Department of Applied Data Science
DATA 228

Big Data Technologies
Under the Guidance of Dr. Guannan Liu

PROJECT REPORT
Social Media Listening using Reddit Comments

Group 5

Group Members
Eshita Gupta
Madhu Vishwanath Burra
Monica Lokare
Shrinivas Bhusannavar
Veena Ramesh Beknal

Table of Contents

Context and problem statement

Social media usage has become a key part of everyone's life. Over the last few years, research by the Pew Research Center has found that nearly 70% of Americans use some form of social media. Also, as of 2021, YouTube and Reddit were the only 2 platforms measured that saw statistically significant growth since 2019 [link]. Around 22% of American users say that they use Reddit [link], a vibrant platform for mostly text-based conversations in interest-based communities called subreddits with users spending an average of 24 mins per day on it [link]. Reddit is an exclusive social media platform with different communities called subreddits where users can discuss anything under the sun. Given the mindshare that Reddit gets and also given the honest discourse (due to anonymity) that occurs mostly over text, it is a key channel for understanding user behavior and uncovering insights about how they feel about various brands and products. With millions of active users and countless discussions happening daily, Reddit plays a significant role in shaping internet culture and influencing trends across various industries. Reddit comments offer unfiltered and authentic insights into consumer opinions, preferences, and sentiments. Users on Reddit often express their thoughts and experiences in detail, providing rich data for analysis.

Social media listening, also known as social media monitoring or social media intelligence, refers to the process of tracking and analyzing conversations, mentions, and trends across social media platforms**.** Brands and products can benefit greatly from understanding what their existing and potential users have to say since they can address feedback and incorporate improvements to achieve better favorability from their target populations. For companies to grasp customer preferences, views, and actions today, it is important that social media listening becomes part of the strategy, due to the digital era we are in. Insights from social media monitoring help businesses make marketing plans or create products among others.

Given that Reddit is a rich source of information and the data is relatively easy to obtain compared to other social media channels, we attempted to build a social media listening using two approaches – historical data and real-time data. These approaches represent two paradigms in social media listening. Historical data gives us a retrospective view wherein we can study trends over time and obtain aggregated metrics over a larger volume of data. This can help with planning specific campaigns across certain times of the year. Real-time gives us a more immediate pulse of what users are talking about and this can help in gauging the sentiment that's occurring in real-time which is beneficial for ongoing marketing campaigns, events, AMAs and promotions.

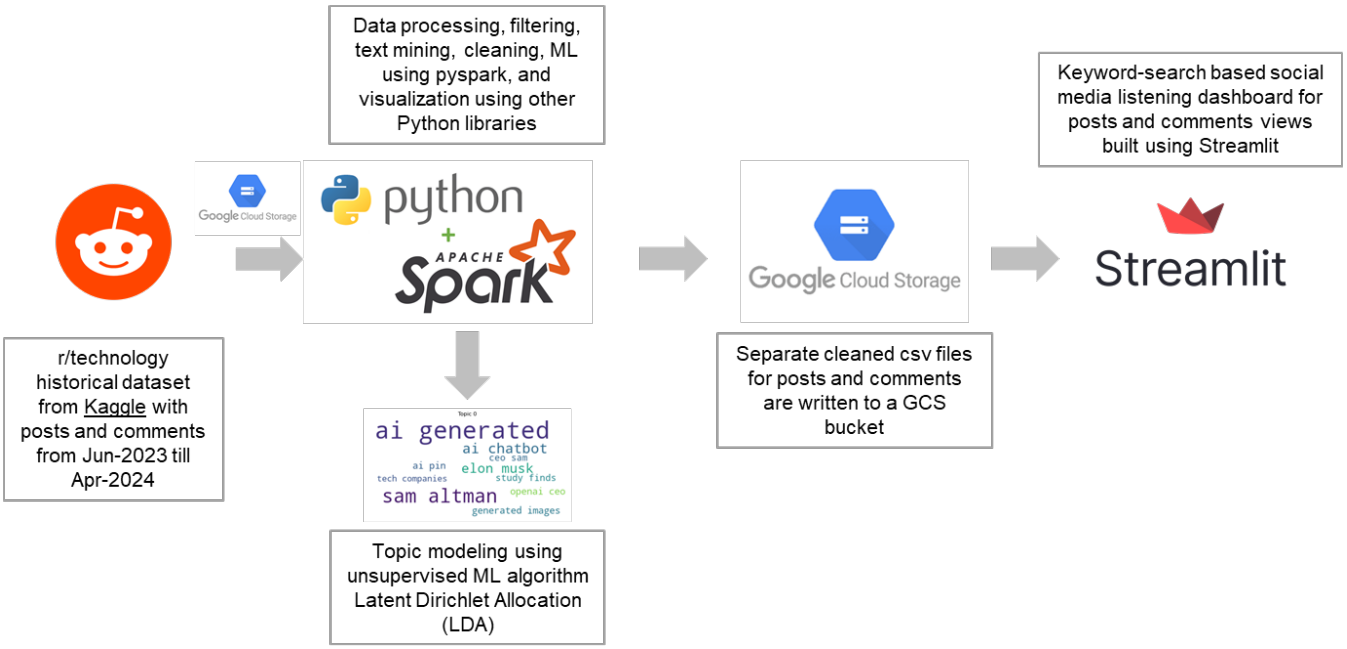
Solution overview – historical data

Streamlit dashboard process flow

The primary source of historical data was the r/technology subreddit's historical dataset which is updated on a weekly level and is available here. The r/technology subreddit is a vibrant subreddit with over 16M members and is a vibrant online community for discussing the latest news, trends, and updates in the ever-evolving technology space. From this dataset, as of the end of April 2024, there are over 1.4M comments across 15k posts in the historical data (starting from June 2023). We used PySpark to process this data at scale.

For our analysis, we first performed exhaustive exploratory data analysis on the posts and comments data separately, after which we calculated multiple metrics like reach, impressions, sentiment, and share of voice and visualized information that was rendered in a Streamlit web application. These metrics help products, brands, and other concerned parties to understand the pulse of customers and their perception about certain topics, brands, and products from an unfiltered open discussion on Reddit. The flow diagram describing the set of steps involved in going from raw data to web application is shown in Figure 1.

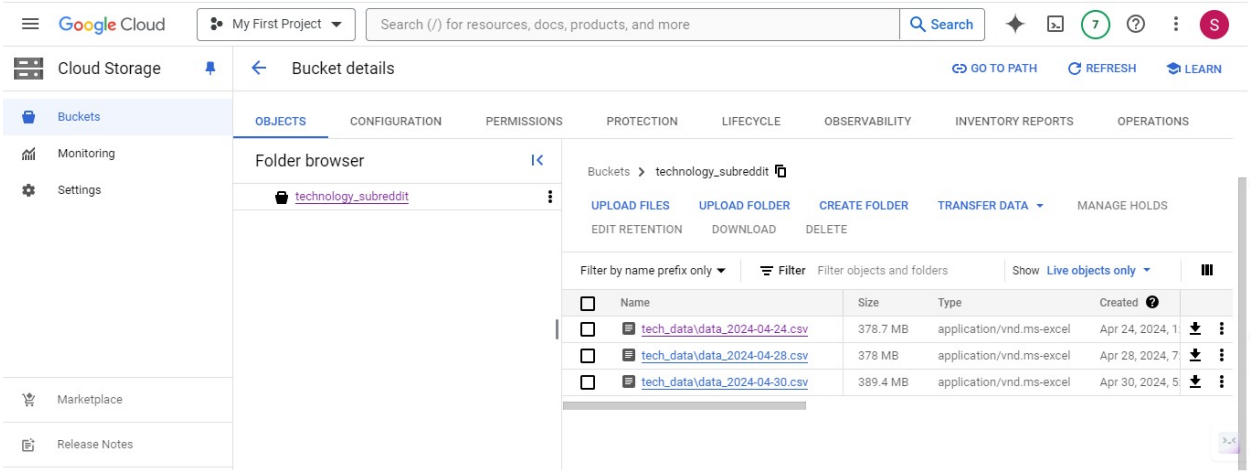
Figure 1: Process flow for historical data processed and fed into Streamlit



This flow diagram outlines the process of social media listening using historical data from Reddit, specifically from the r/technology subreddit. The diagram describes a sequence of operations performed on the dataset to build a keyword-search-based dashboard for social media listening and analysis. A summary of the steps are described below:

1. As referenced earlier, the raw data is obtained from the source and this is written to a Google Cloud Storage bucket using a batch-processing approach as shown in Figure 2
2. The raw data is segregated into posts and comments and each data split goes through multiple processing steps using PySpark for processing large-scale data efficiently. The processing includes filtering, text mining, text cleaning, visualization, and Machine Learning. Spark's in-memory data processing helps in performing these steps quickly and efficiently.
3. As a standalone one-time process, an unsupervised machine learning algorithm, Latent Dirichlet Allocation (LDA), is used to dissect the posts' title text and identify topics within it. This approach facilitates the summarizing of the post titles into key themes. We will elaborate on the approach in upcoming sections of the report.
4. After processing, the cleaned data is separated into CSV files for post titles and comments respectively and written to a Google Cloud Storage (GCS) bucket. This storage acts as a repository for the processed data, making it accessible for further analysis or downstream applications/dashboards.
5. Finally, the processed data is used to build a dashboard using Streamlit. The dashboard provides a user-friendly interactive interface to analyze social media metrics and trends based on keyword searches on posts and comments views respectively.

Figure 2: GCS bucket for batch processing on historical Kaggle dataset



Data overview and analysis of historical data

Anatomy of the Reddit Post

To understand Reddit data, we must first understand what a typical Reddit post looks like. Figure 3 visually demonstrates how a typical Reddit post on r/technology renders on a browser. Markings and annotations have been overlayed on the snapshot for ease of understanding and interpreting the features that we're working within our dataset.

Figure 3: Snapshot of a Reddit post with visual cues to illustrate data features

Features highlighted in the snapshot:

1. Post the author in the chosen subreddit
2. Title of post
3. Post upvotes and comments count
4. Ad – not considered in our analysis
5. Top poster username
6. Top poster comment (called root or parent comment)
7. Top poster upvotes count
8. Next comment (called nested/child comment)

Data Processing & transformation using Spark

PySpark is used as the backbone for data processing due to its ability to handle large datasets efficiently. It is utilized for data cleansing, transformation, and aggregation of Reddit data, which typically involves a high volume of posts and comments. We have performed data preprocessing and transformation using PySpark which is an Apache Spark Python API open-source system for processing and analyzing Reddit datasets. It's a way to program Spark with Python by providing a convenient interface for doing so. As we could take advantage of the scalability and performance offered by Spark when dealing with data-intensive tasks, RDDs (Resilient Distributed Datasets) allow us to do distributed computing, and DataFrames helps in easily working with data at scale. As Reddit comments data was quite large, Spark was the most convenient option for us to process the large comments data. Employed for its NLP capabilities, TextBlob analyzes sentiment and subjectivity in Reddit comments, which aids in gauging public opinion and emotional tone.

Libraries such as Pandas: Used for handling smaller subsets of data for detailed analysis and quick transformations which are less intensive than those handled by PySpark. NumPy: Provides support for operations on arrays and matrices, which are essential for numerical computations in data analysis Seaborn and Matplotlib: These libraries are critical for visualizing data, helping to identify trends, patterns, and outliers in social media engagement on Reddit. re and nltk: These are crucial for text processing and NLP tasks. They help in cleaning and preparing the textual data extracted from Reddit for deeper analysis. Also, some other of the Pyspark libraries were used are StopWordsRemover, NGram, Tokenizer, SentimentIntensityAnalyzer, pyspark.sql, pyspark.sql.functions, pyspark.ml, pyspark.ml.feature. These libraries together provide the necessary tools and functionalities for processing, transforming, and analyzing Reddit posts and comments data using PySpark. These libraries collectively facilitate the effective transformation and analysis of social media data from Reddit, enabling insights into user behavior, trends, and sentiment. The notebook **PySpark_Reddit_DataProcessing.ipynb** contains the code implementation for PySpark

Natural Language Processing

Natural Language Processing (NLP) refers to the branch of artificial intelligence that gives the machine the capability to read, understand, and derive meaning from human language. It is a deciphering language structure. In our project, we have utilized various NLP techniques like tokenization, stop word removal, removing non-ASCII characters, and sentiment analysis. Tokenization is a process of breaking down sentences into word fragments, resulting in a word index and tokenized text that can be represented as numerical tokens. Stop word removal involves eliminating nonessential words like 'are', 'the', 'a',' an', etc. that do not contribute information to

the text's meaning. Also, we have done other additional processing like removing non-ASCII characters and treating URLs using Regex.

We have also performed stemming and lemmatization for post titles and comments. PySpark does not have a native stemmer or lemmatizer in MLlib. Therefore, we performed stemming or lemmatization, using a user-defined function (UDF) and then applied it to the DataFrame. Stemming is a technique used to reduce an inflected word down to its root word. For example, changing, change, and changed can be rooted to chang word. Similarly when we lemmatize the above example the root word for lemmatizing becomes change.WordNetLemmatizer is used for lemmatizing words, i.e., reducing them to their base or root form.

Sentiment analysis, also known as opinion mining, is the process of analyzing large volumes of text to determine whether it expresses positive, negative, or neutral sentiments. Organizations can use sentiment analysis to improve customer support, build a strong brand presence, and conduct market research.

Exploratory Data Analysis – post titles

Figure 4 is a snapshot of the raw data from the source. The raw data contains the posts and comments together.

Figure 4: Raw data snapshot

	register_index	post_id	comment_id	author	datetime	title	url	score	comments	text	author_post_karma	tag
1	145hyeyjnlh1vx	145hyey	jnlh1vx	Illustrious_Risk3732	2023-06-09 23:42:43	NaN	NaN	1.0	NaN	He even gives him self Awards to make sure he ...	17109.0	Social Media
2	1453u2ojnjtk7f	1453u2o	jnjtk7f	ProdigiousPlays	2023-06-09 16:47:51	NaN	NaN	1.0	NaN	RIF already announced it s shutting down at th...	118844.0	Social Media
3	1453u2ojnjtv9u	1453u2o	jnjtv9u	YJSubs	2023-06-09 16:49:52	NaN	NaN	1.0	NaN	I wonder which sub left on that day	50085.0	Social Media
4	1453u2ojnjusm9	1453u2o	jnjusm9	jamnewton22	2023-06-09 16:55:57	NaN	NaN	1.0	NaN	Oh I m sure this will go well lol	171705.0	Social Media
5	1453u2ojnjvwlv	1453u2o	jnjvwlv	TriLink710	2023-06-09 17:03:15	NaN	NaN	1.0	NaN	Oh boy i can t wait for him to say the devs a...	70192.0	Social Media
6	1453u2ojnjt1r4	1453u2o	jnjt1r4	zshift	2023-06-09 16:44:29	NaN	NaN	1.0	NaN	That was almost 6 years ago fuck I thought it...	26996.0	Social Media
7	1453u2ojnjvk6f	1453u2o	jnjvk6f	franky3987	2023-06-09 17:00:57	NaN	NaN	1.0	NaN	It s been a few years since I saw that post ...	2236.0	Social Media

A brief description of the various columns is given below:

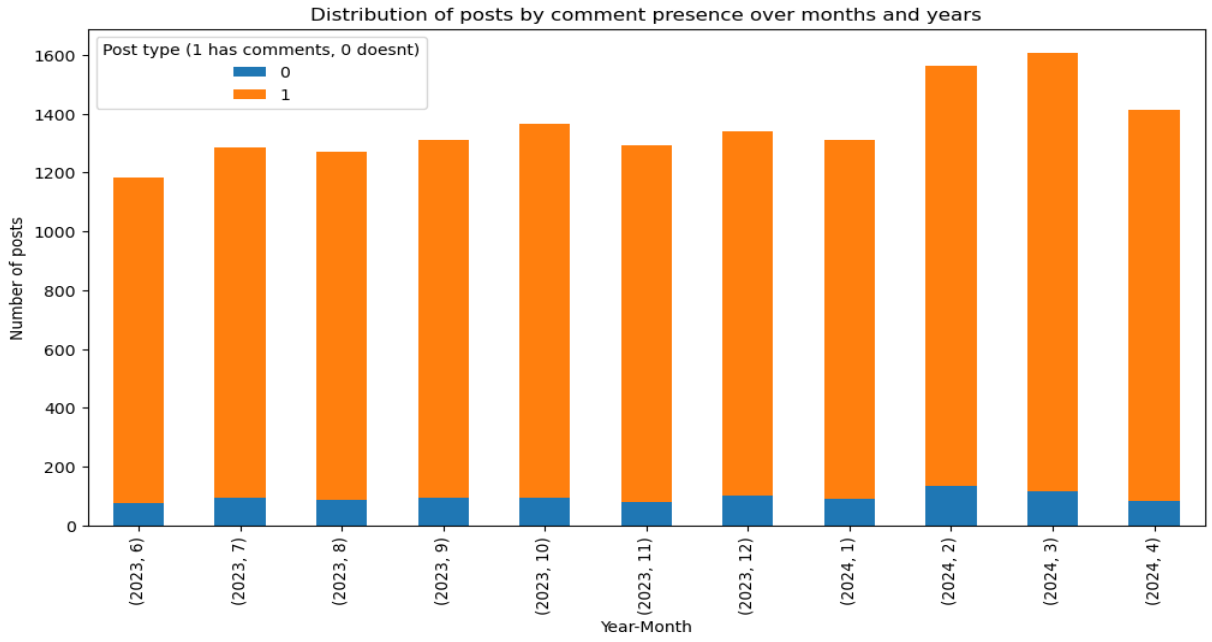
- **register_index**: A unique identifier for each row which we are excluding
- **post_id**: unique identifier for posts in r/technology
- **comment_id**: unique identifier for comments associated to a post
- **author**: Username of the post author or commentor
- **datetime**: Timestamp of post or comment creation
- **title**: Post title summarizing the topic, this is “NaN” for comments
- **url**: Web address associated with the post, this is “NaN” for comments

- **score**: Rating or score received by the post
- **comments**: Total number of comments for a post, this is “NaN” for comments
- **text**: Actual content of the post or comment
- **author_post_karma**: The total net score for the user, referred to as “karma” as per Reddit terminology
- **tag**: tags associated with the post, this is a mandate for the subreddit

Posts distribution over time

The data contains over 15k unique posts and 1.4M comments. Given this considerable difference in volume, we decided to split the data into posts and comments separately. This helps us analyze post titles and comments separately. Post titles are shorter and less noisy, while comments are varied and may contain more noise. For this reason, we decided to use only post titles for unsupervised machine learning.
Once we separate the posts as a separate dataset, we can further classify posts as those with comments and the rest, without. Out of the total 15k posts between Jun-2023 and Apr-2024, we observed ~1.3k posts per month on average. Around 7% of the posts are “orphan posts” (posts without comments) and these were removed since there's no engagement in these posts. Figure 5 shows the distribution of posts by comment presence over months.

Figure 5: Trend of post volume, stacked by comment presence



Posts distribution by comment volume

After excluding the posts without comments, we studied the distribution of posts based on comment volume. Figure 6 illustrates a histogram of posts based on comment volume, we can observe that the distribution is right-skewed with most posts having <200 comments and the same skewed distribution can be observed in the violin plot shown in Figure 7. The median comments per post is 26 and the 90th percentile value is 386.

Figure 6: Histogram of posts based on comment volume

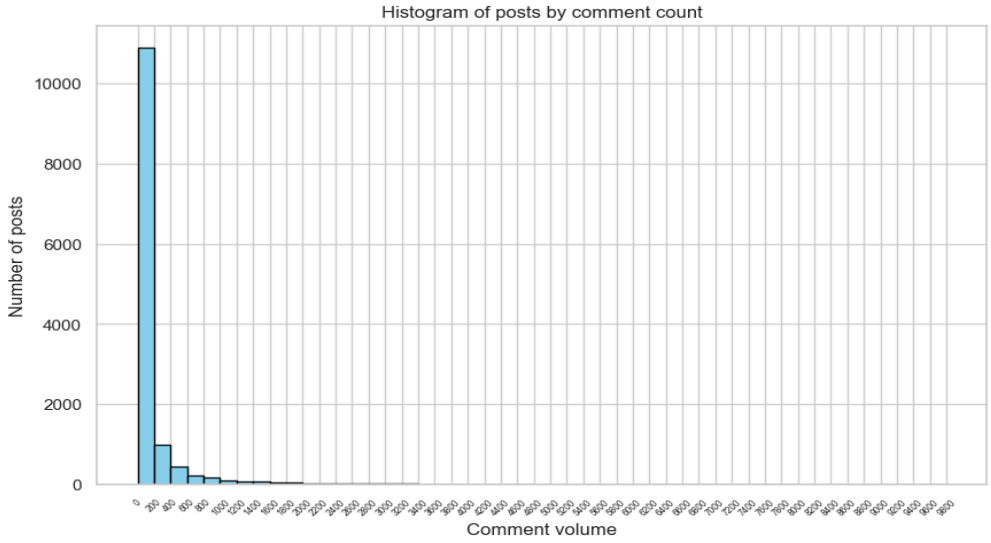
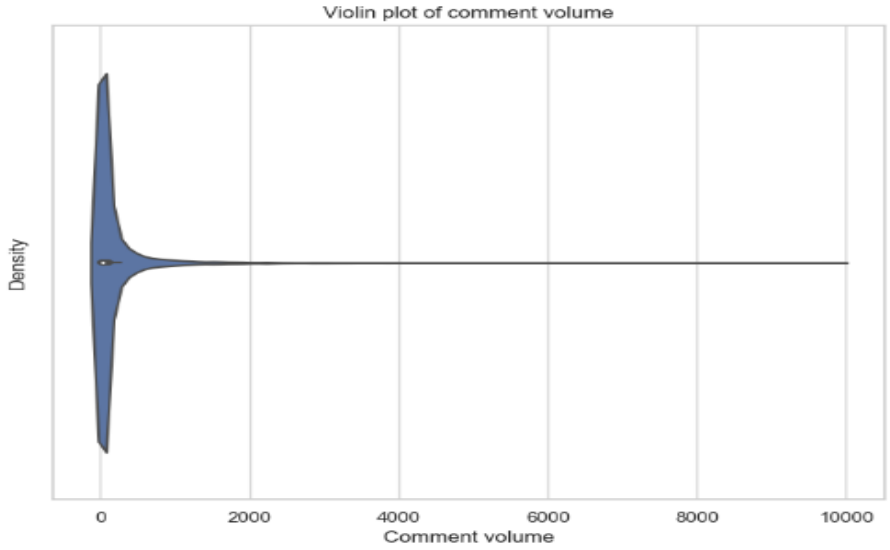


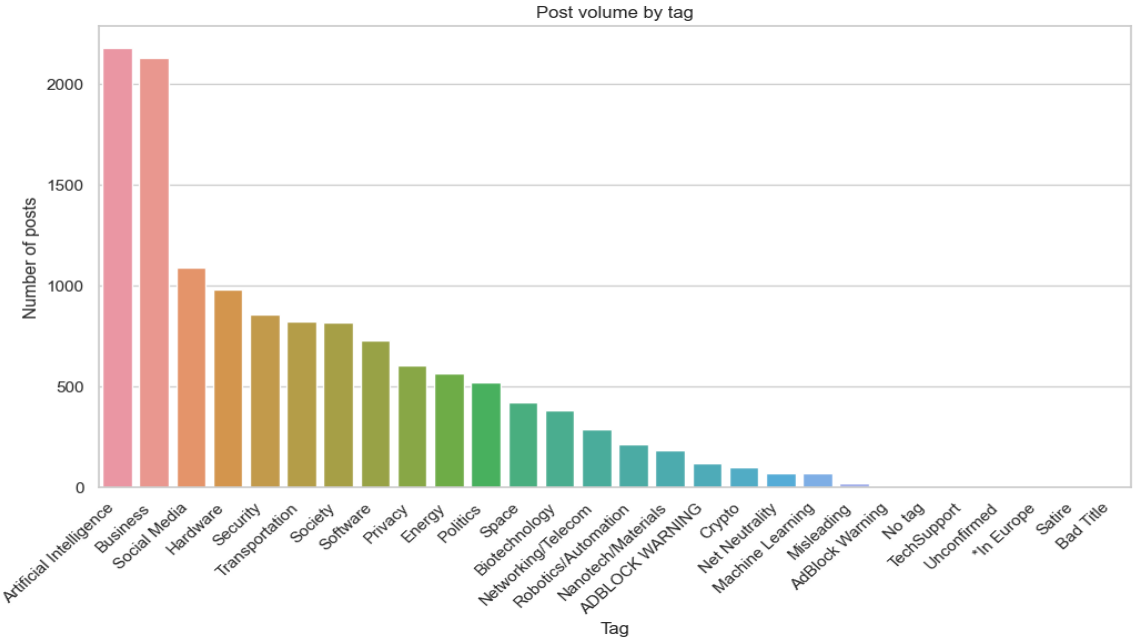
Figure 7: Violin plot of posts based on comment volume



Posts distribution by tags

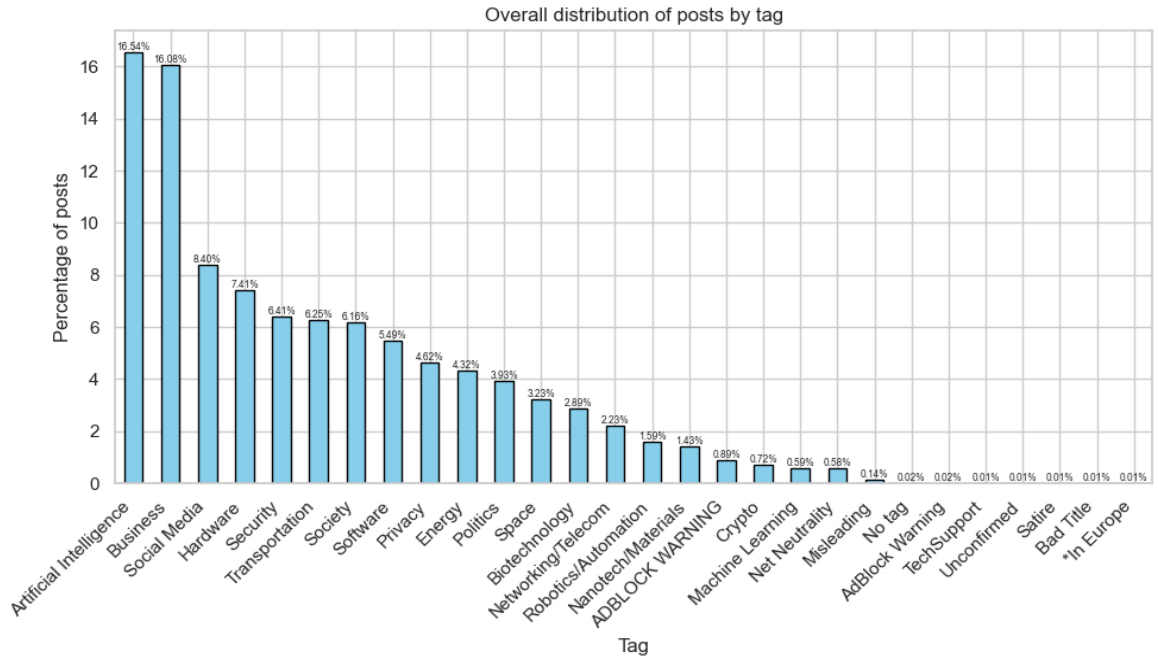
Tags are mandated in the r/technology subreddit by the group's moderators and this helps in categorizing posts based on content. Figure 8 shows a visual summary of post volume by different tags for all posts with comments. Given the recent wave of AI advances, the 'Artificial Intelligence' tag showed the largest post volume for the entire time period, with >2k posts accounting for nearly 17% of all posts. The business tag was a close second with similar volume and share.

Figure 8: Post volume by tag



The share of post volume by tag can be observed in Figure 9.
The top 10 tags collectively amounted to 82% of the post volume with a clear right-skewed distribution.

Figure 9: Post volume share by tag



Word frequency distribution of post title text

Now that we have understood the overall statistics and distribution of posts, we can study the text composition of the post title. Figure 10 illustrates the unigram (single word distribution by frequency) of the words in post title text. Figure 11 visualizes the bigram (two words) distribution from the post title. The unigrams and bigrams tend to be dominated by popular terms, trends, and personalities from the technology world. The distributions are dominated by Artificial Intelligence and Business terms since these two tags formed a combined 33% of all posts.

Figure 10: Top 20 unigrams in post text

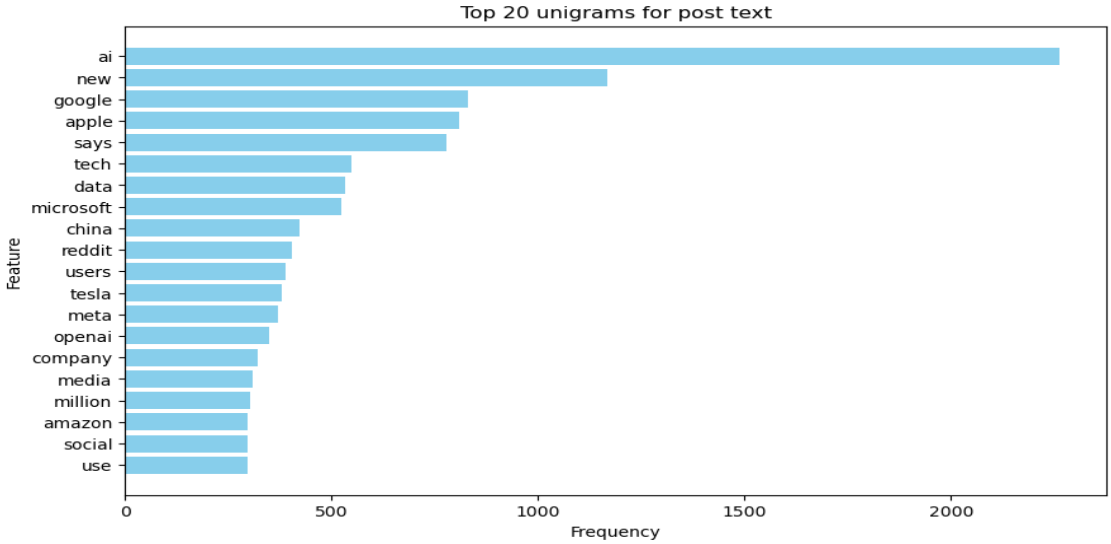
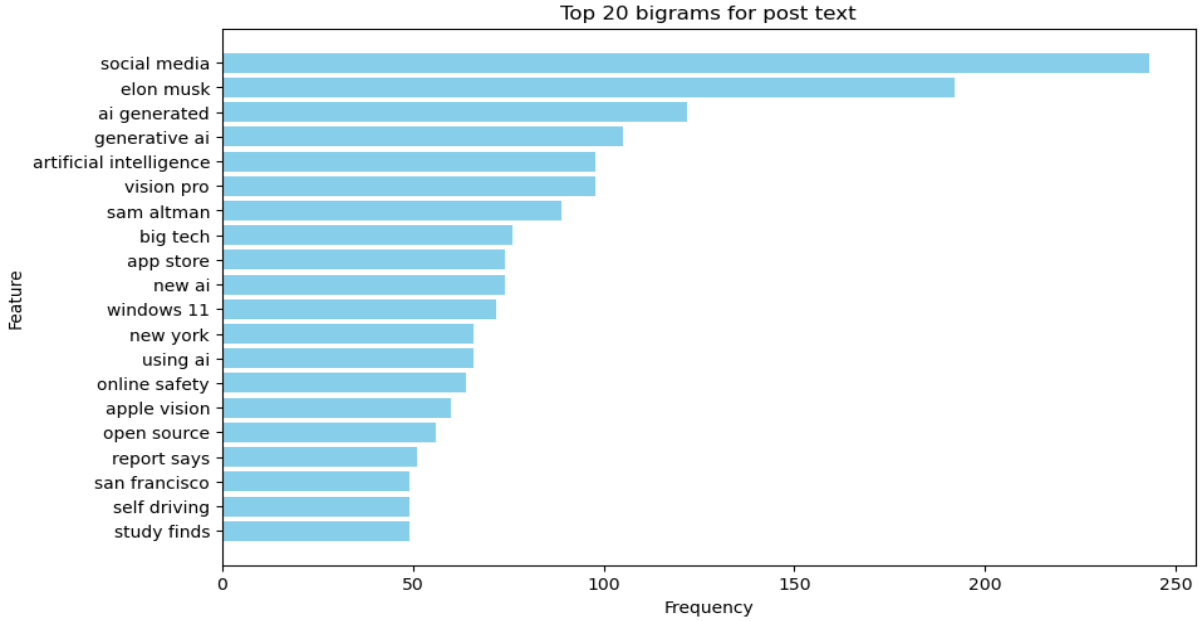


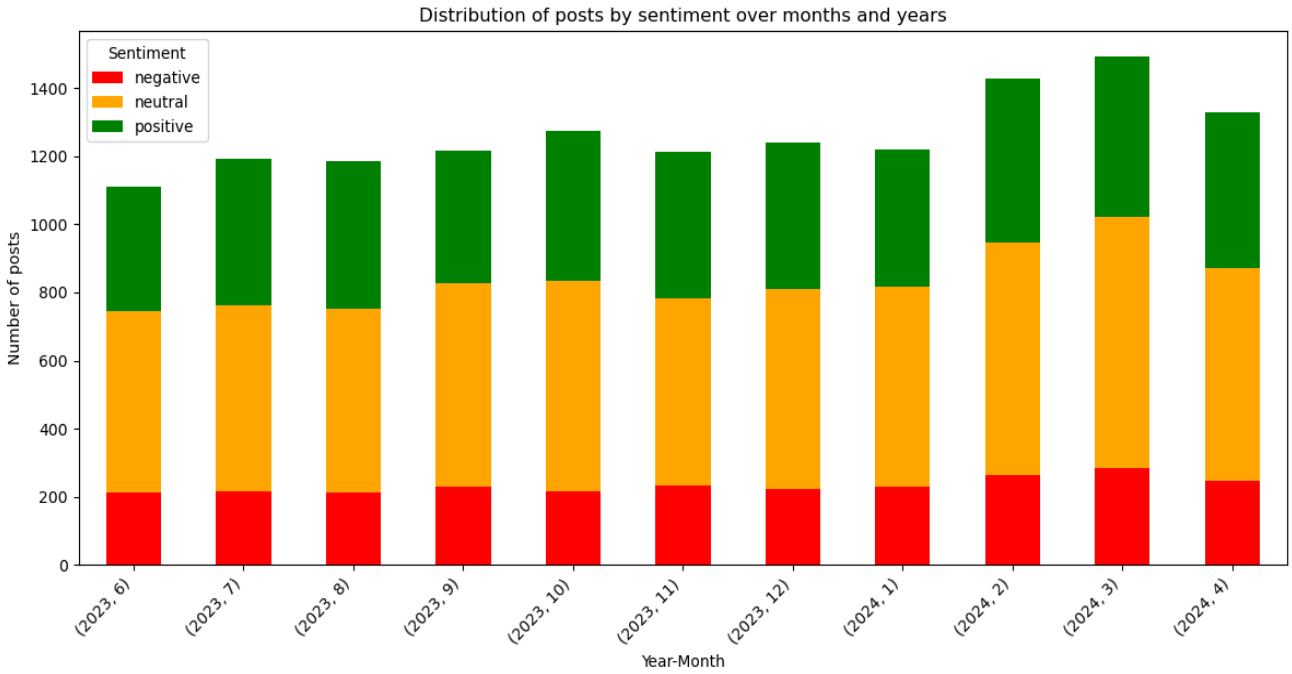
Figure 11: Top 20 bigrams in post text



Sentiment distribution of post titles

Sentiment was extracted using the Python textblob library which provides a sentiment polarity value ranging from -1 to 1. Highly negative sentiment is indicated by -1, whereas highly positive is indicated as 1, and neutral is represented by 0. Figure 12 shows the distribution of sentiment across time and it's evident that most posts seem to have a neutral sentiment.

Figure 12: Distribution of sentiment of post titles across months



Text cleaning performed on post titles

Operations like changing the post title to lowercase, removal of English stopwords, removal of HTML tags and URLs, removal of emoji's and non-ASCII characters were performed using PySpark. Contractions (i.e. expansions of "i'm" to "i am") were also expanded. Figure 13 shows the pre and post-cleaning distributions of title length while Figure 14 shows the pre and post-cleaning distributions of title word count. After performing text cleaning, there is not much difference observed in character and word counts between raw and cleaned post titles. This is not surprising since post titles are largely clean.

Figure 13: Comparison of post title length, before and after cleaning operations

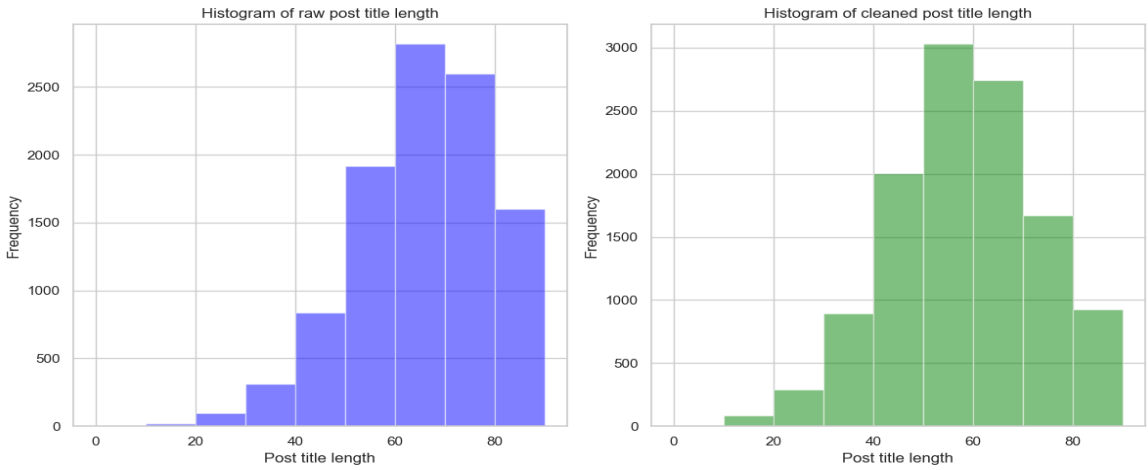
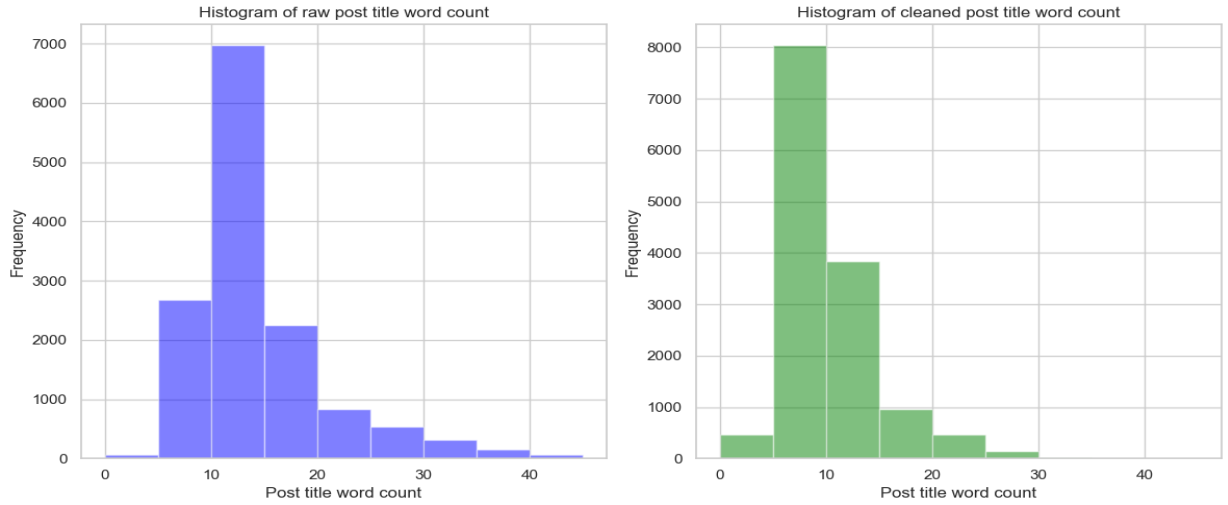


Figure 14: Comparison of post title word count, before and after cleaning operations



Exploratory Data Analysis – comment text

Figure 15 is a snapshot of the comments data that was separated as explained earlier.

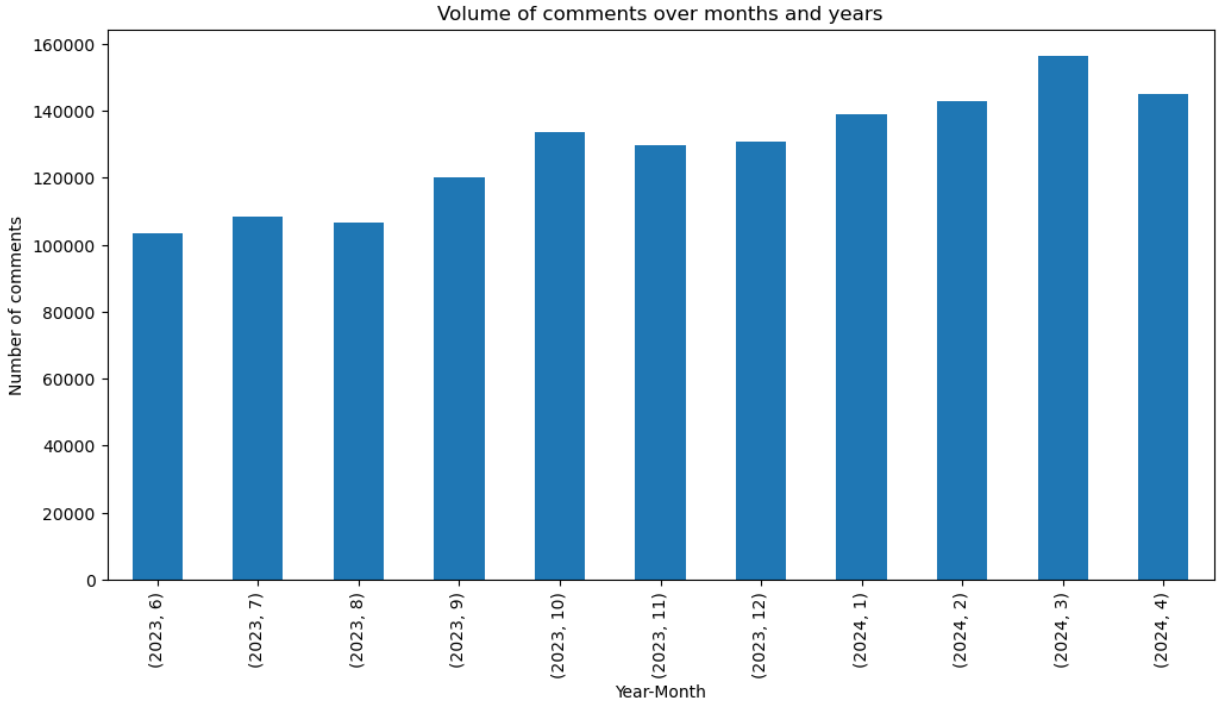
Figure 15: Raw data snapshot of comment data

post_id	comment_id	author	datetime	title	url	score	comments	text	author_post_karma	tag
145hyey	jnlh1vx	Illustrious_Risk3732	2023-06-09 23:42:43	NaN	NaN	1.0	NaN	He even gives him self Awards to make sure he ...	17109.0	Social Media
1453u2o	jnjtk7f	ProdigiousPlays	2023-06-09 16:47:51	NaN	NaN	1.0	NaN	RIF already announced it s shutting down at th...	118844.0	Social Media
1453u2o	jnjtv9u	YJSubs	2023-06-09 16:49:52	NaN	NaN	1.0	NaN	I wonder which sub left on that day	50085.0	Social Media
1453u2o	jnjusm9	jamnewton22	2023-06-09 16:55:57	NaN	NaN	1.0	NaN	Oh I m sure this will go well lol	171705.0	Social Media
1453u2o	jnjvwlv	TriLink710	2023-06-09 17:03:15	NaN	NaN	1.0	NaN	Oh boy i can t wait for him to say the devs a...	70192.0	Social Media

EDA for comment text volume

The r/technology subreddit had over 1.4M comments across posts (between Jun-2023 and Apr-2024) with a monthly average of around 120k comments. Figure 16 shows the volume of comments by month and year.

Figure 16: Trend of comment volume over time



Histogram for comment text length

To reduce noise, we only considered comments with at least 5 characters in length. By excluding the comments with less than 5 characters in length, we excluded around 1000 records which amounted to 0.06% reduction from the raw comments data and this is negligible. Figure 17 shows the histogram of comment length for those comments with over 5 characters and we can observe a right-skewed distribution with most comments having <200 character count and the same skewed distribution can be observed in the violin plot in Figure 18.

Figure 17: Histogram of comment character count

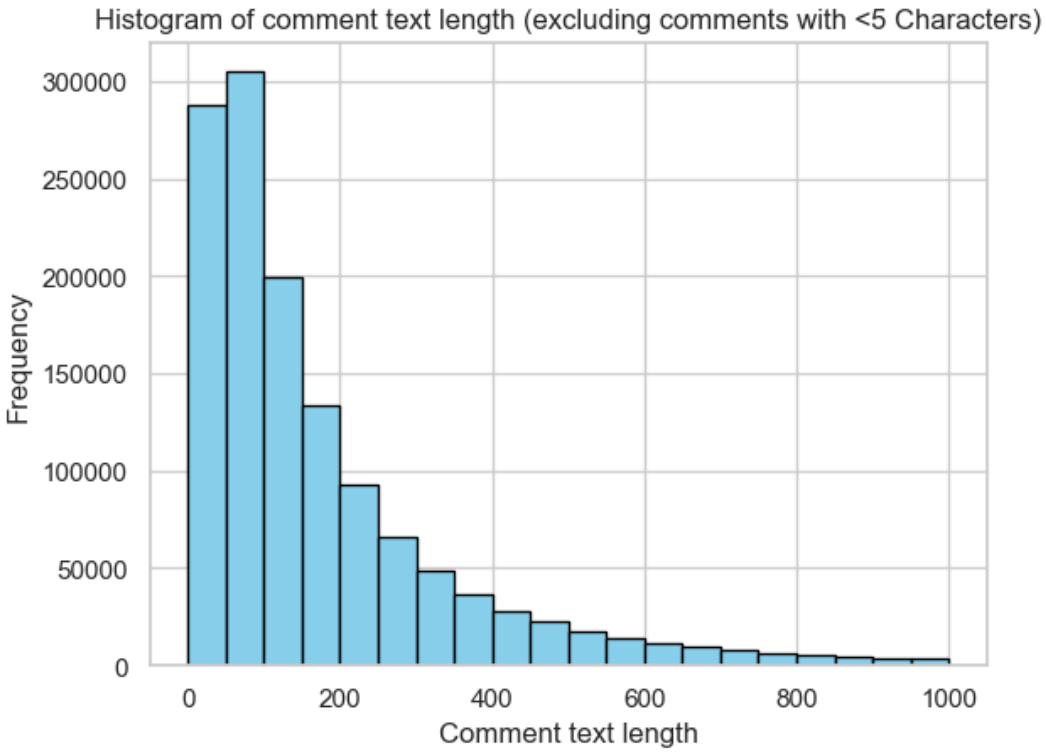
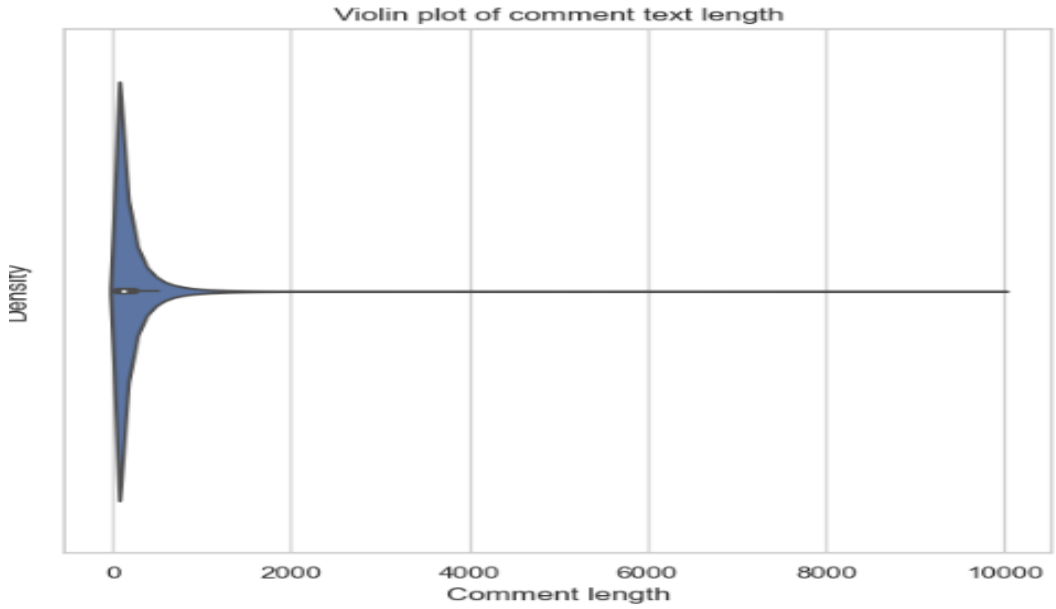


Figure 18: Violin plot of comment character count



Comments distribution by tag

Figure 19 shows a visual summary of comment volume by different tags while Figure 20 shows the share of comment volume by tag. There's an interesting difference

in distribution between posts and comments i.e. Business is the largest tag by comment volume compared to the highest being AI in posts. Business accounted for 17% of all comments volume while social media is a close second with 16% and AI came third. AI seems to dominate posts but not comments likely due to AI being a relatively new phenomenon and niche subject. The top 8 tags account for ~80% of the post volume.

Figure 19: Comment volume by tag

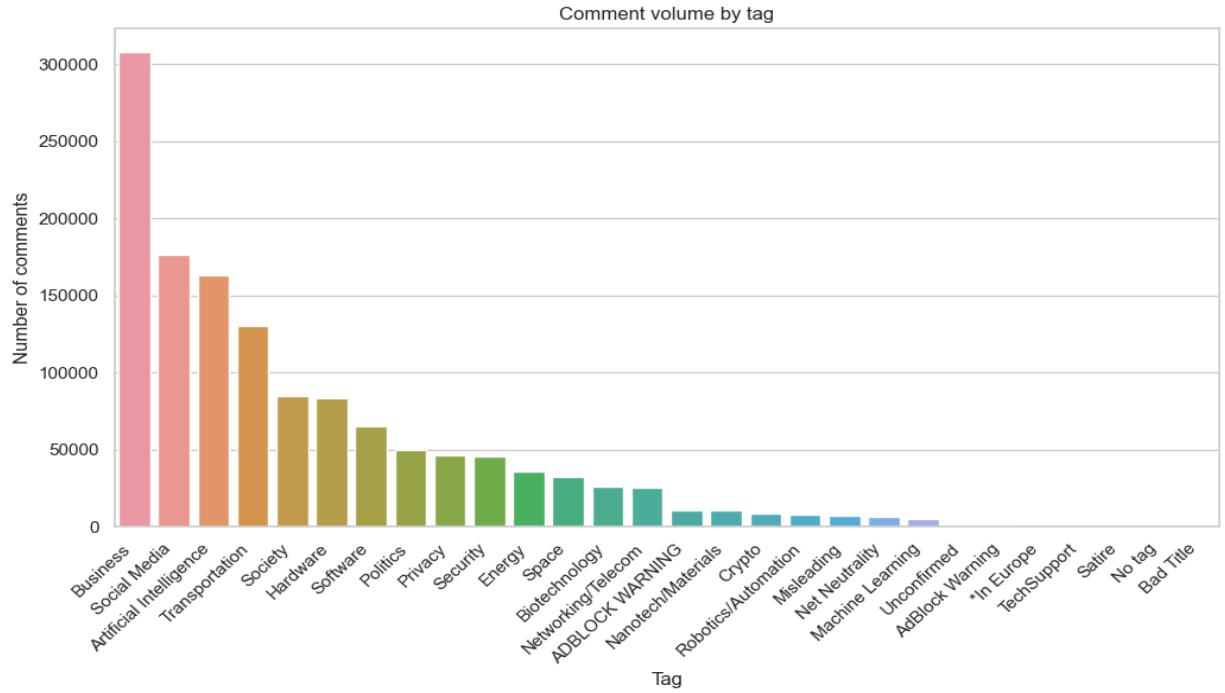
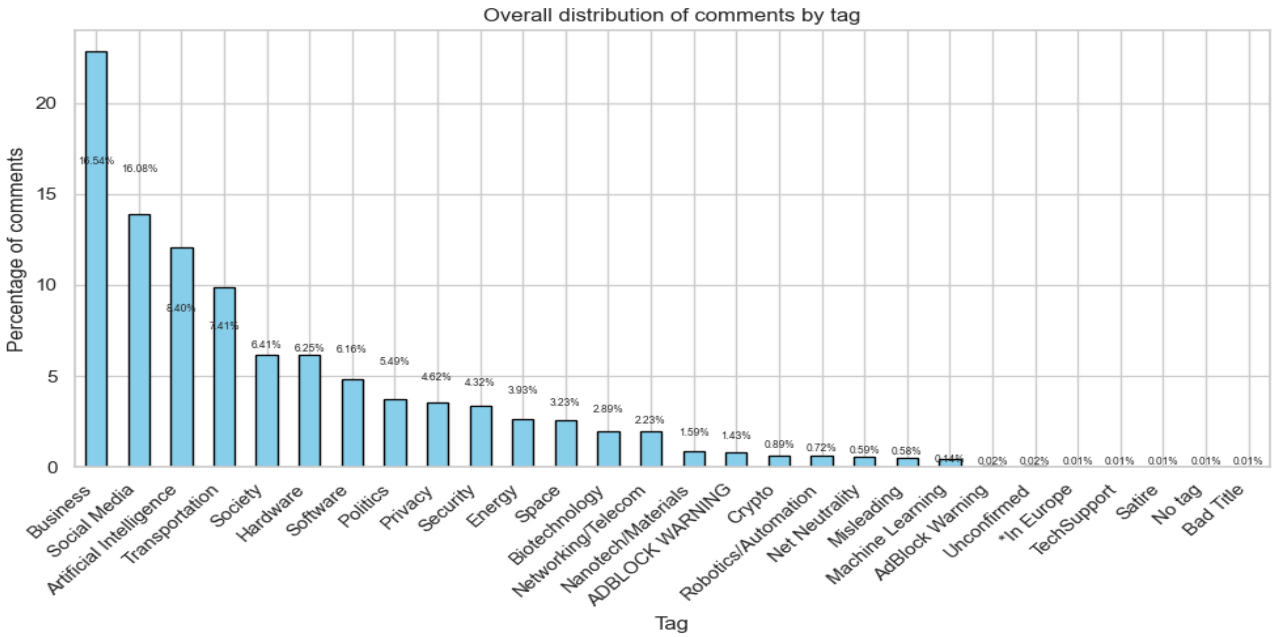


Figure 20: Comment volume share by tag



Comments per post by tag

Figure 21 is a strip plot that shows an interesting pattern in comments per post when studied by tag. Certain tags like politics, social media, business, hardware, society, AI, etc. are opinionated topics since they have a wider range of comments per post. This is also visualized in a paired box plot in Figure 22.

Figure 21: Strip plot of comments per post by tag

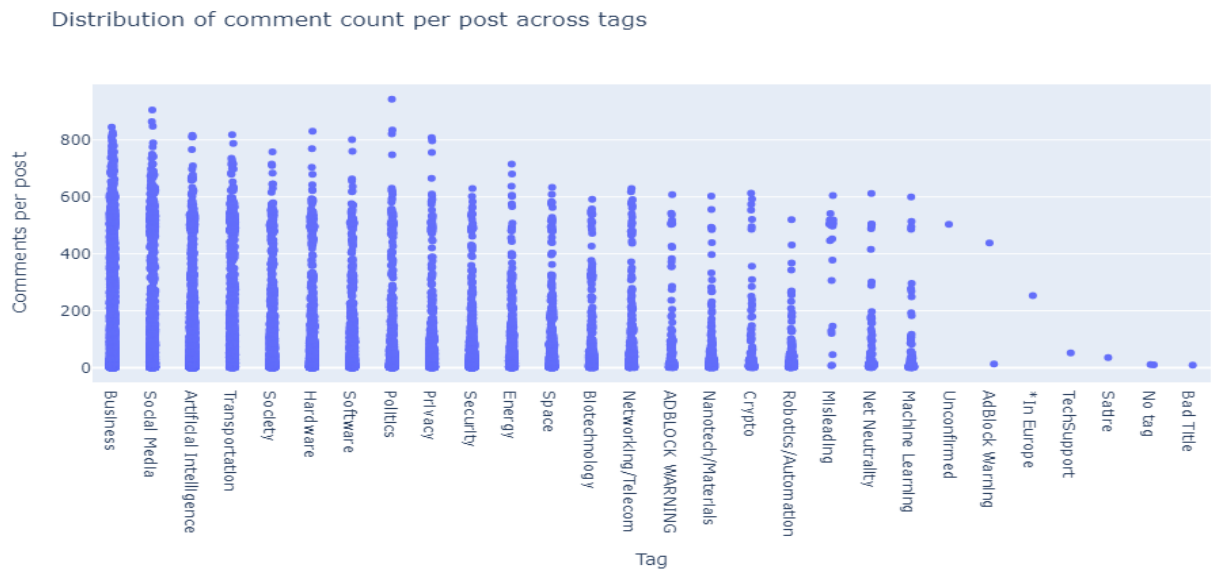
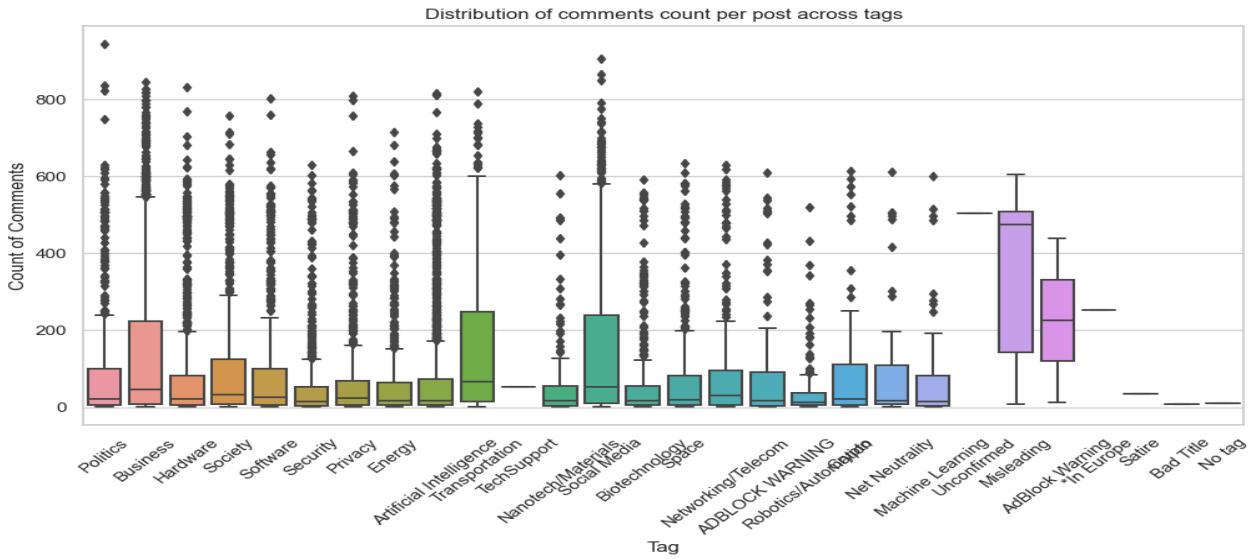


Figure 22: Paired box plot of comments per post by tag



Word frequency distribution of comment text

Now that we have understood the overall statistics and distribution of comments, we can study the text composition of the comment text. Figure 23 illustrates the unigram (single word distribution by frequency) of the words in the comment text. Figure 24 visualizes the bigram (two words) distribution from the comment text. The unigram and bigram distributions are noisy for comments as the data is quite large and diverse. They are mostly dominated by normal conversational language and terms. No insight can be derived from the ngram distributions.

Figure 23: Top 20 unigrams in comment text

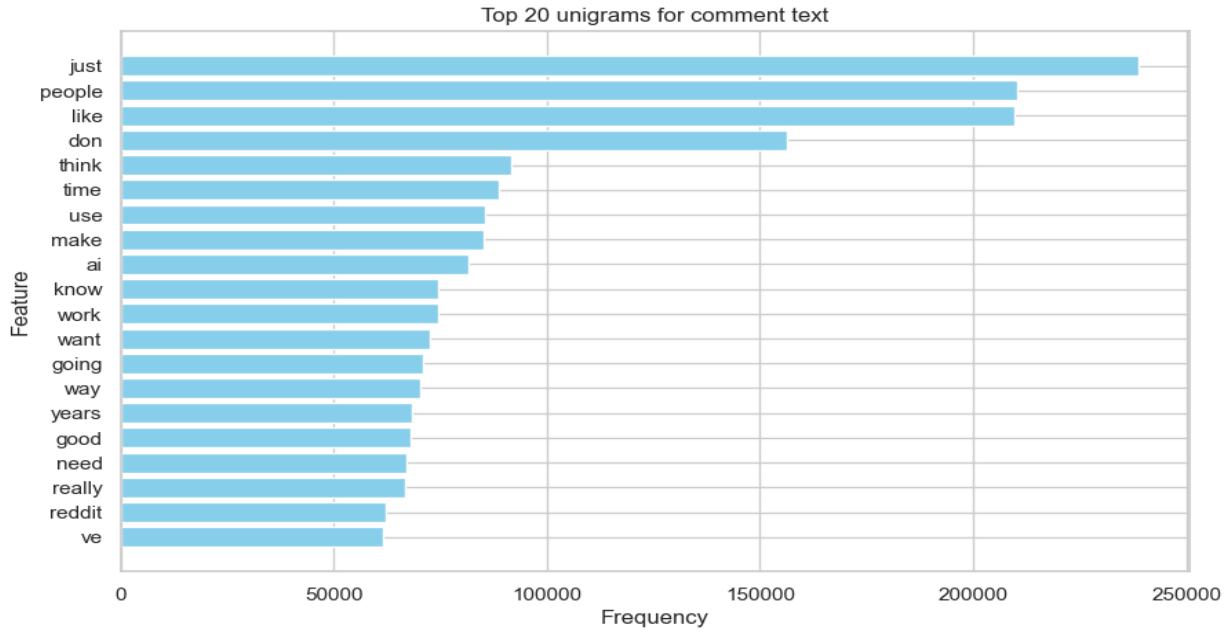
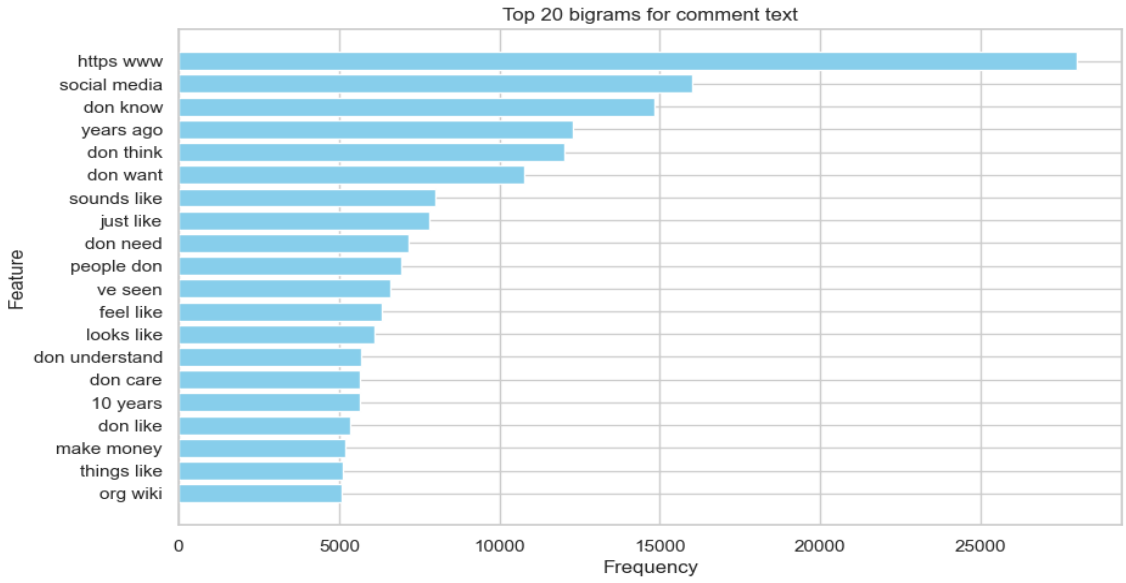


Figure 24: Top 20 bigrams in comment text



Sentiment distribution of comment text

Sentiment was extracted using the Python textblob library which provides a sentiment polarity value ranging from -1 to 1. Highly negative sentiment is indicated by -1, whereas highly positive is indicated as 1, and neutral is represented by 0. Figure 25 shows the distribution of sentiment of comment text across time and interestingly, most comments are positive. This is in contrast to post titles which were mostly neutral. A similar pattern can be observed in the sentiment distribution of comments by tag shown in Figure 26.

Figure 25: Distribution of sentiment of comment text across months

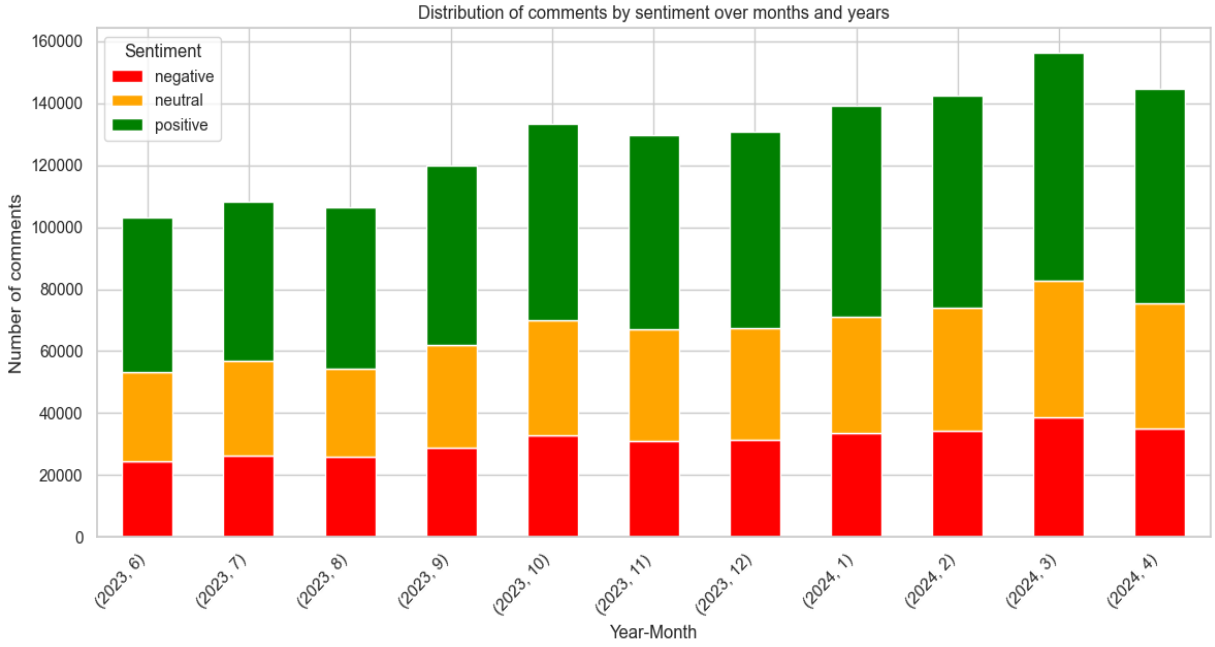
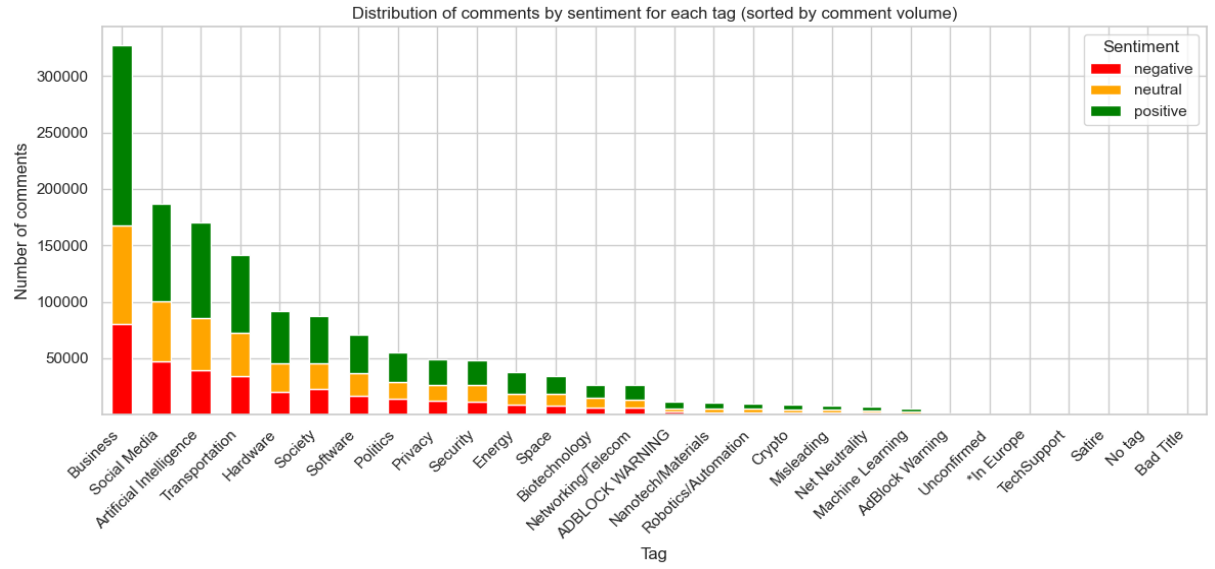


Figure 26: Sentiment distribution of comments by tag



Text cleaning performed on comment text

Operations like changing the comment text to lowercase, removal of English stopwords, removal of HTML tags and URLs, removal of emoji's and non-ASCII characters were performed using PySpark. Contractions (i.e. expansions of "i'm" to "i am") were also expanded. Figure 27 shows the pre and post-cleaning distributions of comment text length while Figure 28 shows the pre and post-cleaning distributions of comment word count. After performing text cleaning, we can clearly observe that the concentration of the lower text length and word count increases. This indicates that many characters and words are getting removed due to our cleaning steps. This is expected due to the inherent nature of the comments data on Reddit (multiple parent-child comments, single-word replies, and references) and this reduction is expected.

Figure 27: Comparison of comment text length, before and after cleaning operations

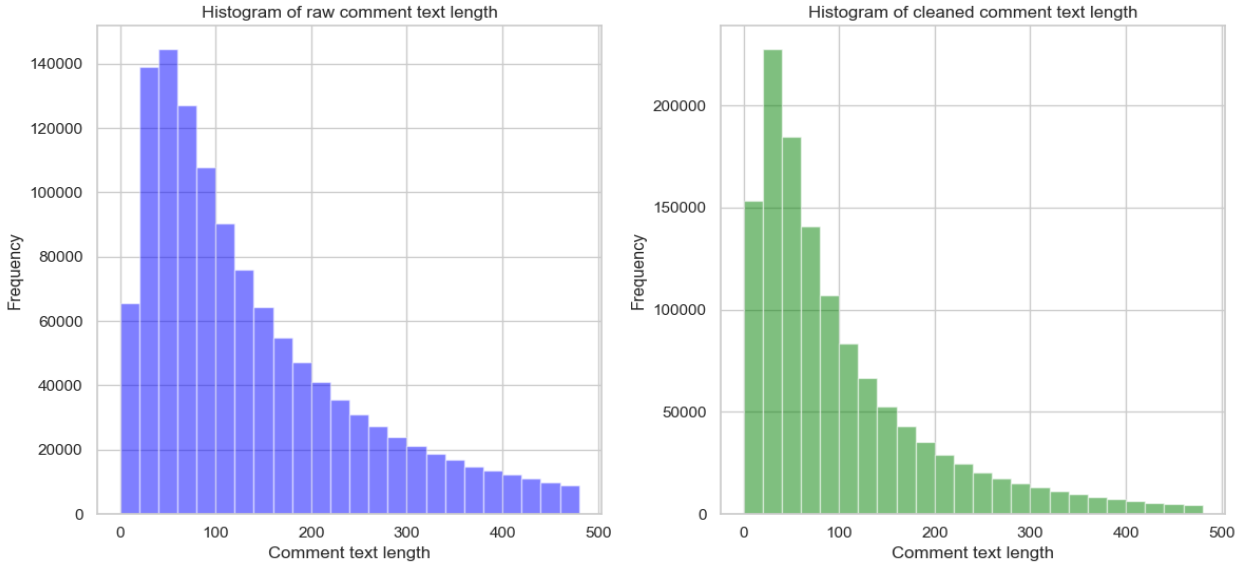
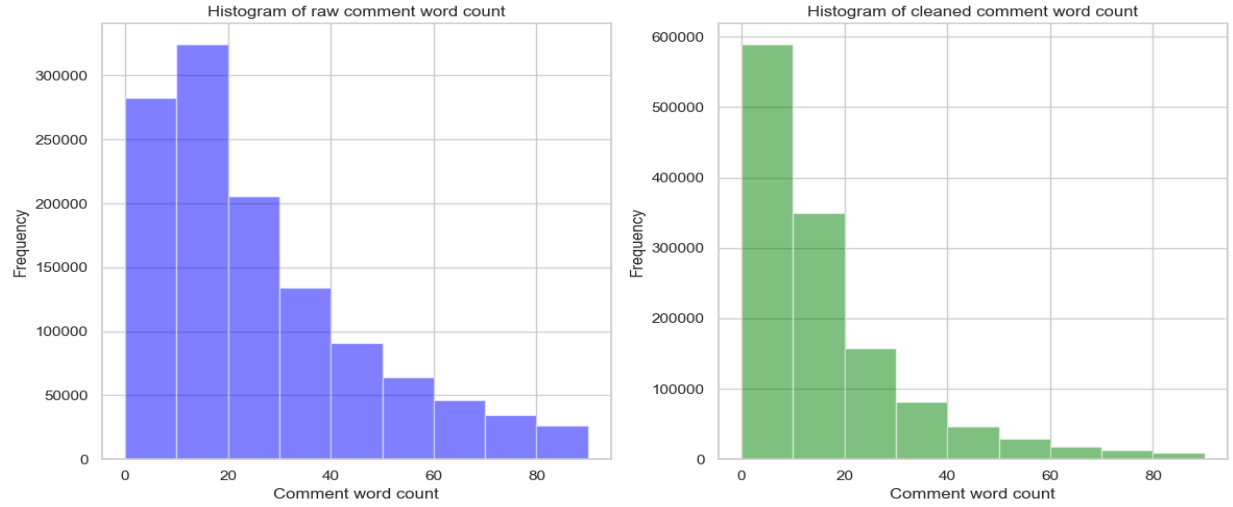


Figure 28: Comparison of comment word count, before and after cleaning operations



Unsupervised machine learning on post titles

The historical dataset doesn't contain any labeled data and hence we cannot use any supervised learning techniques. However, we have attempted two unsupervised machine-learning techniques - k-means clustering and topic modeling using Latent Dirichlet Allocation (LDA) on the cleaned post title. To reduce noise and improve the signal, we also focused on the largest tag i.e. Artificial Intelligence. We did not attempt any machine learning approaches on comments text due to a lot of noise in the data.

K-means clustering

k-means clustering is the simplest and most popular unsupervised machine learning model. Unsupervised models infer from the datasets using only the input data points without any labeled target variable. A cluster is a homogeneous collection of data points combined because of similarity across certain dimensions.
The k-means model identifies k number of centroids and then allocates every data point to the nearest cluster while keeping the centroid as small as possible. "k" refers to the number of centroids you can get from the dataset, and "means" refers to the averaging of the data while locating the cluster centroid.
For the Reddit post title text, focused on AI tags, we attempted kmeans by converting the cleaned title data into a TFIDF matrix using the top 1000 features. Figure 29 shows a snapshot of the code snippet used to create the TFIDF feature matrix.

Figure 29: Code snippet for creating the feature matrix for kmeans

kmeans for clustering

```
In [77]: # Initializing TfidfVectorizer to convert text data into TF-IDF features
         tfidf_vectorizer_ai = TfidfVectorizer(stop_words='english', max_features = 1000)

         # Fitting and transform the text data to TF-IDF features
         tfidf_matrix_post_text_ai = tfidf_vectorizer_ai.fit_transform(posts_with_comments_only_ai_tag['title_clean'])

In [78]: tfidf_matrix_post_text_ai

Out[78]: <2180x1000 sparse matrix of type '<class 'numpy.float64'>'
                with 12747 stored elements in Compressed Sparse Row format>
```

TFIDF stands for Term Frequency-Inverse Document Frequency and is used to measure the relevance of terms in a document and a collection of documents. TF stands for Term Frequency, which is the number of word occurrences. IDF stands for Inverse Document Frequency, it is the log of the ratio of the total number of documents to the number of documents in which a word appears. TFIDF score is the product of TF and IDF. This multiplication ensures that words that occur too frequently get a low score and rare or relevant terms get a higher score, ensuring that we score terms based on their relative importance and not just the frequency of occurrence. We used the TfidfVectorizer method from sklearn and restricted the features to the top 1000 (using

the max_features parameter) since a larger matrix is difficult to interpret and also has very high dimensionality.

Before clustering, we performed the Hopkins test to check for cluster tendency in the data. The Hopkin test statistic calculates the probability that a given input dataset is generated by a uniform distribution since uniform distributions inherently lack cluster tendency. In the Hopkins test, if the value is closer to 1, we can infer a higher tendency for clustering and a value close to 0 indicates that the data is closer to a uniform distribution and doesn't have clusters in the data. Figure 30 shows the code snippet for calculating Hopkin's statistic.

Figure 30: Hopkin's statistic code snippet

```
In [70]: # Function to define Hopkin's test of clustering tendency
         def hopkins(X, n):
             d = X.shape[1]
             # Creating random data points
             random_data = np.random.rand(n, d)
             # Computing nearest neighbors for both actual and random data
             knn_actual = NearestNeighbors(n_neighbors=1).fit(X)
             knn_random = NearestNeighbors(n_neighbors=1).fit(random_data)
             # Computing distances and sum them
             actual_distances = knn_actual.kneighbors(X)[0]
             random_distances = knn_random.kneighbors(random_data)[0]
             w = sum(actual_distances) / (sum(actual_distances) + sum(random_distances))
             return w
```

When we ran the Hopkins test for the TFIDF matrix of the overall cleaned post title, we got a value of 0.0117, indicating a low clustering tendency. Also, we ran the Hopkins test only for AI-tagged posts, where we achieved a value of 0.02017 which is still low but slightly better than the overall data value.

Even though Hopkin's test for both overall and AI-tagged data indicates no clustering tendency, we attempted to find the optimal value of k using the elbow curve method.

However, we could not identify clear clusters or "elbow points" from the elbow plot for either the overall data (shown in Figure 31) or for AI-tagged posts (shown in Figure 32).

Figure 31: Elbow curve to identify optimal k value for overall posts data

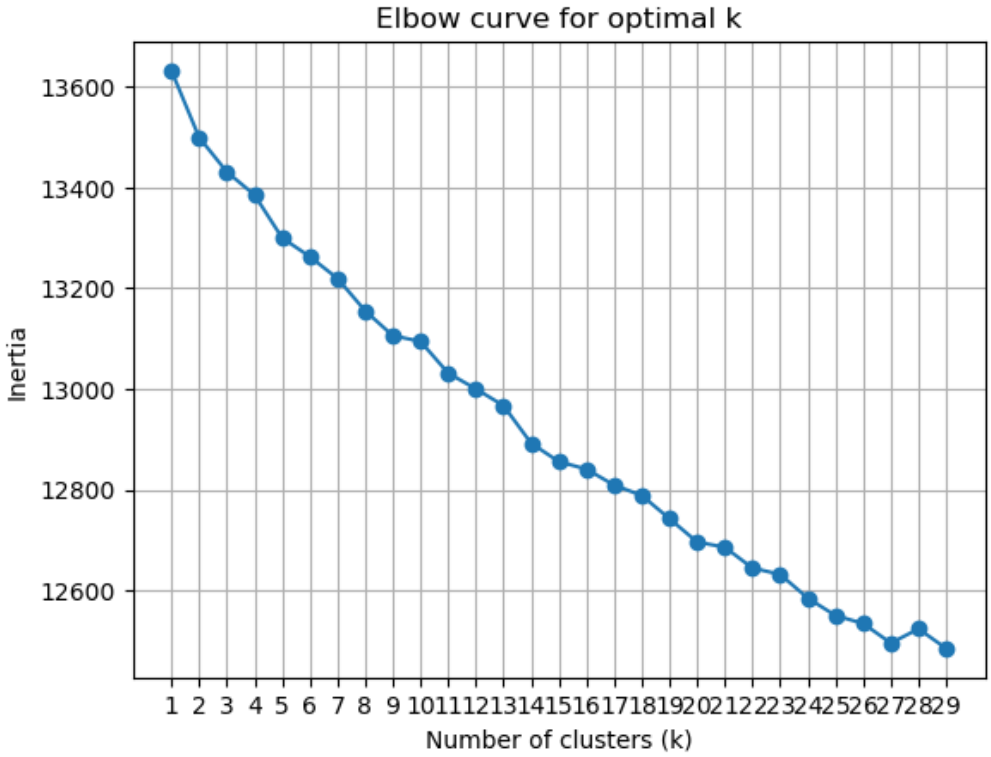
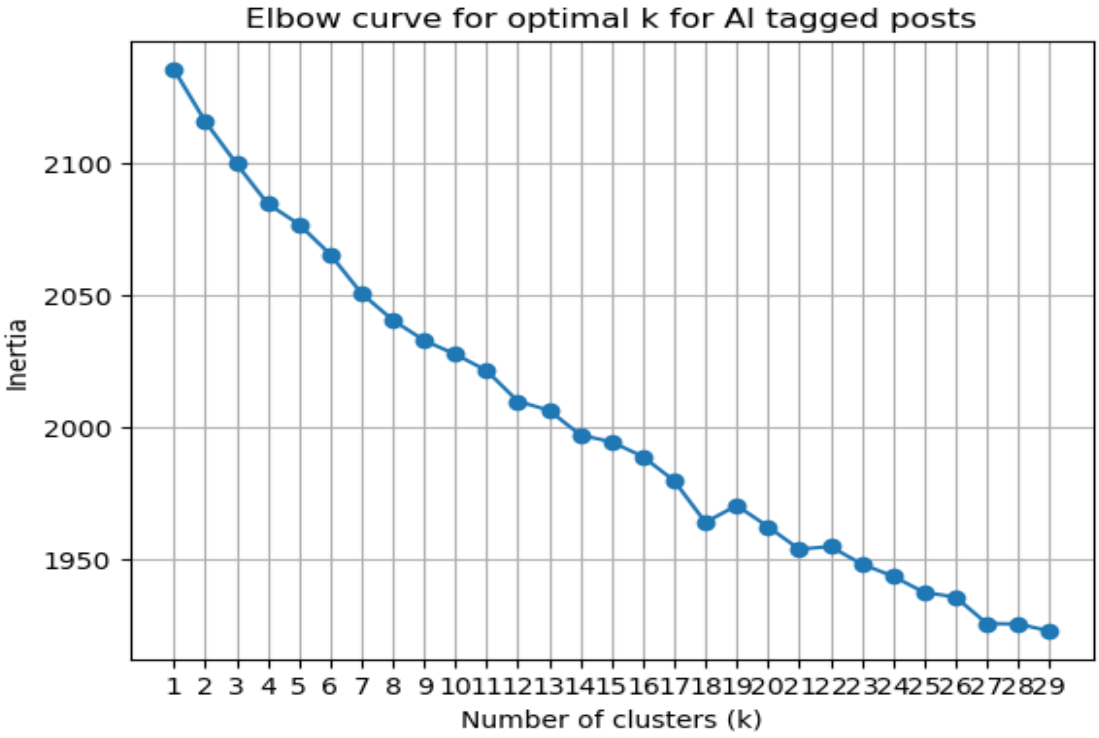


Figure 32: Elbow curve to identify optimal k value for AI tagged posts data



Latent Dirichlet Allocation (LDA)

Modeling training and results

LDA is a generative probabilistic model for topic modeling that can be used to summarize text when we are dealing with multiple documents of data. LDA assumes that documents are represented as random mixtures over latent topics, where each topic is composed of a distribution of words. The assumption is that the latent space discovered by the model is meaningful and useful although evaluating these assumptions is challenging due to the unsupervised training process. There is no definite list of topics to compare against. The evaluation like the optimal number of topics, in this case, is heuristic and subjective. This subjectivity makes the usage and interpretation of LDA challenging.
We performed LDA with five topics on posts tagged under AI The same feature dataset used for kmeans was reused in this case. The code snippet used for training the LDA model is shown in Figure 33.

Figure 33: LDA training on overall posts data and AI tagged posts data

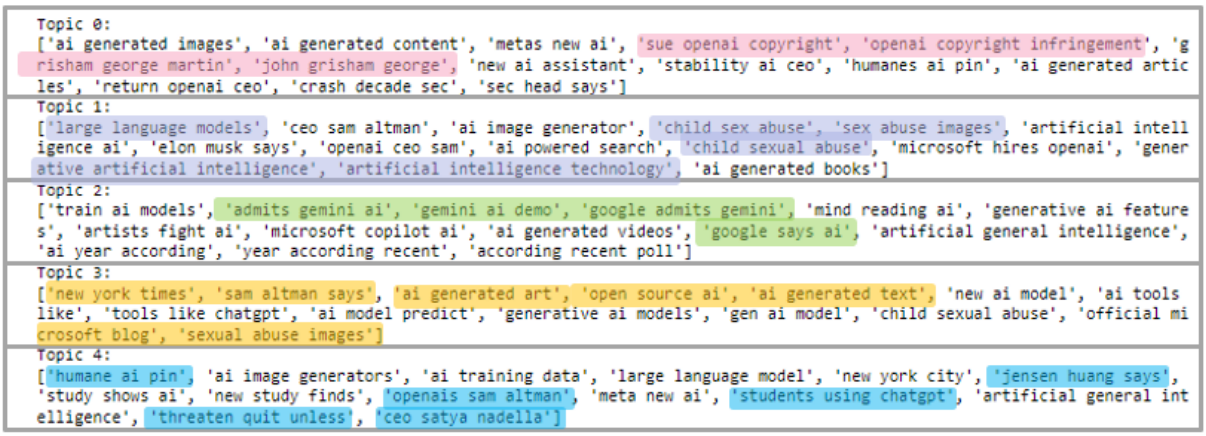
Topic modeling using Latent Dirichlet allocation

In [118]:
```
# Tokenization and vectorization
preprocessed_post_text = posts_with_comments_only['title_clean']
preprocessed_post_text_ai = posts_with_comments_only_ai_tag['title_clean']

vectorizer_for_lda = CountVectorizer(stop_words='english', max_features=1000)
vectorizer_for_lda_ai = CountVectorizer(ngram_range=(3, 3), stop_words='english', max_features=1000)

X_post_lda = vectorizer_for_lda.fit_transform(preprocessed_post_text)
X_post_lda_ai = vectorizer_for_lda_ai.fit_transform(preprocessed_post_text_ai)
```

In [127]:
```
# Training the LDA model
num_topics = 10
num_topics_ai = 5

lda_model_post = LatentDirichletAllocation(n_components=num_topics, random_state=123)
lda_model_post_ai = LatentDirichletAllocation(n_components=num_topics_ai, random_state=123)

lda_model_post.fit(X_post_lda)
lda_model_post_ai.fit(X_post_lda_ai)
# Extracting topic-term distribution matrix from the LDA model
topic_term_dists = lda_model_post.components_

topic_term_dists_ai = lda_model_post_ai.components_
```

We obtained explainable results from the LDA model of posts tagged as AI with five topics and trigrams. This was identified after studying multiple iterations of combinations of the number of topics and n-grams (unigrams, bigrams, and trigrams). From this, we were able to summarize five key themes from the output shown in Figure 34 and the interpretation of the topics follows.

Figure 34: LDA output for AI tagged posts with keywords highlighted to gauge topics



- **Topic 0**: News around training data for AI models (since there are references to authors and copyrights)
- **Topic 1**: Danger of AI, focused on deep fakes (since there are references to LLMs and sexual abuse)
- **Topic 2**: News about AI tools, dominated by Google's Gemini launch
- **Topic 3**: Mixed topic about announcements and AI tool announcements
- **Topic 4**: New AI product launches like Humane AI pin and controversies around AI tool usage by students

Visualizing LDA outputs

The outputs of LDA can be visualized for a more intuitive interpretation using 2 approaches – word clouds of topics (shown in Figure 35 and Figure 36) and word distribution (Figure 37).

Figure 35: LDA output for AI tagged posts – word cloud of Topic 0

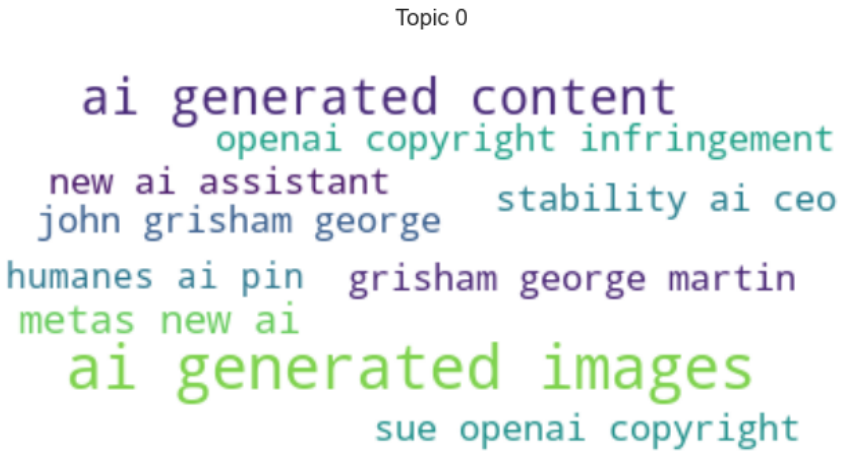
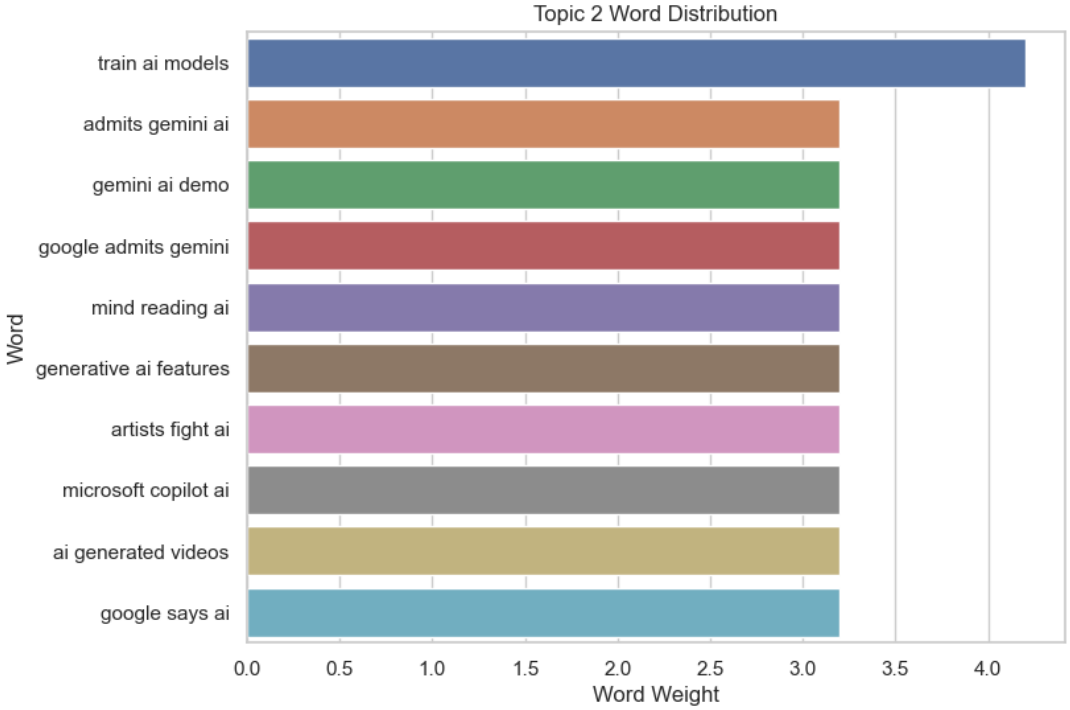


Figure 36: LDA output for AI tagged posts – word cloud of Topic 4

Figure 37: LDA output for AI tagged posts – word distribution of Topic 2



We can also study the probability of the documents' membership to each topic. Figure 38 shows the code snippet along with the topic distribution for the tagged AI post titles for three documents. All the implementation is available in file **Reddit social media listening - technology subreddit.ipynb.**

Figure 38: LDA output for AI tagged posts – topic probability for 3 documents

```
# Visualizing topic distributions for a few documents tagged in AI
num_documents_to_visualize_ai = 3
document_topics_ai = lda_model_post_ai.transform(X_post_lda_ai[:num_documents_to_visualize_ai])
for i, document in enumerate(preprocessed_post_text_ai[:num_documents_to_visualize_ai]):
    print(f"Document {i}: {document}")
    print("Topic Distribution:")
    for topic_idx, prob in enumerate(document_topics_ai[i]):
        print(f"Topic {topic_idx}: {prob:.4f}")
    print()
```

```
Document 0: sam altman says worries making chatgpt 'something really bad' given potential ai risks
Topic Distribution:
Topic 0: 0.1000
Topic 1: 0.1004
Topic 2: 0.1000
Topic 3: 0.5996
Topic 4: 0.1000

Document 1: hundreds protestants attended sermon nuremberg given chatgpt, told fear death
Topic Distribution:
Topic 0: 0.1000
Topic 1: 0.1000
Topic 2: 0.1000
Topic 3: 0.5999
Topic 4: 0.1000

Document 2: researchers discover chatgpt prefers repeating 25 jokes
Topic Distribution:
Topic 0: 0.0333
Topic 1: 0.0333
Topic 2: 0.8667
Topic 3: 0.0333
Topic 4: 0.0333
```

Takeaways from unsupervised learning

We got meaningful results using LDA but not K-means since the content on Reddit is interconnected and frequently referenced. Hence, identifying mutually exclusive groups (by clustering) may be difficult, but LDA can have the same terms repeated across topics and group topics relatively better. The challenge with using unsupervised ML is that the end output is often not clear, and interpretation can be subjective, but they're powerful methods to identify hidden patterns in the data nonetheless.

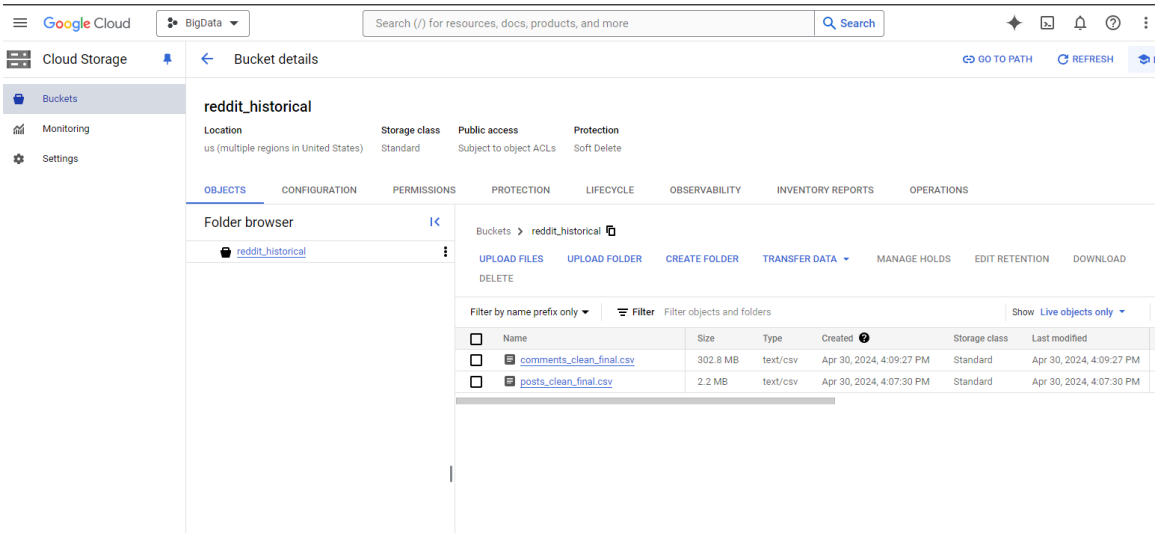
Streamlit social media listening app for historical data

Streamlit overview and ingestion framework

Streamlit is a free and open-source framework to build and share dashboards as web apps. Creating effective user interfaces for data-heavy apps can be challenging. Streamlit, an open-source Python library, allows developers to create attractive UIs quickly and effortlessly. We've built an app in Streamlit that can help in tracking and analyzing multiple metrics and trends as part of social media listening.
Since we wanted the app to be responsive with limited processing and transformation overhead being done in the app layer, the data ingestion or input data layer of the app has been designed to directly establish a connection to a secure GCS bucket and read

in posts and comments CSV files. These files were processed using an elaborate PySpark pipeline that concluded with the processed files for posts and comments directly written to the mentioned secure GCS bucket as shown in Figure 39.

Figure 39: GCS bucket after the data processing and transformation



App functionality overview

The app loads the posts or comments historical data dynamically from the GCS bucket when “Load historical data” is clicked based on a chosen view of either “Posts” or “Comments” present in the sidebar. Users can then proceed to specify input keyword(s) in the “Enter keywords” input field to filter the posts or comments. This is visible in the app snapshot shown in Figure 38 wherein the keyword “adobe” is input. Next, the user can click on “Generate charts” button to generate the metrics and charts. The app is meant to provide the user with a comprehensive summary of metrics, trends, and other charts that can help understand how the input keyword search is perceived on the r/technology subreddit. The metrics and charts change dynamically based on the keywords provided by the user. All the implementation is available in **streamlit_reddit_historical.py** file and all the libraries are available in **requirement.txt**. A secure toml file is created and embedded directly in the Streamlit app – this is a secret file meant to establish a secure API connection for read and write access to the GCS bucket and has not been shared in the submission due to security reasons.

The live app can be accessed through the URL: https://reddit-historical-listening.streamlit.app/

Dashboard navigation

As explained in the overview, clicking on “Generate charts” will generate 5 rows of metrics and charts filtered for the keyword specified. The modular row and column approach is designed to be straightforward from a user experience standpoint.

App rows 1 & 2

Row 1 of the app shown in Figure 40 shows the key aggregate numeric metrics such as:

- “Count of posts”: count of the number of posts the keyword is mentioned in
- “Influence”: This is the share of posts containing the searched keyword out of all posts
- “Share of voice”: This is the percentage of the searched keyword among all the post title words
- “Total aggregate score”: the total aggregate difference between upvotes and downvotes

Row 2 of the app shown in Figure 40 stretches the interactive heatmap of post volume across time for the searched keyword

App rows 3 & 4

Row 3 of the app shown in Figure 41 displays 2 charts side by side dividing the space equally:

- “Count of posts by tag”: this shows the distribution of the posts with the selected keyword across tags using a bar chart
- “Count of posts over time”: this shows the volume of the posts with the selected keyword across time as an area chart

Row 4: This row displays the overall distribution of the sentiments and wordcloud of the words associated with the searched keyword across the three sentiment types as seen in the lower half of Figure 41.

App row 5

The last row shows the top 10 unigrams, bigrams, and trigrams for posts which gives the contextual usage of the keyword and this is shown in Figure 42. N-gram charts are computationally intensive and hence, to optimize the user experience and reduce latency, the 3 charts are created using a random sample of 10,000 rows selected from the results of the posts filtered for the keyword. A callout has been added at the bottom notifying the user about this.

Figure 40: Top layer of app showing load button, keyword input and rows 1 and 2

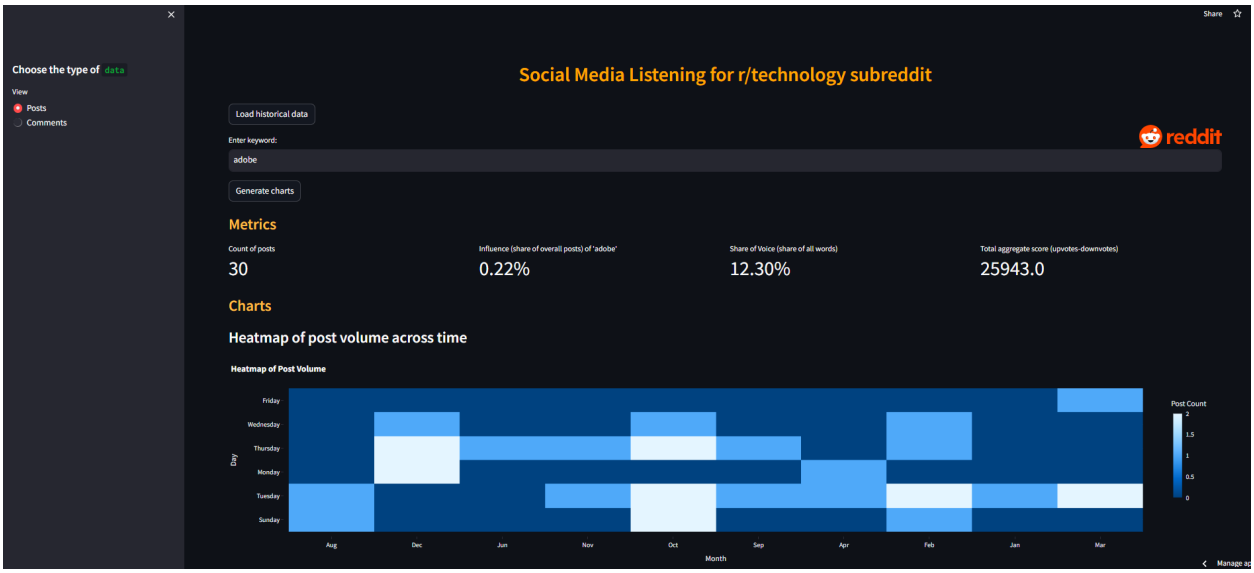


Figure 41: Middle layer of app showing rows 3 and 4

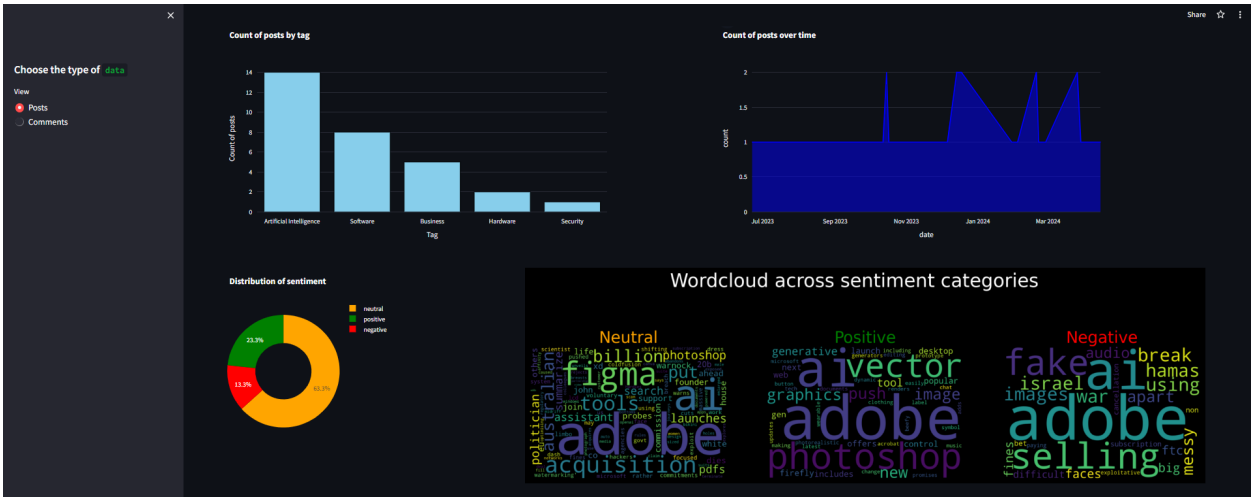
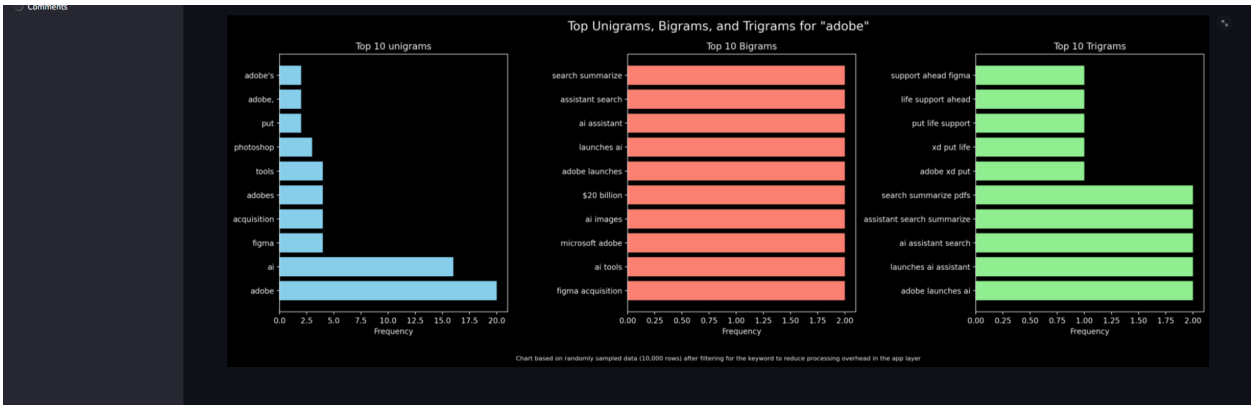


Figure 42: Last row of app showing row 5 with ngram charts



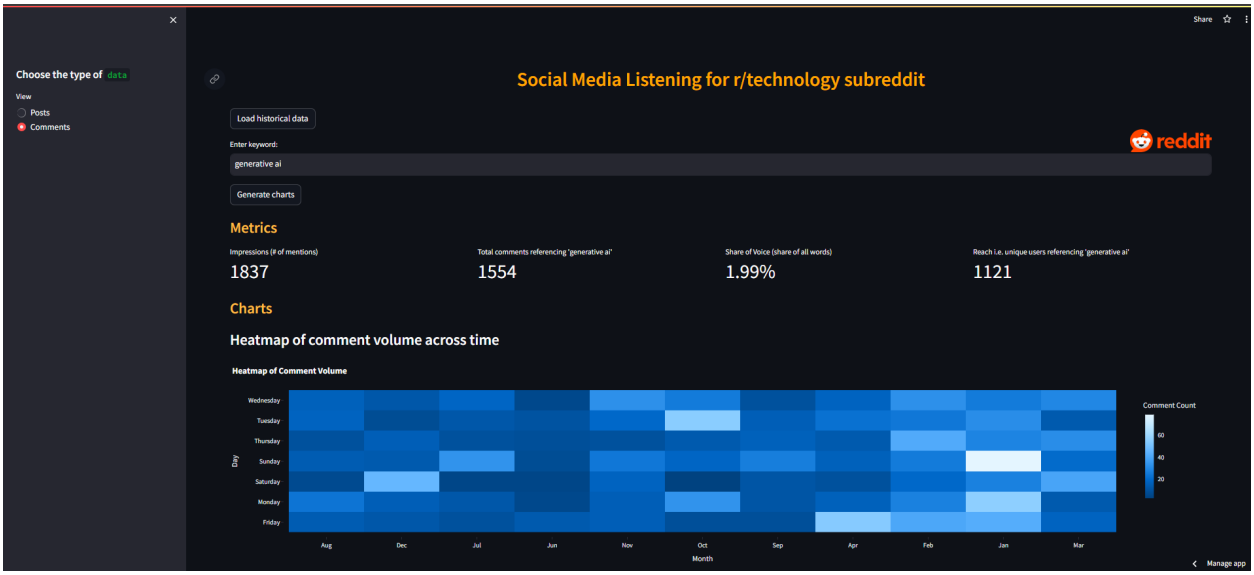
Comments View

The layout of the comments view is almost identical except for Row 1 which has metrics that are relevant for comments. The metrics in row 1 are described below:

- "Impressions" i.e. number of times the searched keywords are mentioned in the comments and "Total comments referencing <keyword>". For instance, if there are two comments and the keyword is mentioned twice in each comment the **impressions** will be 4 and the **total comments referenced** will be two
- Share of voice is the percentage of all the keywords in comments while Reach is the unique number of users referencing the searched keyword

A snapshot of row 1 for the "Comments" view is shown in Figure 43.

Figure 43: Top layer with rows 1 and 2 for Comments view



Rows 2, 3, 4 and 5 remain the same as Posts view and are shown in Figure 44 and 45.

Figure 44: Middle layer with rows 3 and 4 for Comments view

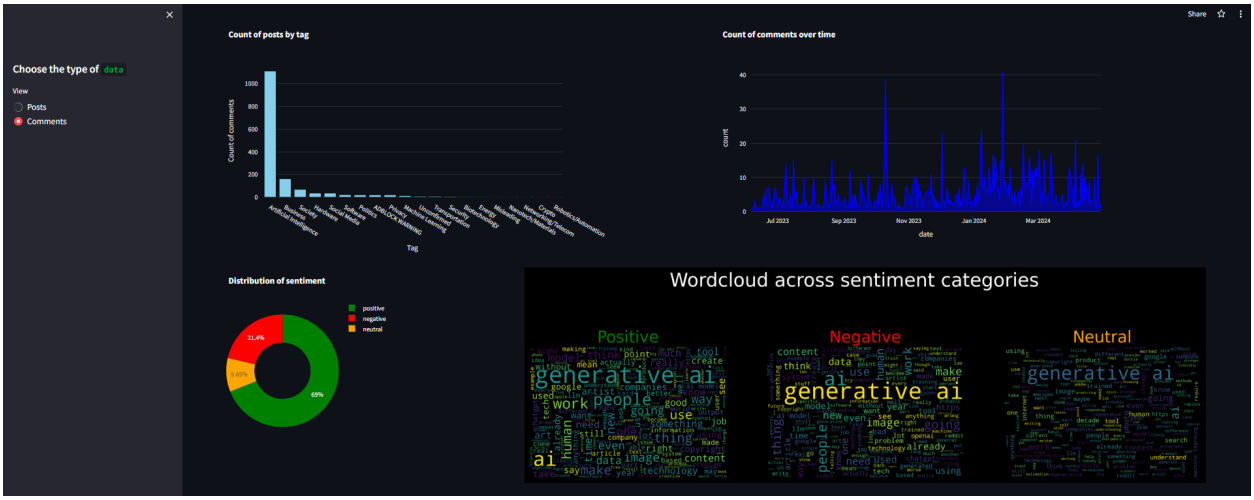
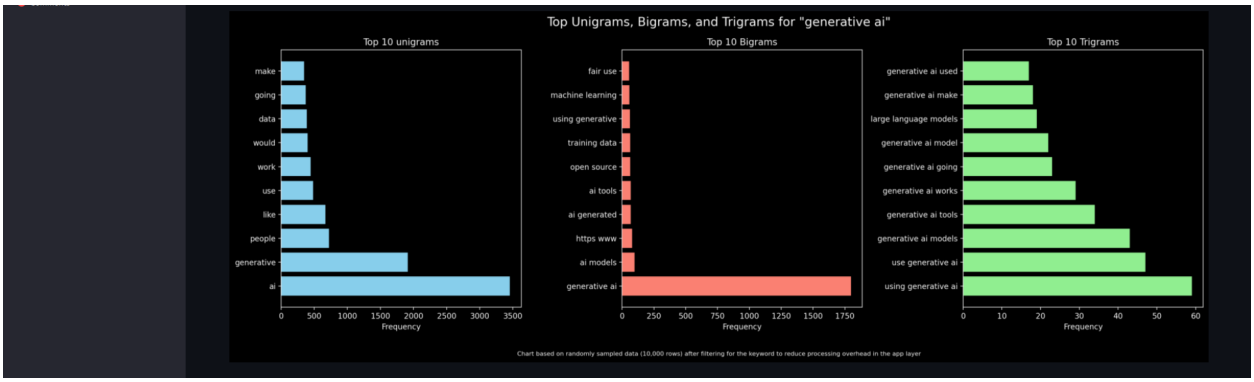


Figure 45: Last row of app showing row 5 with ngram charts for Comments view



Possible enhancements for the historical data Streamlit app

- We can integrate multiple social media channel data streams like X (formerly Twitter), Instagram, Google Trends, etc.

- Robust data engineering backend using big data frameworks to manage and process large streams of data at scale
- We can perform ML on historical and real-time data to predict trends and patterns of user behavior
- We can track the impact of social media marketing campaigns
- We can identify and reach out to power users (heavy users) and influencers to increase engagement and improve brand positioning of products, particularly technology-focused
- Brands can use this tool to understand the pulse of users and engage with them more effectively
- Additionally, we can use the latest open-source LLMs like Langchain or Llama to include a generative AI component that can describe and provide nuanced context about the keyword and can also possibly provide insights based on looking at the metrics and charts

Real Time: Social Media Listening on Reddit using Kafka and Streamlit

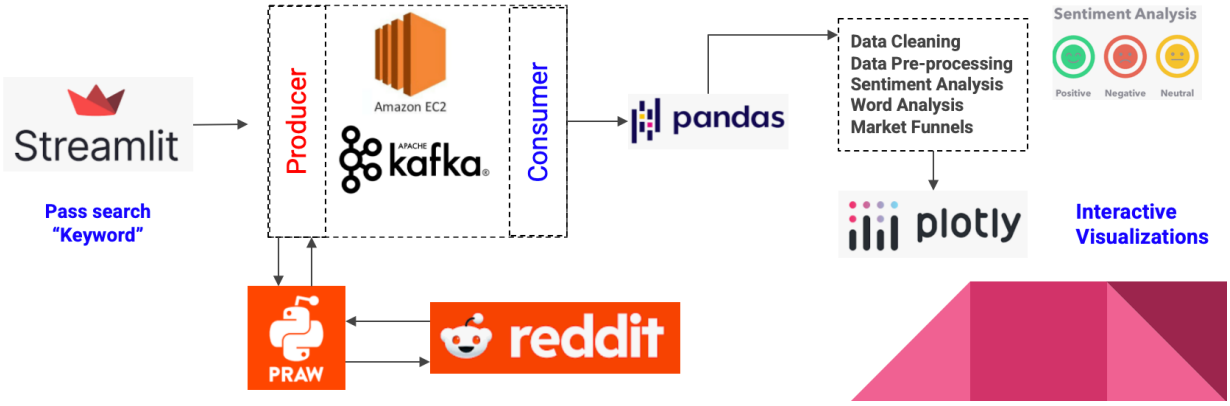
Based on the EDA performed above on historical data, we have identified how trends and sentiments vary across different topics and how key discussion points impact buyer decisions. We utilized this learning from extensive analysis of historical data. Using the mentioned insights, we created a real time app to focus on those key aspects to be able to quickly visualize using real time data.

Solution Expansion

We utilized our understanding of Kafka and API usage to handle real-time data fetch from Reddit. Kafka is an open-source distributed event streaming platform used for building real-time data pipelines and streaming applications. It is designed to handle large volumes of data with high throughput and low latency. Kafka is scalable, fault-tolerant, and durable, making it an ideal choice for building reliable and robust data streaming applications. We are using Kafka to fetch real-time data from reddit using PRAW library. We have deployed Kafka on Amazon AWS EC2 instance.

Solution Implementation

We will discuss the steps sequentially to create the real-time usage app.



Step 1: We created an EC2 instance on Amazon AWS. Using AWS Console, we launched a new instance with the "Instance Type" as t3.micro which is a LINUX based instance. We kept the IP address dynamic to be in line with AWS tier requirements. Also, we generated a private key to authenticate the connection to the instance. We also, added an entry for Port 9092 so that it can accept tcp/ip commands from all sources.

Step 2: The instance was started on the client machine using the following command wherein the "reddit-kafka.pem" is the private key for the instance.

ssh -i "reddit-kafka.pem" ec2-user@ec2-18-234-36-200.compute-1.amazonaws.com

Step 3: Once logged into the instance, we installed Kafka on EC2 instance using the following command.

```
A newer release of "Amazon Linux" is available.
  Version 2023.4.20240429:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sat May  4 19:02:45 2024 from 98.210.10.93


[ec2-user@ip-172-31-26-5 ~]$ ls
kafka_2.12-3.7.0  kafka_2.12-3.7.0.tgz
```

After installation, we updated the EC2 instance IP address in the server.properties file so that when kafka starts it uses this static IP and listens on port 9092.

```
 GNU nano 5.8                                        config/server.properties
############################ Server Basics #############################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0

############################# Socket Server Settings #############################

# The address the socket server listens on. If not configured, the host name will be equal to the value of
# java.net.InetAddress.getCanonicalHostName(), with PLAINTEXT listener name, and port 9092.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092

# Listener name, hostname and port the broker will advertise to clients.
# If not set, it uses the value for "listeners".
advertised.listeners=PLAINTEXT://18.234.36.200:9092

# Maps listener names to security protocols, the default is for them to be the same. See the config documentation for more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL

# The number of threads that the server uses for receiving requests from the network and sending responses to the network
```

We then created an entry in the bashrc file to auto start Kafka whenever user logs into the EC2 instance. This script automatically starts both zookeeper and kafka servers.

```
 GNU nano 5.8                                        /home/ec2-user/.bashrc

# User specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

cd kafka_2.12-3.7.0

echo 'starting zookeeper..'
bin/zookeeper-server-start.sh config/zookeeper.properties &

sleep 5

echo 'exporting heap opts'
export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M"
sleep 2
echo "Starting Kafka..."

bin/kafka-server-start.sh config/server.properties &
# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=
```

Step 3: We then tested the connectivity to the Kafka servers on EC2 from the client machine using Python. We had to install kafka-python library to get this to work. We were able to successfully launch and test both producer and consumer. We have added logic in our code to launch both producer and consumer in parallel threads and each thread waits for each other before terminating.

```python
# Call Producer and Consumer
def start_data_fetch(keywords_input):
    producer_done_event = threading.Event()
    data_list = []
    producer_thread = threading.Thread(target=start_producer, args=(keywords_input,producer_done_event))
    consumer_thread = threading.Thread(target=start_consumer, args=(data_list, producer_done_event))
    producer_thread.start()
    consumer_thread.start()
    producer_thread.join()
    consumer_thread.join()
    if not data_list:
        return pd.DataFrame()
    else:
        df = pd.json_normalize(data_list,'comments',
                               ['post_id', 'title', 'url', 'score', 'upvotes', 'downvotes',
                               'num_comments', 'text', 'author', 'author_post_karma', 'tag'],
                               record_prefix='comment_')
        return df
```

- **Kafka Producer**: We created simple function to launch the Producer by using the bootstrap server as the same server IP we configured above.

```python

    # Start Kafka Producer
    def kafka_producer(subreddit_name, keywords):
        producer = KafkaProducer(bootstrap_servers=['18.234.36.200:9092'])
        for data in fetch_data_from_reddit(subreddit_name, keywords):
            serialized_data = json.dumps(data).encode('utf-8')
            producer.send('technot', serialized_data)
        producer.flush()
        producer.close()

    def start_producer(keywords_input, producer_done_event):
        subreddit_name = 'technology'
        keywords = [keyword.strip() for keyword in keywords_input.split(",")]
        kafka_producer(subreddit_name, keywords)
        producer_done_event.set()

```

- **Kafka Consumer** – We created simple function to launch the Producer by using the bootstrap server as the same server IP we configured above.

```python
    # Start Kafka Consumer
    def start_consumer(data_list, producer_done_event):
        consumer = KafkaConsumer('technot', bootstrap_servers=['18.234.36.200:9092'])
        try:
            while not producer_done_event.is_set():
                batch = consumer.poll(timeout_ms=1000)
                for tp, messages in batch.items():
                    for message in messages:
                        data = loads(message.value.decode('utf-8'))
                        data_list.append(data)
                    if len(batch) == 0:
                        print("No new messages received.")
        except KeyboardInterrupt:
            pass  # Catch KeyboardInterrupt to gracefully stop the consumer
        finally:
            consumer.close()
```

21.6 KB

Step 4: Kafka fetches real-time data from Reddit using the PRAW (Python Reddit API Wrapper) library. We have limited the extraction of data to '**r/technology**" and the search will focus only on new comments for the past month. In sequence, the code will

launch producer and consumer in parallel instances and then start posting messages to the producer. We then consume the messages using consumer and create a data list.

```
reddit = praw.Reddit(
    client_id=[redacted]
    client_secret=[redacted]
    password=[redacted]
    user_agent=[redacted]
    username=[redacted]
)

def fetch_data_from_reddit(subreddit_name, keywords):
    subreddit = reddit.subreddit(subreddit_name)
    for submission in subreddit.search(' '.join(keywords), time_filter='month', sort='new', limit=50):
        if all(keyword.lower() in submission.title.lower() for keyword in keywords):
            post_id = submission.id
            title = submission.title
            url = submission.url
            score = submission.score
            upvotes = submission.ups
            downvotes = submission.downs
            num_comments = submission.num_comments
            text = submission.selftext
```

Step 5: Data Pre-processing – After we have the real-time raw comments data in JSON format, we convert them into a more readable format by converting it into a simple dataframe. This is done by normalizing the raw JSON using standard pandas functionality.

The Raw dataset has the following columns:

Column Name	Description
COMMENT_ID	ID of the Reddit user who posted a comment on the searched keyword
COMMENT_AUTHOR	Name of the Reddit user
COMMENT_DATETIME	Date and time when the comment was posted
COMMENT_TEXT	Text data for the actual comment
POST_ID	ID of the post on the subreddit under which the comment was posted

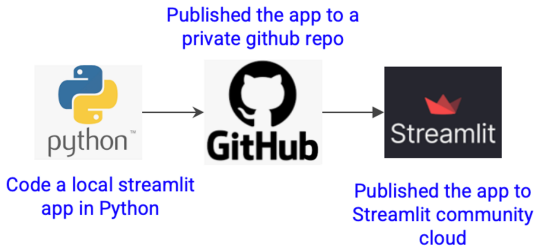
After Pre-processing and cleaning, we utilized the Vader Sentiment library to draw the sentiment on the data. After the sentiment analysis and pre-processing, few additional column were generated as shown below:

Column Name	Description

Sentiment	ID of the Reddit user who posted a comment on the searched keyword
Sentiment Type	Using Polarity scores, label the sentiment, positive if > 0.05, negative if less than -0.05, else neutral
Datetime	Convert object datetime into standard datetime format
Aspects	Generate data aspect labels using pre-defined aspect terms
Hour	Hour when the review was posted
Weekday	Day when the review was posted

Building the Streamlit App

Created a Streamlit App to run our entire pipeline end to end by invoking the producer and consumer on EC2 instance and fetching real time data from Reddit. **Once the data is retrieved, pre-processing and visualization is handled at real time as well.** All this happens on a click of a button on the app. The user passes a keyword in the designated input box and clicks a button. **As consumer and producer need to be running parallely, we have used the threading library to create parallel threads and then waiting until both finish.** The app is published to Streamlit using a github repo and it internally **invokes Producer and Consumer by using a pre-configured bootstrap server on EC2 compute instance on AWS**.



The app was published on the Streamlit cloud and it can be accessed using the URL below.

URL: https://reddit-realtime-listening.streamlit.app/

The code for the app was written and testing locally using python. We created multiple functions for visualizing the data based on different aspects which we will

explain below. All the functions are handled in the file **functions.py** and the main code for app is **RedditDashboard.py**. Below is the screenshot for the front page of the app.

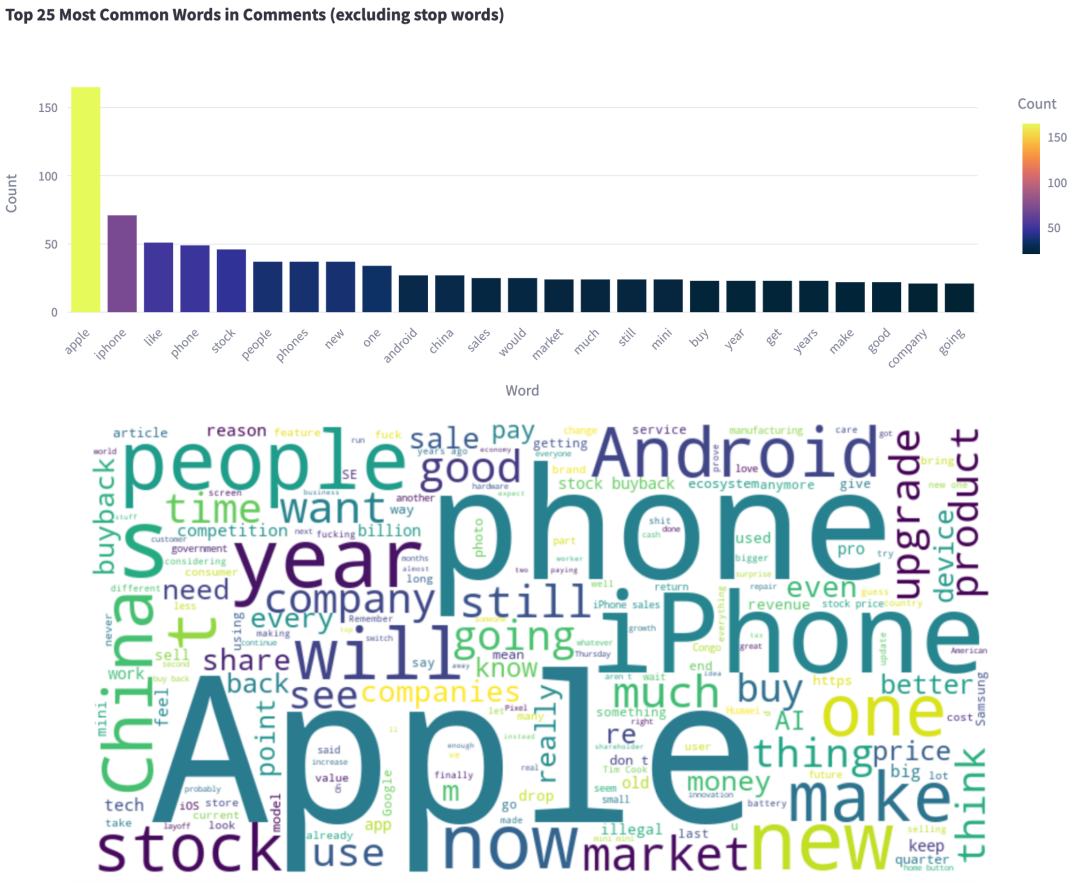
The page's sizing and appearance is handled by creating a custom CSS file which provides the sizing configuration. On the left-hand side, we have text input field for user to enter a keyword of his choice. When the search button is clicked by user, our entire pipeline will start its job to produce the results on screen.

When a search is completed, the above page will be displayed to the user giving the information. On the left sidebar, there is information about certain metrics that define how the keyword is performing.

Reach: The number of unique comment authors.
Engagement: The sum of upvotes and number of comments.
Share of Voice: The percentage of posts that mention <Keyword>.

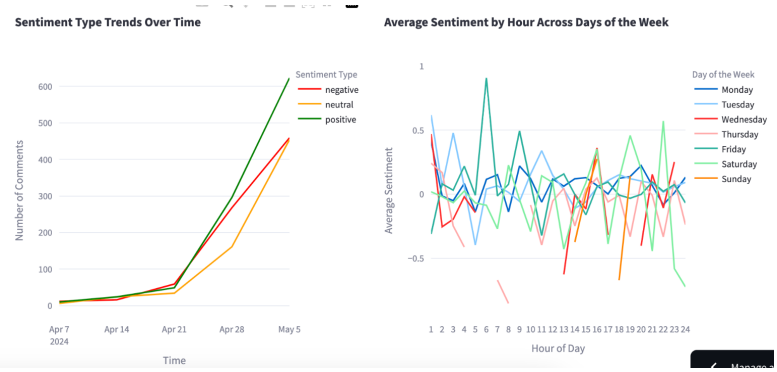
On the main page, there are sub-tabs for different analytic visualization that we are performed on the processed data.

- **Word Analysis** - Analyzing the comments text and visualizing the top 25 most used words along with a word cloud of most frequently occurring words.

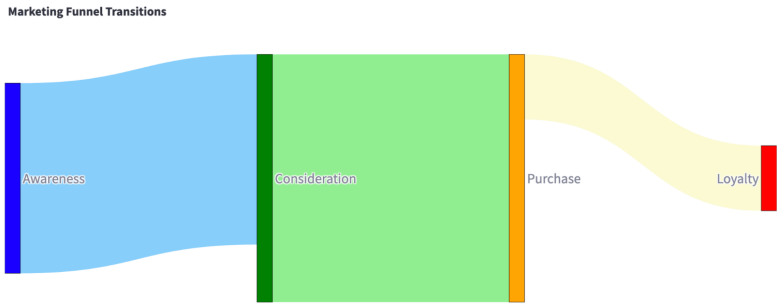


- **Sentiment Analysis** - Visualizing the sentiment of the posted comments by aspects as well as overall sentiments.

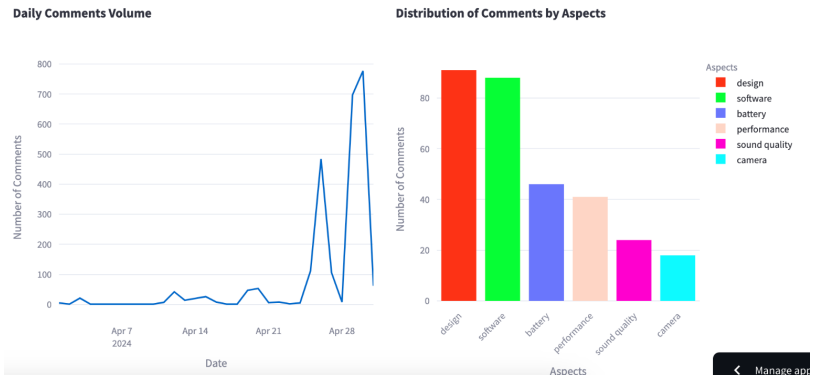
- **Sentiment Trends** - Visualizing Sentiment changes by over multiple days to understand how user perceptions are changing.



- **Market Funnel** - Visualization of awareness, purchase considerations and actual purchases based on the comments data.



- **Comment Analysis** - Daily volume of comments along with key topics of discussion.



Future Enhancements

We plan to make this application usable for a wide variety of use cases. We want to decouple the domain limitation which is currently focusing only non the **r/technology**.

Currently, the app focuses only on one month worth of data for the search, we want to expand this criterion to at least 3 months to perform a more extensive search. However, we need to find a way to make it efficient as the search is usually time intensive and it is not a good user experience if the page takes more than 30-40 seconds to load.

The end goal of this app is to provide a holistic unbiased audience opinion on any topic. This can help users understand how a product/service is being perceived by wider audience. All this in a matter of seconds/minutes rather than searching through multiple reddit posts/comments manually.

Appendix

Python Files

- **Reddit social media listening - technology subreddit.ipynb**: This files contains the python code and unsupervised machine learning code.
- **PySpark_Reddit_DataProcessing.ipynb**: This file contains the pyspark code for data processing and data transformation.

Streamlit Files-Historical

- **streamlit_reddit_historical.py**: This file contains the streamlit code for historical app
- **requirement.txt**: File contains all the libraries that are required for streamlit app

Streamlit Files – Real Time Streamlit App

Github URL: https://github.com/eshita1991/Data228-Real-Time-Social-Media-Listening

- **DashStyles.css** – Style sheets for Dashboard App
- **RedditDashboard.py** – Python file for Streamlit dashboard
- **functions.py** – Helper functions for the Streamlit App
- **reddit-kafka.pem** – Private Key for launching EC2 Instance
- **reddit-logo-new.svg** – Image file for Reddit Logo
- **requirements.txt** – Requirements file with Python libraries for Streamlit installation