:
Issue#1. Duplication
Issue#2. Non-English reviews
Issue#3. Unrelated characters – emojis, hashtags, and mentions
Issue#4. Stop Words
Issue#5. Punctuation

1. Issue#1. Duplication

Library used: Pandas
We found that some of the reviews are duplicated. We employed the "duplicated" function to identify these duplicated reviews for the purpose of removal. The following figure illustrates some of these reviews:

```
[42] dataset[dataset.duplicated(subset="review_description")].head()
```

	source	review_description	rating	review_date
3773	Google Play	It's a good start but they have to add some th...	5	2023-07-07 12:01:14
4516	Google Play	It's a great start! I do feel like when adding...	4	2023-07-07 00:40:24
4604	Google Play	The user experience so far so good, clean and ...	4	2023-07-12 01:16:11
5938	Google Play	Not working properly	1	2023-07-07 18:52:59
5939	Google Play	Not working properly	1	2023-07-06 05:32:10

Figure 11 implements of duplicated function.

This figure shows the number of the duplicated reviews:

```
dataset.duplicated(subset="review_description").sum()
```

```
6204
```

Figure 12 number of duplicated reviews.

Then, we utilized the "drop_duplicates" function to eliminate all duplicated reviews (line 44). Subsequently, we verified the absence of any remaining duplicates by printing the count of duplicated reviews, which now are zero (line 45) and by printing the number of rows in the dataset after the removal process (line 46).

```
[44] deduplicated_dataset = dataset.drop_duplicates(subset="review_description")

[45] deduplicated_dataset.duplicated(subset="review_description").sum()

     0
```

Figure 13 drop of duplicated reviews.

```
[46] len(deduplicated_dataset)

     26706
```

Figure 14 number of rows of the updated dataset.

Lastly, we saved the updated dataset to a CSV file:

```
[47] deduplicated_dataset.to_csv("dataset_without_duplicates.csv", index= False)
```

Figure 15 save the updated dataset.

2. Issue#2. Non-English Reviews

Library used: langdetect

We discovered that many reviews were written in various languages, not just English. To address this, we imported the "detect" function from the "langdetect" library, enabling us to filter and retain only the English reviews. The following figures illustrate this process:

```
!pip install langdetect
```

Figure 16 import detect function.

```
[49] from langdetect import detect

    # Function to check if a text is in English
    def is_english(text):
        try:
            return detect(text) == 'en'  # 'en' represents English
        except:
            return False  # Handle potential errors in language detection

    # Filter the dataset to keep only English reviews
    deduplicated_dataset['is_english'] = deduplicated_dataset['review_description'].apply(is_english)
    english_dataset = deduplicated_dataset[deduplicated_dataset['is_english']]
```

Figure 17 code of removing non-English reviews.

Again, we verified our data's integrity by printing the number of rows in the dataset to ensure the successful removal of non-English reviews.

```
len(english_dataset)

20048
```

Figure 18 number of rows of the updated dataset.

3. Issue#3 (Unrelated Characters – emojis, special character, URLs)

Libraries used: Pandas, NumPy, re

Many reviews in our dataset contained unnecessary elements, including meaningless words and characters like emojis, URLs, usernames (mentions), and extra spaces. These elements can adversely impact the quality of our data analysis. In Figure 20 below, we present a code snippet that eliminates and removes these irrelevant or potentially problematic components, ensuring that the dataset maintains its structure and remains well-suited for data analysis.

```
[ ]  def remove_emojis_unrelated_chars(text):
        # Remove emojis (including Unicode emojis)
        text = re.sub('[\U0001F600-\U0001F64F\U0001F300-\U0001F5FF\U0001F680-\U0001F6FF\U0001F700-\U0001F77F\U0001F780-\U0001F7FF\U
        # Remove URLs
        text = re.sub(r'http\S+', '', text)
        text = re.sub(r'www\S+', '', text)
        # Remove usernames (mentions)
        text = re.sub(r'@\w+', '', text)
        # Remove multiple spaces
        text = re.sub(r'\s+', ' ', text).strip()
        # Replace @username with an empty string
        text = re.sub('@[^\s]+', ' ', text)
        # Remove underscores
        text = re.sub('[_]', ' ', text)
        return text

    without_emoji_dataset = english_dataset.copy()
    # Apply the function to the 'review_description' column
    without_emoji_dataset['review_description'] = without_emoji_dataset['review_description'].apply(remove_emojis_unrelated_chars)

    # Save the new dataset to a CSV file
    without_emoji_dataset.to_csv("without_emoji_dataset.csv", index=False)
```

figure 19 Unrelated characters, emojis, special character, URLs code.

4. Issue#4 (stop words)

Tool used: nltk library.

We have implemented a function to remove stop words from reviews. Stop words are common words that do not carry significant meaning in the analysis process. By eliminating these stop words, we can focus on the essential content of the reviews for more accurate analysis.

```
[ ]  import re
     import nltk
     from nltk.corpus import stopwords
     from textblob import TextBlob

     nltk.download('punkt')
     nltk.download('stopwords')

     ar_stops = set(stopwords.words('english'))
     stopwords_list = {"now", "should", "both", "each", "few", "more", "most", "other", "some", "such",
                       "no", "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s", "t",
                       "can", "will", "just", "don", 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
                       'you', 'your', 'yours', 'yourself', 'yourselves',
                       'he', 'him', 'his', 'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself',
                       'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom',
                       'this', 'that', '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'}

   def remove_stopwords(text):
       words = TextBlob(text).words
       clean_words = [word for word in words if word.lower() not in ar_stops
                      and word.lower() not in stopwords_list and len(word) >= 2]
       text=' '.join(clean_words)

       return text

   without_stopwords_dataset = without_emoji_dataset.copy()

   # Apply the remove_stopwords function to the 'review_description' column in the copied dataset
   without_stopwords_dataset['review_description'] = without_stopwords_dataset['review_description'].apply(remove_stopwords)

   # remove rows with empty 'review_description' columns in the copied dataset
   without_stopwords_dataset = without_stopwords_dataset[without_stopwords_dataset['review_description'] != '']

   # Save the new dataset to a CSV file
   without_stopwords_dataset.to_csv("without_stopwords_dataset.csv", index=False)
```

figure 20 Remove stop words code.

5. Issue#5 (Punctuation)

Libraries used: Pandas, String, re.
the re.sub() function is used to substitute any non-word characters (\W) and non-whitespace characters (\s) with an empty string. This effectively removes all punctuation from the text.
After applying the remove_punctuation function to the dataset, the without_punctuation_dataset will contain the dataset with the punctuation removed.

```
[ ]  import re
     import pandas as pd

     def remove_punctuation(text):
         # Remove punctuation and special characters
         text = re.sub(r'[^\w\s]', ' ', text)
         return text


     # Filter the dataset to keep only English reviews withou emojis
     without_punctuation_dataset = without_stopwords_dataset.copy()
     without_punctuation_dataset['review_description'] = without_punctuation_dataset['review_description'].apply(remove_punctuation)
     without_punctuation_dataset = without_punctuation_dataset[without_punctuation_dataset['review_description'].str.encode('ascii', 'ignore').str.decode('ascii') ==
     without_punctuation_dataset.to_csv("clean_reviews.csv", index=False)
```

Figure 21 Remove punctuation code.

Files table

The table below shows our development files with short description for each one:

File name	Description
Threads_reviews.csv	**A CSV file that contains all the reviews about threads platform, which are more than 32,000 reviews.**
Clean_reviews.csv	**A CSV file that contains all the reviews after cleaning them, which are approximately 20,000 reviews.**
Threads.ipynb	**A google Collaboratory notebook, which is a python code file that contains all the steps and instructions necessary for data cleaning.**

Table 3 development files

Phase 3

1. Descriptive Analysis

In this section, we will implement descriptive analysis, an approach to analyzing data with the objective of providing meaningful descriptions, displays, or summaries of data points. Its purpose is to show patterns that align with all data conditions, making it a crucial first step in the process of statistical data analysis. To do so, we followed these steps:

1- importing Libraries: First, we imported the necessary libraries for both descriptive and predictive parts:

```
[ ] from sklearn.naive_bayes import MultinomialNB
    from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
    from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
    from sklearn.preprocessing import label_binarize
    import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd
    from time import time
    import warnings
    from imblearn.over_sampling import RandomOverSampler
    from sklearn.feature_extraction.text import TfidfTransformer
    import matplotlib.patches as mpatches
    from sklearn.metrics import classification_report, fbeta_score, accuracy_score, confusion_matrix, roc_curve, auc, ConfusionMatrixDisplay, make_scorer
```

Figure 22 Importing Libraries

2- Applying sentiment analysis based on the ratings of the reviews:

```
[ ] # Map ratings to sentiment labels (example mapping, adjust as needed)
    clean_dataset['sentiment'] = clean_dataset['rating'].apply(lambda x: 'Positive' if x > 3 else 'Negative' if x < 3 else 'Neutral')
```

Figure 23 Applying sentiment analysis

3- Identify shape and column of the data frame:

```
[28] clean_dataset.shape

     (20001, 4)

[30] clean_dataset.columns

     Index(['source', 'review_description', 'rating', 'review_date'], dtype='object')
```

Figure 24 identify shape and column

4- Summary and overview of the data:

```
clean_dataset.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 20001 entries, 0 to 32909
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   source              20001 non-null  object
 1   review_description  20001 non-null  object
 2   rating              20001 non-null  int64
 3   review_date         20001 non-null  object
dtypes: int64(1), object(3)
memory usage: 781.3+ KB
```

Figure 25 overview of data

1- More advanced statistics and information, such as the mean, standard deviation, variance, minimum and maximum, and much more using **Describe** method:

```
[ ] clean_dataset.describe()
```

	rating
count	20022.000000
mean	3.034063
std	1.739500
min	1.000000
25%	1.000000
50%	3.000000
75%	5.000000
max	5.000000

```
[ ] clean_dataset.var(numeric_only=True)

rating    3.025861
dtype: float64
```

Figure 26 statistics information

```
[ ] clean_dataset.describe(include=['object'])
```

	source	review_description	review_date	sentiment
count	20022	20022	20022	20022
unique	2	18764	19557	3
top	Google Play	good	2023-07-06 11:56:55	Positive
freq	17490	48	3	9339

Figure 27 statistics information

2- Calculating Sentiment Counts:

```
[ ] clean_dataset["sentiment"].value_counts()

Positive    9339
Negative    8620
Neutral     2063
Name: sentiment, dtype: int64
```

Figure 28 calcluating sentinment

3- Calculating word occurrences:

a. **Preparing the environment and using CountVectorizer method**

```
[ ] countVector = CountVectorizer()
    word_count_vector = countVector.fit_transform(clean_dataset['review_description'].values.astype('U'))
    word_count_vector.shape

    (20022, 11992)

[ ] tfidf = TfidfTransformer(smooth_idf = True , use_idf = True)
    tfidf.fit(word_count_vector)
    TfidfTransformer()

    ▾ TfidfTransformer
    TfidfTransformer()

[ ] dfidf =pd.DataFrame(tfidf.idf_,index=countVector.get_feature_names_out() , columns = ['idf_weights'])
```

Figure 29 using countvector method

4- Getting the most frequent and less frequent 15 words:

```
dfidf.sort_values(by = ['idf_weights']).head(15)
```

	idf_weights
app	2.011888
twitter	2.531545
instagram	3.074609
like	3.127682
threads	3.142041
good	3.184175
account	3.682801
better	3.716224
see	3.772938
people	3.774538
use	3.787431
follow	3.932031
please	3.950953
great	4.038746
new	4.084621

Figure 30 most frequent word

```
[ ] dfidf.sort_values(by = ['idf_weights']).tail(15)
```

	idf_weights
independly	10.21149
independence	10.21149
indecency	10.21149
ind	10.21149
incrementally	10.21149
incremental	10.21149
increased	10.21149
increadible	10.21149
incorporation	10.21149
incorporating	10.21149
incorporates	10.21149
inconsistencies	10.21149
inconsiderate	10.21149
incompetent	10.21149
kneecap	10.21149

Figure 31 less frequent word

2. Predictive Analysis

We have selected two models for our Predictive Analysis: Naïve Bayes and Logistic Regression. Naïve Bayes is a supervised learning algorithm that utilizes Bayes theorem to solve classification problems. Logistic Regression, on the other hand, is a classification algorithm that assigns observations to distinct classes. we will provide a detailed explanation of the implementation of each model below.

2.1 Model 1 (Logistic Regression)

Logistic Regression is a supervised machine learning algorithm used primarily for classification tasks; logistic regression predicts the probability of an input belonging to a certain class. It is particularly useful when the dependent variable is binary or categorical [11].

2.2 Model 2 (Naïve Bayes)

Naive Bayes is a supervised machine learning algorithm commonly used for solving classification problems. It is based on ' Bayes' theorem, which calculates the probability of a certain event occurring given prior knowledge. Naive Bayes assumes that the features used for classification are independent of each other. Naïve Bayes is particularly effective when working with large datasets and text classification tasks such as spam detection or sentiment analysis [12].

First, we start by importing all the required python libraries (as shown in the descriptive part).

```
[ ]  from sklearn.naive_bayes import MultinomialNB
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
     from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
     from sklearn.preprocessing import label_binarize
     import matplotlib.pyplot as plt
     import numpy as np
     import pandas as pd
     from time import time
     import warnings
     from imblearn.over_sampling import RandomOverSampler
     from sklearn.feature_extraction.text import TfidfTransformer
     import matplotlib.patches as mpatches
     from sklearn.metrics import classification_report, fbeta_score, accuracy_score, confusion_matrix, roc_curve, auc, ConfusionMatrixDisplay, make_scorer
```

Figure 32 Required Importing

Feature Extraction and Pre-processing

Afterward, feature extraction and pre-processing are performed by removing data with missing values (NAN) as well as eliminating the (Neutral) class sentiment. The number of reviews in the dataset is then printed to confirm the successful removal.

```
[ ]  # remove data with NAN sentiment
     clean_dataset = clean_dataset[~clean_dataset["sentiment"].isna()]
     clean_dataset = clean_dataset[~clean_dataset["review_description"].isna()]

[ ]  # remove the "Neutral" class
     clean_dataset = clean_dataset[clean_dataset['sentiment'] != "Neutral"]
```

Figure 33 Dropping NAN and Neutral records implementation

Then, we convert the column ‘sentiment’ from categorical to numerical data type; where 0 represents “Negative” and 1 represents “Positive” and we print the number of reviews in the dataset after converting.

```
[ ]  # change values to numeric
     clean_dataset['sentiment'] = clean_dataset['sentiment'].map({'Negative' : 0, 'Positive' : 1})
```

Figure 34 changing type of data from categorial to numerical.

Next, we proceeded to split the dataset into features and targets. The features consist of sentiments and reviews, represented by X. Meanwhile, the target value, which is the classification we are predicting, is denoted as Y.

```
[ ]  X = clean_dataset['review_description']
     y = clean_dataset['sentiment']
```

Figure 35 separating the data into features and target.

To ensure effective model performance, data must be in a specific format. Therefore, we need to vectorize our textual data. Despite pre-processing our data, there is a significant class imbalance that can lead to overfitting and biased results. The underrepresentation of the negative class affects the dataset's balance, resulting in poor model performance. To demonstrate the impact of balancing the data, we will work with both balanced and unbalanced datasets, each with its own training and testing data. Although resampling helps address the imbalanced dataset issue, further adjustments will be made later. For now, we will convert and split our unbalanced dataset:

```
[ ]  # Use TfidfVectorizer for feature extraction (TFIDF to convert textual data to numeric form):
     unbalanced_tfidf = TfidfVectorizer()
     unbalanced_X = unbalanced_tfidf.fit_transform(X)
     unbalanced_X.shape

     (17959, 11247)

[ ]  unbalanced_X

     <17959x11247 sparse matrix of type '<class 'numpy.float64'>'
            with 151700 stored elements in Compressed Sparse Row format>
```

Figure 36 transform textual data.

Afterward, we prepare our dataset for training:

```
[ ]  # Training Phase
     X_train_unbalanced, X_test_unbalanced, y_train_unbalanced, y_test_unbalanced = train_test_split(unbalanced_X, y, test_size=0.25, random_state=40)

[ ]  print("Training set has {} samples.".format(X_train_unbalanced.shape[0]))
     print("Testing set has {} samples.".format(X_test_unbalanced.shape[0]))

     Training set has 13469 samples.
     Testing set has 4490 samples.
```

Figure 37 Split raw training and testing records

Similar to the unbalanced dataset, we applied the same steps to balance the other training and testing dataset. We began by converting the textual data, followed by splitting the dataset into separate training and testing sets. Then, we performed a dataset transformation.

```
balanced_tfidf = TfidfVectorizer()
balanced_tfidf.fit(X)
```

```
TfidfVectorizer()
```

```
[ ] X_train_balanced, X_test_balanced, y_train_balanced, y_test_balanced = train_test_split(X , y , test_size = 0.25, random_state = 40)
```

```
[ ] print("Training set has {} samples.".format(X_train_balanced.shape[0]))
    print("Testing set has {} samples.".format(X_test_balanced.shape[0]))

    Training set has 13469 samples.
    Testing set has 4490 samples.
```

```
[ ] X_train_balanced =balanced_tfidf.transform(X_train_balanced)
    X_test_balanced = balanced_tfidf.transform(X_test_balanced)
    balanced_X = balanced_tfidf.transform(X)
    X_train_balanced

    <13469x11247 sparse matrix of type '<class 'numpy.float64'>'
            with 113458 stored elements in Compressed Sparse Row format>
```

Figure 38 Transform textual data and split the dataset

The outputs from the code segments below illustrate the distinction between the imbalanced and balanced datasets, and shows how to balance our data using RandomOverSampler method.

```
[ ] ros = RandomOverSampler(random_state=40)
    X_train_balanced, y_train_balanced = ros.fit_resample(X_train_balanced,y_train_balanced)
```

```
[ ] print("Data before balence: {} samples.".format(y_train_unbalanced.value_counts()[0]))
    print("Data before balance: {} samples.".format(y_train_unbalanced.value_counts()[1]))

    Data before balence: 6455 samples.
    Data before balance: 7014 samples.
```

```
[ ] print("Data after balance: {} samples.".format(y_train_balanced.value_counts()[0]))
    print("Data after balance: {} samples.".format(y_train_balanced.value_counts()[1]))

    Data after balance: 7014 samples.
    Data after balance: 7014 samples.
```

Figure 39 Compare dataset count before and after balancing

After splitting our data and balancing it, we started training our two machine learning models each for unbalanced and balanced datasets which are the Naive Bayes and Logistic Regression algorithms (as mentioned before).

```
# Naive Bayes for Unbalanced Data
naive_bayes_model_unbalanced = MultinomialNB()
naive_bayes_model_unbalanced.fit(X_train_unbalanced, y_train_unbalanced)

# Logistic Regression for Unbalanced Data
logistic_regression_model_unbalanced = LogisticRegression(C=1.0, max_iter=3000)
logistic_regression_model_unbalanced.fit(X_train_unbalanced, y_train_unbalanced)

# Naive Bayes for Balanced Data
naive_bayes_model_balanced = MultinomialNB()
naive_bayes_model_balanced.fit(X_train_balanced, y_train_balanced)

# Logistic Regression for Balanced Data
logistic_regression_model_balanced = LogisticRegression(C=1.0, max_iter=3000)
logistic_regression_model_balanced.fit(X_train_balanced, y_train_balanced)
```

Figure 40 Training machine learning models implementation

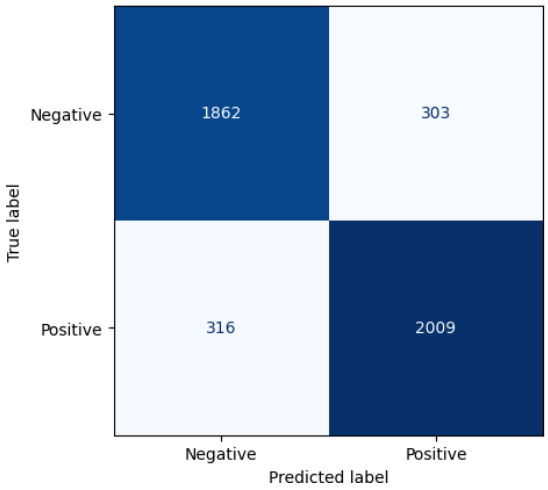
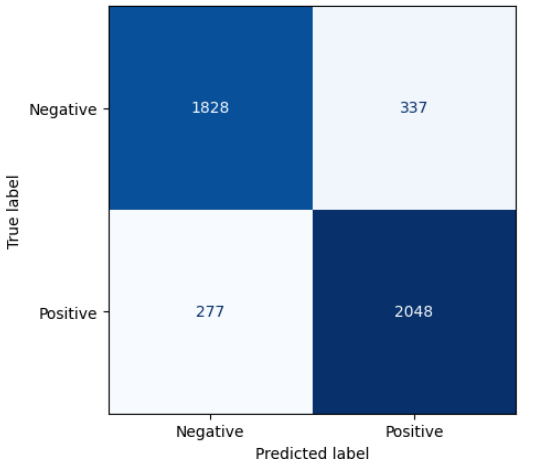
We started getting information from the data we have trained:

1- Accuracy and confusion matrix: The code below shows how we got our accuracy information and how we developed our confusion matrix for balanced and unbalanced data for both models:

```
[ ]
def plot_confusion_matrix(model, X_test, y_test, title, color):

    # Print accuracy information
    print_accuracy_info(model, X_train_unbalanced, y_train_unbalanced, X_test_unbalanced, y_test_unbalanced, title)

    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    classification_rep = classification_report(y_test, y_pred, target_names=['Negative', 'Positive'])

    display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Negative', 'Positive'])
    display.plot(cmap=color, values_format='d', colorbar=False)
    plt.title(f'{title}\nAccuracy: {accuracy:.2f}\n\nClassification Report:\n{classification_rep}')
    plt.show()

# Function to print accuracy information
def print_accuracy_info(model, X_train, y_train, X_test, y_test, title):
    # Training accuracy
    train_accuracy = model.score(X_train, y_train)

    # 10-fold cross-validation accuracy
    cross_val_scores = cross_val_score(model, X_train, y_train, cv=10)
    cross_val_accuracy = cross_val_scores.mean()

    # Testing accuracy
    test_accuracy = model.score(X_test, y_test)

    # Print information
    print(f'\n{title} Training')
    print('Accuracy Report')
    print(f'Model Accuracy: {train_accuracy:.2f}')
    print(f'10-Fold Cross Validation: {cross_val_accuracy:.2f}')
    print(f'Training Score: {train_accuracy:.2f}')
    print(f'Testing Score: {test_accuracy:.2f}')

# Plot Confusion Matrices with Details and Accuracy
plot_confusion_matrix(naive_bayes_model_unbalanced, X_test_unbalanced, y_test_unbalanced, 'Confusion Matrix - Naive Bayes (Unbalanced)', 'Blues')
plot_confusion_matrix(logistic_regression_model_unbalanced, X_test_unbalanced, y_test_unbalanced, 'Confusion Matrix - Logistic Regression (Unbalanced)', 'BuGn')
plot_confusion_matrix(naive_bayes_model_balanced, X_test_balanced, y_test_balanced, 'Confusion Matrix - Naive Bayes (Balanced)', 'Blues')
plot_confusion_matrix(logistic_regression_model_balanced, X_test_balanced, y_test_balanced, 'Confusion Matrix - Logistic Regression (Balanced)', 'BuGn')
```

Figure 41 Accuracy and confusion matrix code

a. Output of Confusion matrix and Accuracy for Naïve-Bayes:

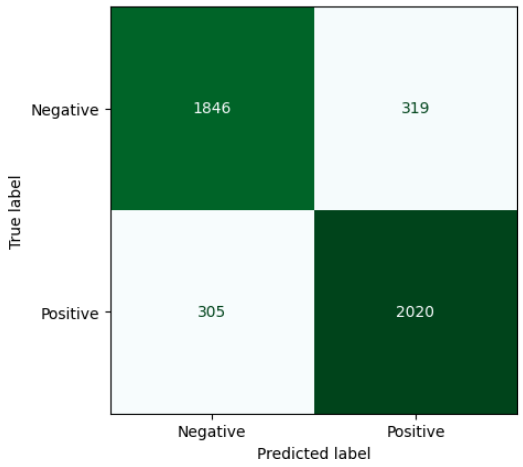
	Balanced	Unbalanced
Accuracy	Accuracy Report Model Accuracy: 0.90 10-Fold Cross Validation: 0.85 Training Score: 0.90 Testing Score: 0.86	Accuracy Report Model Accuracy: 0.90 10-Fold Cross Validation: 0.85 Training Score: 0.90 Testing Score: 0.86
Confusion Matrix	Confusion Matrix - Naive Bayes (Balanced) Accuracy: 0.86 Classification Report: precision recall f1-score support Negative 0.85 0.86 0.86 2165 Positive 0.87 0.86 0.87 2325 accuracy 0.86 4490 macro avg 0.86 0.86 0.86 4490 weighted avg 0.86 0.86 0.86 4490 	Confusion Matrix - Naive Bayes (Unbalanced) Accuracy: 0.86 Classification Report: precision recall f1-score support Negative 0.87 0.84 0.86 2165 Positive 0.86 0.88 0.87 2325 accuracy 0.86 4490 macro avg 0.86 0.86 0.86 4490 weighted avg 0.86 0.86 0.86 4490 

b. Output of Confusion matrix and Accuracy for Logistic Regression:

<table>
<tr><th></th><th>Balanced</th><th>Unbalanced</th></tr>
<tr><td>Accuracy</td><td>Accuracy Report
Model Accuracy: 0.90
10-Fold Cross Validation: 0.85
Training Score: 0.90
Testing Score: 0.86</td><td>Accuracy Report
Model Accuracy: 0.90
10-Fold Cross Validation: 0.85
Training Score: 0.90
Testing Score: 0.86</td></tr>
<tr><td>Logistic Regression</td><td>Confusion Matrix - Logistic Regression (Balanced)
Accuracy: 0.86

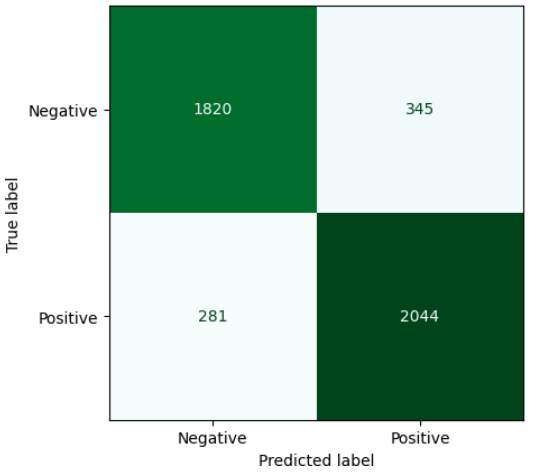
Classification Report:
precision recall f1-score support

Negative 0.86 0.85 0.86 2165
Positive 0.86 0.87 0.87 2325

accuracy 0.86 4490
macro avg 0.86 0.86 0.86 4490
weighted avg 0.86 0.86 0.86 4490
 
</td><td>Confusion Matrix - Logistic Regression (Unbalanced)
Accuracy: 0.86

Classification Report:
precision recall f1-score support

Negative 0.87 0.84 0.85 2165
Positive 0.86 0.88 0.87 2325

accuracy 0.86 4490
macro avg 0.86 0.86 0.86 4490
weighted avg 0.86 0.86 0.86 4490
 
</td></tr>
</table>

2- ROC curve: The code below shows how we developed our ROC curve for balanced data for both models (Naïve-Bayes and Logistic Regression):

```
# Generate ROC for Naive Bayes
y_score_nb = naive_bayes_model.predict_proba(X_test_balanced)[:, 1]
fpr_nb, tpr_nb, thresholds_nb = roc_curve(y_test_balanced, y_score_nb)
roc_auc_nb = auc(fpr_nb, tpr_nb)

# Generate ROC for Logistic Regression
y_score_lr = logistic_regression_model.predict_proba(X_test_balanced)[:, 1]
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test_balanced, y_score_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_nb, tpr_nb, color='navy', lw=2, label='ROC curve (Naive Bayes) (AUC = {:.2f})'.format(roc_auc_nb))
plt.plot(fpr_lr, tpr_lr, color='green', lw=2, label='ROC curve (Logistic Regression) (AUC = {:.2f})'.format(roc_auc_lr))
plt.plot([0, 1], [0, 1], color='darkorange', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

Figure 42 ROC Curve Code

ROC curve output:

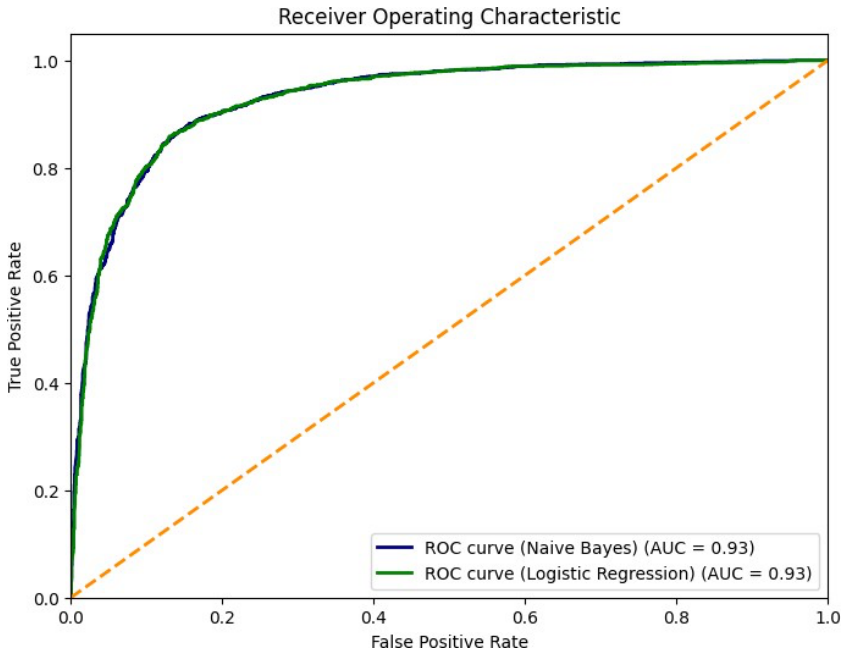


Figure 43 ROC Curve

Questions Discussion for Naïve-Bayes Model:

Does the model appear valid and accurate on the test data?	The naive Bayes model's accuracy is 0.90, which we consider high, and that would result in it being valid and accurate enough.
Does the model output/behavior make sense to the domain experts?	Considering the model's absence of unusual behavior and its high accuracy of 0.90, and we also we have a value of the ROC 0.93 which is close to 1, the model does make sense to the domain experts
Do the parameter values make sense in the context of the domain?	As our data has been successfully addressed to resolve common issues, it can be considered clean data. We have effectively employed the dataset for classification purposes. Therefore, the parameter values do align with the context of the domain.
Is the model sufficiently accurate to meet the goals?	Our main objective is to assess the satisfaction level of Threads Application users, gather valuable insights regarding their preferences and opinions, and identify any potential challenges they may be facing. By achieving this goal, we can significantly enhance the overall experience of Threads Application users. In this context, the obtained accuracy of 90% from the balanced dataset is considered sufficient to fulfill the objectives of the project.
Are more data or inputs needed?	It is possible that more data or inputs may be needed for further evaluation and potential improvement of the model's performance.

Questions Discussion for Logistic Regression Model:

Does the model appear valid and accurate on the test data?	Yes, the model appears to be valid and accurate on the test data with an accuracy score of 0.90 which is a high result.
Does the model output/behavior make sense to the domain experts?	Yes, the model output/behavior makes sense to the domain experts. Because we have a value of the ROC 0.93 curve that is near (1), the confusion matrix results are good.
Do the parameter values make sense in the context of the domain?	Before the classification step, we cleaned and stemmed all datasets and ensured that all parameters were within our scope. Hence, the parameters make sense within the context of the domain in both approaches
Is the model sufficiently accurate to meet the goals?	Yes, according to our classification results, the positive class is higher than the negative class. The model's accuracy score of 0.90 indicates that it is performing well and meeting the goals.
Are more data or inputs needed?	It is possible that more data or inputs may be needed for further evaluation and potential improvement of the model's performance.

Phase 4

1. Distribution of classification visualization:

In this section we want to visualize our results after analyzing the customer reviews of "Threads" regarding the initial hypothesis. Using these visualizations, we aim to understand the overall sentiment of the users towards the app, and to prove our initial hypothesis. We employ various frameworks and libraries such as pandas, matplotlib and sklearn to visualize the sentiments.
By utilizing pie and bar charts, we present the distribution of positive, negative, and neutral reviews in terms of both the number of reviews and their proportions to support our second hypothesis which is “Through data analysis of customer feedback, reviews, and ratings, we seek to measure and evaluate user satisfaction with the Threads app”. This visual representation allows us to gain insights into the users' opinions about the app and assess its overall reception.

Code explanation:

- Firstly, we initiate the process by reading the sentiment analysis file, followed by conducting a count of the classification column:

```
data = pd.read_csv('clean_reviews.csv')
```

```
import pandas as pd
# Load the dataset from CSV
clean_reviews = pd.read_csv('clean_reviews.csv')
# Calculate the value counts of the 'rating' column
classification_report = clean_reviews['sentiment'].value_counts()
```

Figure 44 count data

Visualize the classification results using a pie chart, which effectively displays the proportion of each sentiment classification (neutral, positive, negative):

```
classification_report.plot(kind="pie")
```

```
<Axes: ylabel='sentiment'>
```

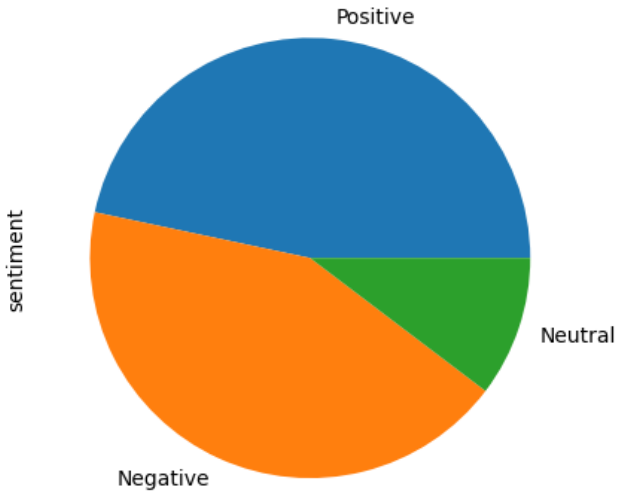


Figure 45 pie chart

visualize the classification results using a bar chart, which represents the number of tweets for each sentiment classification (neutral, positive, negative):

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('clean_reviews.csv')

# Count the number of reviews for each sentiment classification
sentiment_counts = df['sentiment'].value_counts()

# Create a bar chart
plt.bar(sentiment_counts.index, sentiment_counts.values)

# Set the labels and title
plt.xlabel('Sentiment')
plt.ylabel('Number of Reviews')
plt.title('Sentiment Classification Distribution')

# Show the bar chart
plt.show()
```

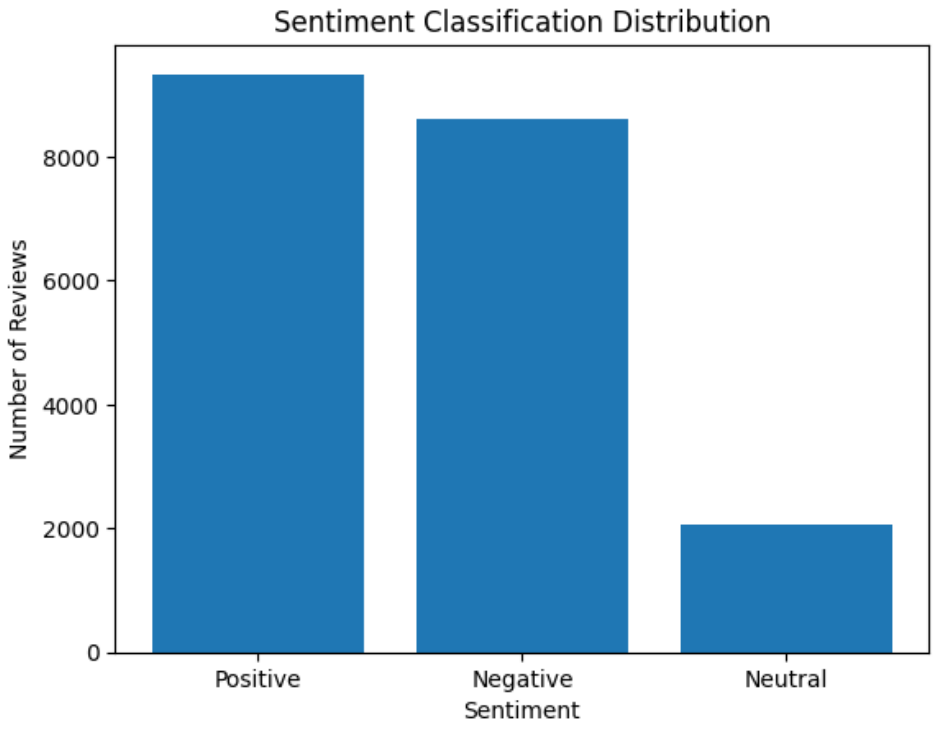


Figure 46 bar chart

2. Visualizing Common Positive Words:

We utilized a bar chart to visualize the top 10 positive words based on their frequency count. This approach allowed us to identify key terms and concepts associated with Threads. The following code illustrates the process and highlights the libraries employed (pandas, matplotlib and sklearn) in this analysis:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer

df = pd.read_csv('clean_reviews.csv')
positive_reviews = df[df['sentiment'] == 'Positive']['review_description']

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(positive_reviews)

word_counts = X.sum(axis=0)
feature_names = vectorizer.get_feature_names_out()

word_counts_df = pd.DataFrame({'word': feature_names, 'count': word_counts.flat})
word_counts_df = word_counts_df.sort_values(by='count', ascending=False)
top_n_words = 10
top_words_df = word_counts_df.head(top_n_words)

plt.figure(figsize=(10, 6))
plt.bar(top_words_df['word'], top_words_df['count'], color='skyblue')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top {} Positive Words'.format(top_n_words))
plt.xticks(rotation=45, ha='right')
plt.show()
```

Figure 47 Visualizing Common Positive Words.

The bar chart below shows common words in positive reviews. Notably, 'twitter' is the second most frequent term, indicating a preference for Threads over Twitter.

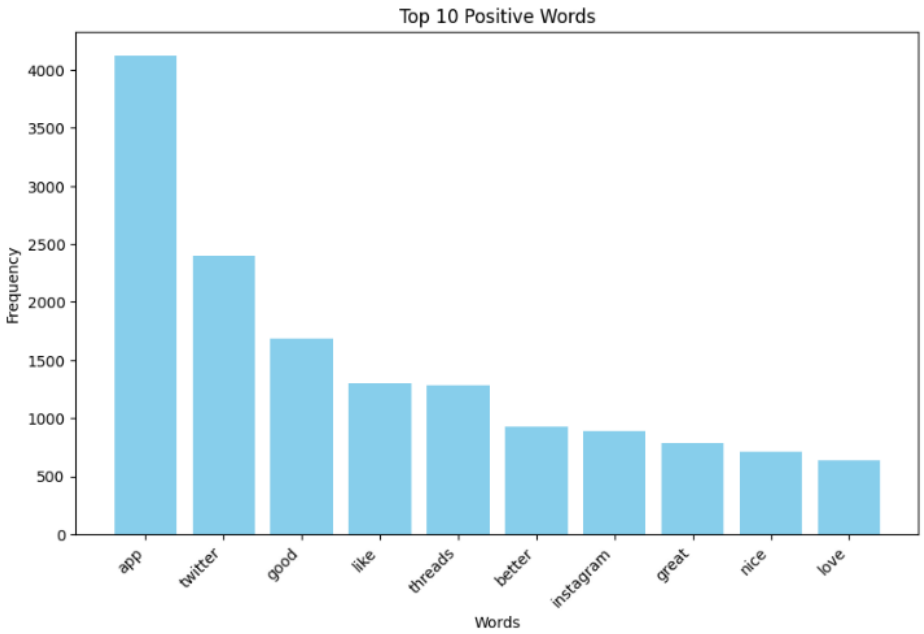


Figure 48 most Common Positive Words bar chart

3. Visualizing Common negative words:

Similar to what we did in the above code, but this time it is for the negative words, The code below shows what we did (using the same libraries):

```
[ ]
    negative_reviews = df[df['sentiment'] == 'Negative']['review_description']

    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(negative_reviews)

    word_counts = X.sum(axis=0)

    feature_names = vectorizer.get_feature_names_out()

    word_counts_df = pd.DataFrame({'word': feature_names, 'count': word_counts.flat})
    word_counts_df = word_counts_df.sort_values(by='count', ascending=False)

    top_words_df = word_counts_df.head(top_n_words)

    plt.figure(figsize=(10, 6))
    plt.bar(top_words_df['word'], top_words_df['count'], color='lightcoral')
    plt.xlabel('Words')
    plt.ylabel('Frequency')
    plt.title('Top {} Negative Words'.format(top_n_words))
    plt.xticks(rotation=45, ha='right')
    plt.show()
```

Figure 49 Visualizing Common Negative words

The bar chart below shows common words in negative reviews. Notably, 'Twitter' is the second most frequent term, indicating a preference for Twitter over Threads, which is related to our first hypothesis.

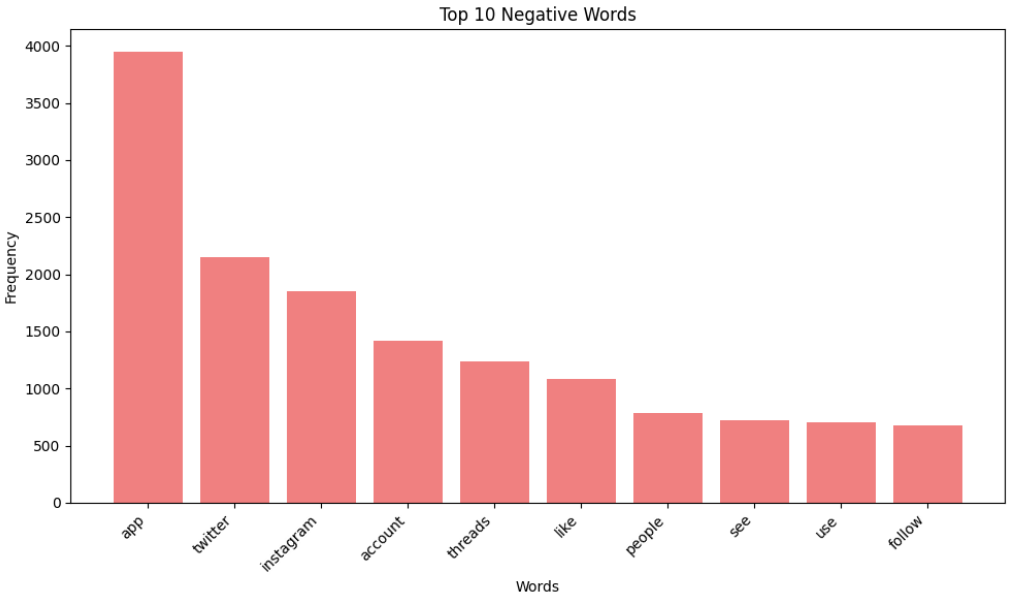


Figure 50 most Common Negative Words bar chart

4. Most frequent words in both negative and positive reviews:

Since our first hypothesis is “we aim to compare user satisfaction levels between Twitter and the Threads app”. We can see clearly from the second and the third visualizations, that the second most frequent word in both positive and negative was twitter, and the fifth one was threads, with a near number of frequencies, which means that people were neutral, part of them preferred twitter and part of them preferred Threads. To be more concise of what we did, we developed a bar chart combining both. The code and bar chart below shows this comparison:

```
# Merge dataframes on the 'word' column
merged_df = pd.merge(top_words_df_pos, top_words_df_neg, on='word', how='outer').fillna(0)

# Plotting
plt.figure(figsize=(12, 8))
bar_width = 0.35
index = range(len(merged_df))

plt.bar(index, merged_df['count_pos'], width=bar_width, color='skyblue', label='Positive')
plt.bar(index, merged_df['count_neg'], width=bar_width, color='lightcoral', label='Negative', align='edge')

plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top {} Words in Positive and Negative Reviews'.format(top_n_words))
plt.xticks(index, merged_df['word'], rotation=45, ha='right')
plt.legend()
plt.show()
```

Figure 51 Most frequent words in both negative and positive reviews

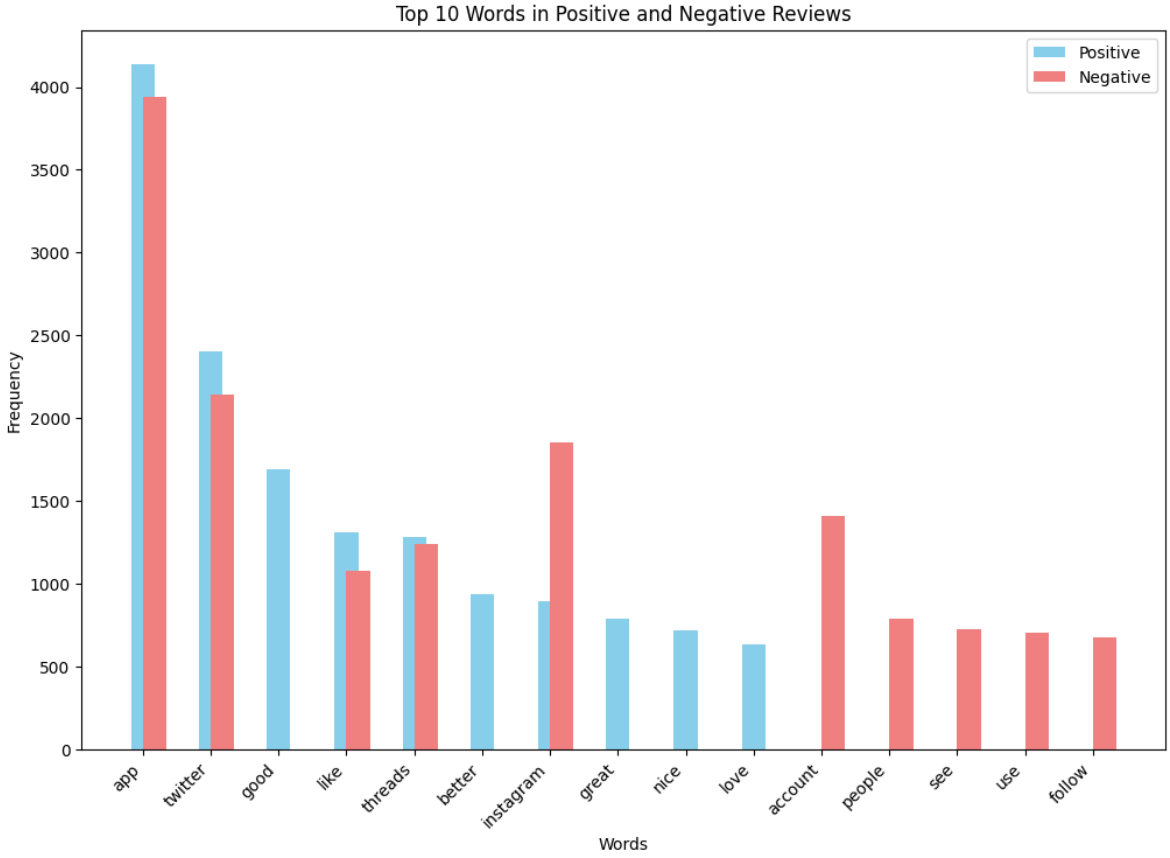


Figure 52 Most frequent words in both negative and positive reviews bar chart

5. Findings:

Based on our visualization we have been reach to:

- The reviews with positive sentiment have the highest number among all sentiments categories.

H1: By analyzing user feedback, ratings, and sentiment analysis, we aim to compare user satisfaction levels between Twitter and the Threads app.

The findings indicate that there was a mixed sentiment among users when comparing the two platforms. In both positive and negative reviews, the second most frequent word was "Twitter," suggesting a preference for Twitter over Threads. However, the term "Threads" also appeared frequently, indicating that there were users who preferred the Threads app. This finding suggests that user satisfaction levels varied between the two platforms, with no clear dominance of one over the other.

H2: Through data analysis of customer feedback, reviews, and ratings, we seek to measure and evaluate user satisfaction with the Threads app.

This finding suggests that overall, users have a positive perception of the Threads app based on the analyzed customer feedback. The higher proportion of positive sentiments indicates a higher level of user satisfaction, supporting the hypothesis that sought to measure and evaluate user satisfaction with the Threads app.

H3: Conducting a comprehensive analysis, we aim to compare user engagement, satisfaction, feature preferences, content relevance, user experience, and performance between Twitter and the Threads app.

Based on the analysis of positive and negative word visualization, the word "App" appeared as the most frequent term in the positive chart, while the words "Twitter" and "Threads" ranked second and fifth respectively in both positive and negative charts. This suggests that the level of user engagement, satisfaction, feature preferences, content relevance, user experience, and performance between Twitter and the Threads app appears to be balanced or neutral.

6. Recommendations

- Collecting additional data to improve the accuracy and reliability of our analysis.
- Ensure the collected data is accurate, complete, and reliable. By implementing data validation techniques, perform thorough data cleaning and address any missing or inconsistent values.
- To enhance the quality of our analysis, it is recommended to gather the data from different sources.
- To enhance our analysis, it is advised to explore new libraries and tools.

References

[1] Threads (social network) (2023) *Wikipedia*. Available at: (https://en.wikipedia.org/wiki/Threads_(social_network) (Accessed: 29 September 2023).

[2] The absolute basics for beginners# (no date) *NumPy*. Available at: https://numpy.org/doc/stable/user/absolute_beginners.html (Accessed: 30 September 2023).

[3] Visualization with python (no date) *Matplotlib*. Available at: https://matplotlib.org/ (Accessed: 30 September 2023).

[4] (No date) *Pandas introduction*. Available at: https://www.w3schools.com/python/pandas/pandas_intro.asp (Accessed: 30 September 2023).

[5] M, S. (2022) *NLTK: A beginners hands-on guide to natural language processing*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2021/07/nltk-a-beginners- hands-on-guide-to-natural-language-processing/ (Accessed: 30 September 2023).

[6] Low-code data app development (no date) *Plotly*. Available at: https://plotly.com/ (Accessed: 30 September 2023).

[7] Google-play-scraper (no date) *PyPI*. Available at: https://pypi.org/project/google-play- scraper/ (Accessed: 30 September 2023).

[8] App-store-scraper (no date) *PyPI*. Available at: https://pypi.org/project/app-store-scraper/ (Accessed: 30 September 2023).

[9] Jhalani, S. (2023) *Threads, an Instagram app reviews*, *Kaggle*. Available at: https://www.kaggle.com/datasets/saloni1712/threads-an-instagram-app- reviews?select=threads_reviews.csv (Accessed: 29 September 2023).

[10] Google-play-scraper (no date) *PyPI*. Available at: https://pypi.org/project/google-play- scraper/#description (Accessed: 30 September 2023).

[11] Logistic regression (no date) Logistic Regression - an overview | ScienceDirect Topics. Available at: https://www.sciencedirect.com/topics/computer-science/logistic-regression (Accessed: 16 November 2023).

[12] naive Bayes (no date) scikit. Available at: https://scikit-learn.org/stable/modules/naive_bayes.html (Accessed: 16 November 2023).