Topic Modelling Cryptocurrency News: A Comparative Study Using BERTopic and Hybrid BERTopic-KMeans NLP Models

Abstract

Applying transformer models to cryptocurrency news topic discovery can significantly benefit investors, analysts, and developers by filtering out irrelevant information and highlighting the key topics driving the industry. This study investigates the effectiveness of transformer-based machine learning models, specifically for topic discovery in cryptocurrency news. Data was sourced from various cryptocurrency news platforms using a combination of a WebCrawler, RSS feeds, and API calls, resulting in 1,108 observations comprising titles and descriptions. The data underwent rigorous preprocessing, including text cleaning, tokenization, stop-word removal, and lemmatization. Descriptive statistical analysis was conducted during preprocessing to determine the necessary normalisation steps. Two machine learning models were compared: one utilizing the BERTopic framework for topic modelling and a hybrid model where BERTopic was used solely for embedding, complemented by the KMeans algorithm for clustering. The findings revealed that the BERTopic-only model slightly outperformed the hybrid model in both cluster scoring and qualitative topic word analysis. These results suggest that the BERTopic transformer framework is a powerful tool for extracting relevant topics from cryptocurrency news, with significant potential for trend analysis and content recommendation.

Overview

The cryptocurrency industry is characterized by a vast amount of information generated daily across various platforms, including news articles, social media, and research reports. As a global, 24/7 news cycle, it can be difficult to separate out the most important topics of the day. This report aims to explore the application of natural language processing (NLP) topic discovery techniques to quickly sift through cryptocurrency news articles. This works is inspired by work by Gadi et al. (2021), creating a crypto news summarisation tool (Cryptoblend) that integrated continuous web scraping, a transformer machine learning model, and an agglomerative clustering algorithm. It showed that automated transformer models could sift large numbers of cryptocurrency articles. By applying NLP topic discovery, investors, analysts, and developers can sift through this information to uncover the most relevant topics, such as regulatory changes, technological advancements, market sentiment, and investment opportunities.

The objective of this report is to apply NLP topic discovery techniques to cryptocurrency news articles, to better understand if it can be used to automate the extraction of the most relevant topics and themes.

Data

Data was sourced from various cryptocurrency news sites, with the full list of sources provided in Tables 1 & 2 in 'Appendix A – Data Sources. The selected domains were chosen based on their high activity levels in the industry, as indicated by RSS feed output and daily user traffic. All the sites were centred on US English, reflecting the dominance of US-based news providers in the sector. This focus introduces a US bias, likely influenced by traditional US markets and macroeconomic factors. Given that the sites are US-based, the web crawling process was conducted in compliance with US copyright laws, which protect original works of authorship, including text, images, and other media found on these websites. The data collection adhered strictly to legal requirements, avoiding unauthorized copying or reproduction, and followed the terms of service set by each website.

The original raw data is all drawn from the 19th of August 2024 and consists of 1108 observations of which 208 are RSS feeds and 900 are news articles. The collected metadata initially included author, title, description, URL, and publication date (publishedAt). However, during the preprocessing stage, as described in the Methods section below, the metadata was streamlined to focus solely on the title and description, resulting in 1,075 observations. The reduction was based on the rationale that the collection period already provides temporal context, and the author and URL fields were not deemed necessary for effective topic discovery.

Methods

Computational Environment

Python version '3.8' (Python Software Foundation, 2019) was utilised for all data transformations and machine learning analysis. Please see Appendix C, for the documented code. The code is also available via the accompanying scripts, or the projects code and data can be viewed on the project repository on GitHub (Ross, 2024).

The computational environment included the following:

- Hardware Specifications:
    - CPU: AMZ Ryzen 9 3900X 12-Cor Processor, 3801 Mhz. All processing was conducted on the CPU rather than GPU.
    - RAM: 32 GB DDR 4 RAM at 3200MHz, and,
    - Storage: Samsung SSD 860 QVO 1TB.
- Operating System: Windows 11 Home. This was compatible with all Python tools and libraries.
- Software Environment:
    - Programming Language: Python 3.8.
    - Libraries and Frameworks:

        - Data Collection: Packages for the data collection included requests, Beautiful Soup, random time, urlparse, re, feedparser, json, datetime, and Selenium.
        - Data Preprocessing & NPL Models: Packages include json, pandas, re, nltk, Beautiful Souo, matplotlib.pyplot, seaborn, Counter, numpy, sklearn, and bertopic.
    - Integrated Development Environment (IDE): Visual Studio Code. No additional plugins or extensions.
    - Package Management: Using pip for package installations.
- Computation Considerations: No significant limitations. Model training times were approximately 10 minutes. Significantly larger data sets may have been impacted by processing time.
- Data Storage and Management: Data is stored on both the home PC and on cloud storage. A copy of the code and data is stored on GitHub for reproducibility (Ross, 2024).

WebCrawler, RSS Feeds, and API Calls

The data collection pipeline developed for the report consisted initially of a WebCrawler designed to scrape the required data. The websites selected for the WebCrawler can be seen in Appendix A – Data Sources.

The WebCrawler was written in Python using the Beautiful Soup library (Richardson, 2024). Beautiful Soup allows for easy navigation and search within a webpage's structure, making it straightforward to locate specific elements by tag, attribute or text content. To assist the crawler navigating websites requiring JavaScript driven interactions, the Selenium package was utilised (Software Freedom Conservancy, 2024). The goal was to extract title, date and description, from the domains utilising specific selectors.

To ensure the crawler adhered to the websites' terms of service and not infringe on copyright, it checked each domain's robots.txt file to confirm that scraping News/Article pages was permitted. The crawler was programmed to avoid any URLs that matched patterns specified in the Disallow directives. This occurred for the domains CoinTelegraph and CoinDesk. Additionally, an analysis revealed that all but one of the five sites (CoinDesk) were using Cloudflare (n.d.), a proxy service that provides robust anti-bot protection. To minimize the risk of triggering anti-scraping measures, the crawler employed several industry standard strategies, including introducing random delays between requests to mimic human behaviour, user-agent handling, using headless browsers with Selenium to fully render pages as a user would, and leveraging proxy services (Lotfi, 2021). Despite these precautions, the crawler was still unable to bypass the anti-scraping measures effectively, indicating that these sites were quite sophisticated in preventing scraping.

As the scraping process was not successful, 'Really Simple Syndication' (RSS) data from the same sites was obtained. In addition, two 'Application Interface' (API) aggregators

were used, one being a general API news aggregator called News API and a crypto community news aggregator called CryptoPanic (n.d.). Both the RSS and API data had the same metadata structure and therefore was sufficient to obtain a robust dataset. The data was all dated from the 19th of August 2024 and was outputted in a JSON format that could then be read by the next python script for data processing and machine learning analysis.

Data Preprocessing

Initial preprocessing of the data set involved performing an initial text cleaning, by removing HTMP tags, non-textual elements and special characters. This included converting all text to lowercasing and removing punctuation (Bird, 2009). Feature selection was conducted at this point by reducing the data to the title and description, as it was determined that the other metadata was of little additional value for topic modelling. The rest of the normalisation steps of tokenisation, determining the need for stopwords, stemming/lemmatization and encoding are conducted in the following sections.

Exploratory Data Analysis

Descriptive statistics and text length analysis was conducted to check the dataset size and look for evidence of missing data or outliers. At this stage one-word titles and empty descriptions were removed. This reduced the total data set to 1075.

Tokenisation was then conducted splitting the text into individual words (Jurafsky & Martin, 2008). A word count distribution and 'most common words' analysis was conducted to determine if stopword removal was required (Manning, 2008). This could be inferred by the skewness of the distribution as well as a visual inspection of the most common words. From this analysis, stopword removal was employed, utilising the standard NLTK list (Bird, 2009). A second word count distribution and 'most common word' analysis was conducted.

Lemmatization

As it is important to maintain semantic integrity, lemmatization was selected over stemming (Manning, 2008). Lemmatization reduces words to their base or dictionary form, ensuring that the root words is an actual word with meaning in the language. Lemmatization also handles inflected words (e.g. running to run), ensuring that words with different surface forms but the same base meaning are grouped together. This is essential in topic modelling where the goal is to capture underlying themes or topics, not just word forms.

BERTopic Embeddings

Embeddings are low-dimensional vectors that capture semantic meaning of words based on their context within a corpus. As discussed by Jurafsky and Martin (2008), traditional vectorization techniques like TF-IDF, which rely on frequency counts, are less effective than embeddings for capturing the semantic relationships between words. This allows topic models to capture more nuanced and context-aware themes, leading to more meaningful and coherent topics.

To implement embeddings, BERTopic is being utilised (Devlin, 2019). BERTopic is a topic modelling framework that leverages the pre-trained transformer based embedding model BERT to generate contextual embeddings of text. BERTopic is superior to traditional vectorisation as it can capture deep contextual meaning, handle synonymy (grouping similar words together) and polysemy (differentiating between different meanings of the same word), reduces dimensionality without information loss, and improves topic coherence.

Machine Learning Models

To explore the effectiveness of deep learning transformer-based approaches for topic modelling, two machine learning models were selected to evaluate the performance of the BERTopic/BERT method. The first model was a hybrid, leveraging BERTopic to generate embeddings and then applying K-Means, an unsupervised learning algorithm that clusters data points based on their similarities, with centroids representing the derived topics (REFERENCE). The hyperparameters for the K-means algorithm include:

- Clusters (5): The number of clusters to form and number of centroids to generate.
- random_state (42): The random number generation for centroid initialisation.
- init (k-means++): The default method for initialisation.
- n_init (10): Number of time the K-means algorithm will be run with different centroid seeds.
- max_iter (300): The default maximum number of iterations of the K-means algorithm in a single run.
- tol (1e-4): The default relative tolerance with regards to inertia to declare convergence.

This hybrid model was then compared to the BERTopic model alone, which employs the following techniques with hyperparameters defined:

- Embedding Model (MiniLM-L6-v2): A lightweight version of BERT. Selected due to the small data size.
- Uniform Manifold Approximation (UMAP): UMAP reduces the numbers of dimensions in a dataset while retaining the original structure and relationships (McInnes, 2018). Unlike linear techniques like Principal Component Analysis,

UMAP can capture complex non-linear relationships in the data. Thee selected hyperparameters include:
    - n_neighbours (15): Default neighbourhood size used for manifold approximation.
    - n_components (5): Default number of dimensions to reduce the embeddings to.
    - min_dist (0.1): How tightly UMAP is allowed to pack points together.
    - metric (cosine): The metric used to measure the distance between points.
- Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) algorithm. HDBSCAN organizes embeddings into clusters that can be interpreted as topics, with each cluster representing a set of semantically related words or phrases (Campello, 2013). The selected hyperparameters include:
    - min_cluster_size (10): The minimum size of clusters.
    - min_samples (1): The number of samples in a neighbourhood for a point to be considered a core point
    - cluster_selection_epsilon (0.0): The distance threshold for clustering.
    - Top-N Words per Topic (10): The number of top words to extract and display for each topic.

BERTopic further identifies and labels the clusters with representative words or phrases, which are considered the main topics extracted from the text.

By comparing a BERTopic-K-Means hybrid model against using BERTopic alone, the aim is to investigate the trade-offs between the two clustering models in terms of coherence, interpretable topics, how they handle noise and outliers, and which approach better scales with the data.

Model Validation and Testing

Given the relatively small size of the dataset, K-fold cross-validation was chosen over a traditional train-test split. K-fold cross-validation systematically splits the data into multiple subsets, ensuring that the model is tested on different portions of the data, which provides a more robust assessment of its generalization ability (Kohavi, 1995). This approach was selected to maximize the use of the limited data, as small datasets can increase the risk of overfitting, reduce generalizability, and amplify sample bias, potentially skewing results. Five folds were used, with the model's performance evaluated across each iteration, and the final performance determined by averaging the results from all iterations.

Performance Metrics

Both models were evaluated using the Silhouette Score and the Davies-Bouldin Index (DBI). The Silhouette Score assesses the quality of a clustering model by measuring how similar each data point is to others within the same cluster compared to points in

different clusters (Rousseeuw, 1987). The Davies-Bouldin Index evaluates the clustering performance by calculating the average similarity ratio between each cluster and its most similar cluster, providing a quantitative measure of the trade-off between cluster compactness and separation (Davies, 1979).

Using both metrics together provides a comprehensive evaluation of clustering performance. The Silhouette Score emphasizes the cohesion and separation of individual data points, while the Davies-Bouldin Index assesses the overall structure by considering both the compactness of clusters and their separation from each other. This dual approach minimizes the risk of choosing a model that excels in one metric but underperforms in another, resulting in more robust and reliable clustering outcomes.

Following cluster scoring, the relevance and coherence of topics are evaluated. In the hybrid model, this assessment is based on the word frequencies within each cluster. For the BERTopic-only model, a coherence scoring technique known as Class-based Term Frequency-Inverse Document Frequency (C_TF-IDF) is employed. This metric, specifically adapted from traditional TF-IDF for topic modelling, evaluates and enhances the coherence of the generated topics, resulting in clearer and more meaningful topic models (Grootendorst, 2022). The results for both models can be found in 'Appendix B – Model Topic Output.' Among the five clusters generated by the hybrid model and the five sample clusters selected from the 30 produced by the BERTopic model, the two most prevalent clusters from each model were chosen for analysis.

Results & Discussion

Exploratory Data Analysis

The initial summary statistics revealed that the minimum title length of 1 word and the minimum description length of 0 words were too low to be meaningful. This can be seen in Table 3.

Analysis	Title Length	Description Length
Count	1108	1108
Mean	10.32	33.14
Std	3.66	24.73
Min	1	0
25%	9	21
50%	10	28
75%	12	50
Max	22	435

Table 3: Text Length Analysis Descriptive Statistics

This was reinforced by an analysis of the title length distribution and description length distribution that can be seen in Figure 1 below. There is a relatively high frequency of both title and description with no words or only 1 or two.

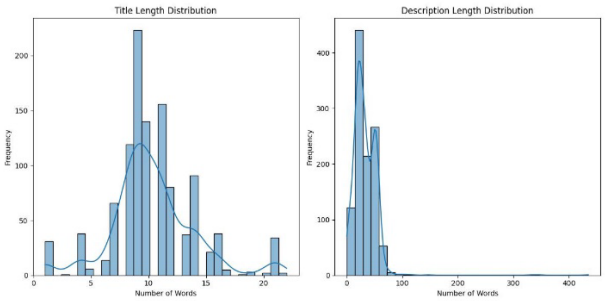

Figure 1: Text Length Distribution & Description Length Distribution

A visual inspection revealed that titles consisting of one or two words often lacked context and failed to contribute valuable information to the analysis. While analysing the dataset, unusually long titles were also examined; however, they were found to contain sufficient context and meaning, warranting their inclusion. As a result, titles and descriptions shorter than three words were removed to enhance the dataset's quality. This refinement reduces noise and ensures that only text with adequate content is retained, leading to more reliable embeddings and ultimately improving the model's ability to accurately identify and cluster topics.

After tokenisation, a word count distribution and 'most common words' analysis was conducted to determine if stopword removal was required. It can be seen in Figure 2 and 3 below, that the most common words were stop words confirming the requirement.

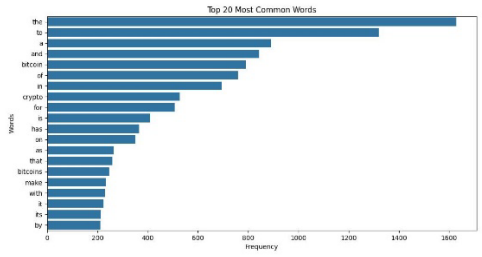

Figure 2: Top 20 Most Common Words

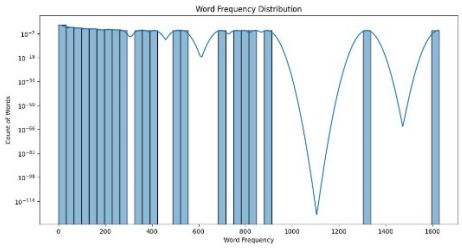

Figure 3: Word Frequency Distribution

The same analysis was conducted again after the application of stop word removal, which can be seen in Figure 4 & 5 below.

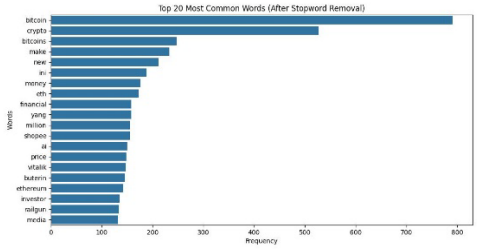

Figure 4: 20 Most Common Words (Post stopword)

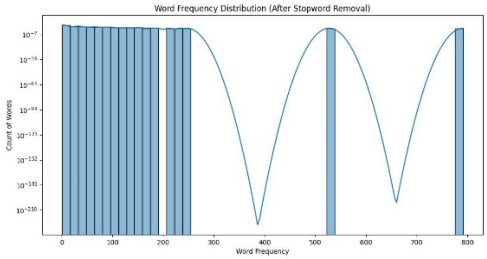

Figure 5: Word Frequency Distribution (Post stopword)

Figure 4 reveals a higher concentration of meaningful terms, with the majority being industry-specific jargon. This indicates a more focused and relevant vocabulary within the dataset. Figure 5 further illustrates a tightened word frequency distribution, although two prominent outliers—'bitcoin' and 'crypto'—remain. While Bitcoin is certainly a relevant and specific topic, the term 'crypto' is more ambiguous. Future iterations could benefit from the development of a customized stopword list that excludes industry-specific terms that are not relevant to the analysis. This enhancement would help in reducing noise and improving the precision of topic modelling results.

Model Evaluation

The average Silhouette and DBI scores both models can be seen in Table 4 below. For the BERTopic + KMeans model a silhouette score of 0.72 is quite good, suggesting that the clusters formed by the model are well-separated and the data points are well-clustered. The DBI score of 0.46 is relatively low, which is favourable. This indicates that the clusters are not only well separated but also relatively compact. The BERTopic model has performed only slightly better in both scores than the hybrid model. It exhibits marginally better cohesion and separation of clusters as well as slightly more compact clusters.

Models	Silhouette Score	DBI Score
BERTopic + KMeans	0.72	0.46
BERTopic Only	0.73	0.42

Table 4: Average Silhouette and DBI Score

For the topic word assessment, two samples for each model were selected from the output that can be seen in Appendix B – Model Topic Output. For the hybrid model, Clusters one and two were most prevalent in terms of word frequency as can be seen as follows:

Cluster 1: {'bitcoin': 85, 'make': 59, 'bitcoins': 50, 'crypto': 38, 'money': 36, 'eth': 29, 'scam': 28, 'new': 28, 'railgun': 28, 'tiktok': 27}

Cluster 2: {'crypto': 49, 'bitcoin': 43, 'shopee': 30, 'membatalkan': 24, 'pinjam': 24, 'cara': 18, 'layanan': 18, 'asset': 16, 'price': 16, 'digital': 14}

Cluster 1 seems to revolve around discussions about Bitcoin and other cryptocurrencies, particularly focusing on how to make money with them. The presence of words like "scam" suggests that there may be discussions related to fraudulent activities or scams within the crypto space. An investigation of the term railgun reveals that it is a crypto mixer that has drawn attention for associated scams around it though this was notable back in July.

Cluster 2 appears to be focused on the use of cryptocurrencies in Southeast Asia, particularly in relation to digital financial services. "Shopee" is a well-known e-commerce platform in Southeast Asia, and words like "membatalkan" (which means "cancel" in Indonesian/Malay) and "pinjam" (which means "borrow") suggest that this cluster is discussing financial transactions or services involving cryptocurrencies, possibly in the context of e-commerce or fintech. The inclusion of terms like "cara" (which means "how" or "way") and "layanan" (which means "service") further supports this interpretation, indicating discussions about how to use or provide crypto-related services in a digital marketplace. An investigation into the terms found no prominent news related to Shoppe, Indonesia, or Malay during the 19th August.

For the BERTopic only model, Topics 3 and 5 were selected:

Topic 3: {'price': 0.16368267364793093, 'ripple': 0.14730736577177658, 'level': 0.13916563928596684, 'xrp': 0.11030775957494215, 'ethereum': 0.10973091037592982, 'respective': 0.08987488332615182, 'stability': 0.08987488332615182, 'whereas': 0.08987488332615182, 'consolidates': 0.08987488332615182, 'failing': 0.08987488332615182}

Topic 5: {'dacceth': 0.19873341832325686, 'tech': 0.17952497544301618, 'vitalik': 0.15593952398127664, 'buterin': 0.15184940318335566, 'domain': 0.13179972803273535, 'cofounder': 0.12868606199083793, 'register': 0.12813862447627047, 'countering': 0.11829685497630239, 'promote': 0.11829685497630239, 'rapid': 0.11808755115279397}

For Topic 3, the high coherence scores for words like "price," "ripple," "level," and "xrp" indicate that this topic is strongly centred around discussions on cryptocurrency prices, particularly focusing on Ripple (XRP) and Ethereum. It likely covers discussions related to the price movements of specific cryptocurrencies, particularly Ripple (XRP) and Ethereum. An investigation into Ripple and its price didn't find anything of significant note.

For Topic 5, the high coherence scores for words like "dacceth," "tech," "vitalik," and "buterin" indicate that this topic is likely focused on technological aspects of Ethereum, specifically involving its co-founder Vitalik Buterin. Further research, found that Vitalik had registered on the 19th of August a new domain name 'dacc.eth' to the Ethereum Name Service.

While the clustering of both models seems quite good, it appears (purely qualitatively), that the BERTopic only model is slightly more relevant.

Conclusions

With a Silhouette Score of 0.73 and a DBI Score of 0.42, the BERTopic-only model demonstrates slightly better performance compared to the hybrid model. The qualitative assessment of the topic word outputs for both models also yielded similar results, with the BERTopic-only model showing a marginal advantage.

However, this study was limited by the small dataset and the constrained feature set used from the outset. Another significant limitation was the interpretability of the topic word assessment; it remains challenging to accurately gauge how well each model captured the major topics, particularly beyond a certain level of subjective understanding.

The WebCrawler used in the data collection process could have been significantly improved by employing a broader range of anti-scraping techniques, such as rotating or retaining IP addresses, preserving user agents, utilising stealth plugins, and incorporating dynamic adaptation strategies. However, many of these methods tread ethical and legal boundaries, and given that much of the data was readily available via APIs, the potential benefits may not justify their implementation.

For the models, a crucial enhancement would be the development of an industry-specific stopword list. Furthermore, introducing a method to score topics based on their relevance and importance, and then aligning the predicted topics with this scoring system, could offer a more quantitative and objective assessment of the topic word clusters.

Overall, BERTopic has shown that it is a potentially powerful too in predicting crypto news topic words. Its potential for further topic identification, trend analysis and content recommendations are compelling.

References

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Toolkit (NLTK)*. Retrieved from https://www.nltk.org/

Campello, R. J. G. B., Moulavi, D., & Sander, J. (2013). *Density-based clustering based on hierarchical density estimates*. In J. Pei, V. Tseng, L. Cao, H. Motoda, & G. Xu (Eds.), *Advances in Knowledge Discovery and Data Mining* (Vol. 7819, pp. 160-172). Springer.

Cloudflare, Inc. (n.d.). *Cloudflare: Content delivery network, DDoS protection, and security*. Retrieved August 21, 2024, from https://www.cloudflare.com/

CryptoPanic. (n.d.). *Crypto news aggregator and portfolio tracker*. Retrieved August 21, 2024, from https://cryptopanic.com/

Davies, D. L., & Bouldin, D. W. (1979). *A cluster separation measure. IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2), 224-227.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: *Human Language Technologies, Volume 1*.

Gadi, R., Varab, D., & Kryściński, W. (2021). Cryptoblend: An AI-Powered Tool for Aggregation and Summarization of Cryptocurrency News. *Informatics*, 4(2), 13

Grootendorst, M. (2022). *BERTopic: Neural topic modeling with a class-based TF-IDF procedure*. arXiv.

Jurafsky, D., & Martin, J. H. (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (2nd ed.). Prentice Hall.

Kohavi, R. (1995). *A study of cross-validation and bootstrap for accuracy estimation and model selection*. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 1137-1143). Morgan Kaufmann Publishers Inc.

Lotfi, C., Srinivasan, S., Ertz, M., & Latrous, I. (2021). *Web scraping techniques and applications: A literature review*. In *SCRS Conference Proceedings on Intelligent Systems* (pp. 381-394).

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

McInnes, L., Healy, J., & Melville, J. (2018). *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. arXiv preprint arXiv:1802.03426.

NewsAPI.org. (n.d.). *News API - A JSON API for live news and blog articles*. Retrieved August 21, 2024, from https://newsapi.org/

Python Software Foundation. (2019). *Python Language Reference, version 3.8*. Available at https://www.python.org.

Richardson, L. (2024). *Beautiful Soup documentation*. Crummy. Retrieved August 21, 2024, from https://www.crummy.com/software/BeautifulSoup/

Ross, James (2024). *MA5851_Assessment3*. Github. https://github.com/jimy-r/MA5851_Assessment3

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics, 20, 53-65.

Software Freedom Conservancy. (2024). *Selenium: Browser automation*. Retrieved August 21, 2024, from https://www.selenium.dev/

Appendix A – Source List

WebCrawler & RSS Feeds

Domain	URL
CoinTelegraph	https://cointelegraph.com
CoinDesk	https://www.coindesk.com
BitcoinMagazine	https://bitcoinmagazine.com
decrypt	https://decrypt.co
CryptoNews	https://cryptonews.com

Table 1: WebCrawler & RSS Feeds Domain List

API Domain List

API domains sourced through CryptoPanic (n.d.) and News API (n.d.).

Domain	URL
AMBCrypto	ambcrypto.com
Altcoin Buzz	altcoinbuzz.io
Atlas21	atlas21.com
BeInCrypto	beincrypto.com
Feeds 2.0	feeds2.benzinga.com
BitcoinMagazine	bitcoinmagazine.com
Bitcoin	bitcoin.com
Bitcoinist	bitcoinist.com
BlockWorks	blockworks.co
CoinDesk	coindesk.com
Coingape	coingape.com
CoinJournal	coinjournal.net
CoinTelegraph	cointelegraph.com
Coinpaper	coinpaper.com
Coinspeaker	coinspeaker.com
Crypto Economy	crypto-economy.com
Crypto Valley Journal	cryptovalleyjournal.com
CryptoBriefing	cryptobriefing.com
Crypto Daily	cryptodaily.co.uk
CryptoGlobe	cryptoglobe.com
CryptoSlate	cryptoslate.com
CryptoNews	cryptonews.com
The Defiant	thedefiant.io
Decrypt	decrypt.co
Fxpro	fxpro.news
Fxstreet	fxstreet.com
Bloomberg	bloomberg.com
Cryptopolitan	cryptopolitan.com
Cryptopotato	cryptopotato.com
Finbold	finbold.com
Forexlive	forexlive.com
Forkast	forkast.news
Franknez	franknez.com
Crypto Franknez	crypto.franknez.com
Live Bitcoin News	livebitcoinnews.com
NewsBtc	newsbtc.com
NFTgators	nftgators.com
Protos	protos.com
SolanaFloor	solanafloor.com
The Block	theblock.co
The Coin Republic	thecoinrepublic.com
The Daily Hodl	dailyhodl.com
U.Today	u.today
ZyCrypto	zycrypto.com
Action Forex	actionforex.com
BabyPips	babypips.com
Bankless	bankless.com
CoinCU	coincu.com
CoinGabbar	coingabbar.com
CoinPaprika	coinpaprika.com
Coinpedia	coinpedia.org
CryptoDNES	cryptodnes.bg
Droom Droom	droomdroom.com
Finance Magnates	financemagnates.com
Investing News	investingnews.com
MarketPulse	marketpulse.com
SFC Today	sfctoday.com
Tokenist	tokenist.com

Table 2: APIs 'CryptoPanic' and 'News API' Domain List

Appendix B – Model Topic Output

BERTopic + KMeans - Topic Word Output

Cluster 0: {'crypto': 13, 'bitcoin': 12, 'scam': 8, 'harris': 7, 'investment': 7, 'new': 6, 'year': 6, 'meme': 6, 'etf': 4, 'ava': 4}

Cluster 1: {'bitcoin': 85, 'make': 59, 'bitcoins': 50, 'crypto': 38, 'money': 36, 'eth': 29, 'scam': 28, 'new': 28, 'railgun': 28, 'tiktok': 27}

Cluster 2: {'crypto': 49, 'bitcoin': 43, 'shopee': 30, 'membatalkan': 24, 'pinjam': 24, 'cara': 18, 'layanan': 18, 'asset': 16, 'price': 16, 'digital': 14}

Cluster 3: {'crypto': 13, 'bitcoin': 12, 'bitcoins': 11, 'new': 9, 'money': 8, 'meme': 5, 'last': 5, 'im': 5, 'strike': 5, 'app': 5}

Cluster 4: {'bitcoin': 18, 'etf': 8, 'btc': 7, 'price': 6, 'trump': 6, 'eth': 5, 'crypto': 4, 'supply': 4, 'shock': 4, 'scam': 4}

BERTopic Only - Coherence Scores

Topic 1: {'market': 0.05562796925517, 'price': 0.05055050325452919, 'bitcoin': 0.039973084254078604, 'stock': 0.034422806131276545, 'analyst': 0.032538198669848875, 'could': 0.030296828483263205, 'rate': 0.029186691718804867, 'btc': 0.0282709176876356, 'mining': 0.026684795803323353, 'altcoin': 0.024982592465438103}

Topic 2: {'harris': 0.08206515171588624, 'trump': 0.07549503681082939, 'conference': 0.06180309861760984, 'kamala': 0.060358766981309, 'president': 0.04931318201079687, 'donald': 0.04563069577664139, 'presidential': 0.04252237927410086, 'election': 0.03874607157991182, 'former': 0.03621895025902238, 'democrat': 0.03368429032404072}

Topic 3: {'price': 0.16368267364793093, 'ripple': 0.14730736577177658, 'level': 0.13916563928596684, 'xrp': 0.11030775957494215, 'ethereum': 0.10973091037592982, 'respective': 0.08987488332615182, 'stability': 0.08987488332615182, 'whereas': 0.08987488332615182, 'consolidates': 0.08987488332615182, 'failing': 0.08987488332615182}

Topic 4: {'wazirx': 0.0603325498505967, 'russia': 0.05656176548493441, 'google': 0.04767318661774023, 'court': 0.04349262287188852, 'report': 0.04304256197404439, 'bitconnect': 0.041858635982237546, 'crypto': 0.040667792436100324, 'indian': 0.038512739198147364, 'laptop': 0.038512739198147364, 'employee': 0.03558487325881788}

Topic 5: {'dacceth': 0.19873341832325686, 'tech': 0.17952497544301618, 'vitalik': 0.15593952398127664, 'buterin': 0.15184940318335566, 'domain': 0.13179972803273535, 'cofounder': 0.12868606199083793, 'register': 0.12813862447627047, 'countering': 0.11829685497630239, 'promote': 0.11829685497630239, 'rapid': 0.11808755115279397}

Appendix C - Python code

The following is the Python code output from Visual Studio, for the completion of all data processing and analysis within this study. The code is broken into two scripts:

- Part 1: Data Collection Pipeline
- Part 2: Data Preprocessing, Exploratory Analysis, and NLP Models

```
# Assessment 3 - Webcrawler and NLP System
# James Ross (14472266)
# PART 1 - Data Collection Pipeline
# For project on GitHub: https://github.com/jimy-r/MA5851_Assessment3

import requests
from bs4 import BeautifulSoup
import random
import time
from urllib.parse import urlparse
import re
import feedparser
import json
from datetime import datetime, timedelta, timezone
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options

# Setup Chrome options (headful browser)
chrome_options = Options()
# chrome_options.add_argument("--headless")  # Running headless
chrome_options.add_argument("--ignore-certificate-errors")  # Ignore certificate errors
chrome_options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36")

# Replace with the path to your WebDriver
service = Service('C:/Users/jimy/chromedriver.exe')
driver = webdriver.Chrome(service=service, options=chrome_options)

# Function to fetch and parse the robots.txt file
def fetch_robots_txt(base_url):
    robots_url = base_url.rstrip('/') + '/robots.txt'
    try:
        response = requests.get(robots_url)
        response.raise_for_status()
        return response.text
    except requests.exceptions.RequestException as e:
```

```
        print(f"Error fetching robots.txt: {str(e)}")
        return ""

# Function to check if a URL is allowed based on robots.txt
def is_allowed_by_robots(robots_txt, url):
    parsed_url = urlparse(url)
    path = parsed_url.path

    disallowed_patterns = []
    for line in robots_txt.splitlines():
        if line.startswith('Disallow:'):
            pattern = line.split(':')[1].strip()
            if pattern:
                disallowed_patterns.append(re.escape(pattern).replace('
\\*', '.*'))

    for pattern in disallowed_patterns:
        if re.match(pattern, path):
            print(f"Robots.txt forbidding access to: {url}")
            return False

    return True

# Function to extract information using multiple selectors
def extract_with_fallback(soup, selectors):
    for selector in selectors:
        element = soup.select_one(selector)
        if element:
            return element.text.strip()
    return None

# General function to extract title, date, and description
def extract_article_details(soup):
    # Define possible selectors for title, date, and description
    title_selectors = ['h1', 'h2', 'h3', '.post-title', '.article-
title', '.headline']
    date_selectors = ['time', '.date', '.published-date', '.pubdate']
    description_selectors = ['meta[name="description"]', '.summary',
'.description', 'p']

    title = extract_with_fallback(soup, title_selectors)
    date = extract_with_fallback(soup, date_selectors)
    description = extract_with_fallback(soup, description_selectors)

    # If the date is found in a 'time' tag, try to get the datetime
attribute
    if date:
        date_tag = soup.select_one('time')
```

```
            if date_tag and 'datetime' in date_tag.attrs:
                date = date_tag['datetime']
            else:
                date = datetime.now(timezone.utc).isoformat()

        return {
            "title": title,
            "date": date,
            "description": description
        }

    # Function to extract article summary details from the main page
    def extract_article_summary(article, base_url):
        link = article.find('a', href=True)
        if link:
            return base_url + link['href']
        return None

    # Function to crawl and parse news articles from a given website using
    Selenium
    def crawl_news_with_selenium(base_url, news_page, article_selector):
        # Fetch and parse robots.txt
        robots_txt = fetch_robots_txt(base_url)

        driver.get(base_url + news_page)
        # Random delay to mimic human behavior
        time.sleep(random.uniform(5, 10))  # 5 to 10 seconds delay

        soup = BeautifulSoup(driver.page_source, 'html.parser')
        articles = soup.select(article_selector)

        if not articles:
            print(f"No articles found on {base_url + news_page}.")
            return []

        news_data = []
        for article in articles:
            article_url = extract_article_summary(article, base_url)
            if article_url and is_allowed_by_robots(robots_txt,
    article_url):
                driver.get(article_url)
                time.sleep(random.uniform(5, 10))  # Delay to mimic user
    behavior
                article_soup = BeautifulSoup(driver.page_source,
    'html.parser')
                full_article_data = extract_article_details(article_soup)
                if full_article_data['title'] and
    full_article_data['description']:
```

```
                news_data.append(full_article_data)

            # Random delay to avoid triggering anti-scraping mechanisms
            time.sleep(random.uniform(5, 10))  # 5 to 10 seconds delay

    if news_data:
        print(f"Successfully crawled {len(news_data)} articles from {base_url + news_page}.")
    else:
        print(f"Failed to crawl articles from {base_url + news_page}.")

    return news_data

# Function to parse and format RSS feed data
def parse_rss_feed(feed_url, decrypt=False, cryptonews=False):
    feed = feedparser.parse(feed_url)
    articles = []

    for entry in feed.entries:
        if decrypt:
            article = {
                "title": entry.title,
                "description": entry.description,
                "publishedAt": entry.published
            }
        elif cryptonews:
            article = {
                "title": entry.title,
                "description": entry.description
            }
        else:
            article = {
                "author": entry.get('author', 'Unknown'),
                "title": entry.title,
                "description": entry.summary,
                "url": entry.link,
                "urlToImage": entry.get('media_content', [{}])[0].get('url', None),
                "publishedAt": entry.published if 'published' in entry else datetime.now(timezone.utc).isoformat(),
                "content": entry.summary
            }
        articles.append(article)

    return articles

# Function to pull data from RSS feeds
def pull_rss_feeds():
```

```
    feeds = [
        {"url": "https://cointelegraph.com/rss", "decrypt": False, "cryptonews": False},
        {"url": "https://www.coindesk.com/arc/outboundfeeds/rss/", "decrypt": False, "cryptonews": False},
        {"url": "https://bitcoinmagazine.com/.rss/full/", "decrypt": False, "cryptonews": False},
        {"url": "https://decrypt.co/feed", "decrypt": True, "cryptonews": False},
        {"url": "https://cryptonews.com/rss/", "decrypt": False, "cryptonews": True}
    ]

    all_articles = []

    for feed in feeds:
        articles = parse_rss_feed(feed["url"], decrypt=feed["decrypt"], cryptonews=feed["cryptonews"])
        all_articles.extend(articles)

    return all_articles

# Function to fetch recent news articles from News API with pagination
def fetch_news_api(api_key, base_url, query='Crypto', days=14, page_size=100, max_calls=1):
    end_date = datetime.now(timezone.utc)
    start_date = end_date - timedelta(days=days)
    current_page = 1
    all_articles = []

    while current_page <= max_calls:
        url = (
            f"{base_url}/everything?q={query}"
            f"&from={start_date.isoformat()}"
            f"&to={end_date.isoformat()}"
            f"&sortBy=popularity"
            f"&apiKey={api_key}"
            f"&pageSize={page_size}"
            f"&page={current_page}"
        )

        try:
            response = requests.get(url)
            response.raise_for_status()
            articles_data = response.json().get('articles', [])

            if not articles_data:
```

```
                print(f"No more articles found after {len(all_articles)} articles.")
                break

            for article in articles_data:
                all_articles.append({
                    "author": article.get('author', 'Unknown'),
                    "title": article.get('title', 'No Title'),
                    "description": article.get('description', 'No description available.'),
                    "url": article.get('url'),
                    "urlToImage": article.get('urlToImage'),
                    "publishedAt": article.get('publishedAt', datetime.now(timezone.utc).isoformat()),
                    "content": article.get('content', 'No content available.')
                })

            print(f"Page {current_page}: Successfully fetched {len(articles_data)} articles.")
            current_page += 1

            # If the number of articles is less than page_size, it means there are no more articles
            if len(articles_data) < page_size:
                break

        except requests.exceptions.HTTPError as http_err:
            print(f"Error during requests to News API on page {current_page}: {http_err}")
            break

        except requests.exceptions.RequestException as req_err:
            print(f"Request error: {req_err}")
            break

    print(f"Total articles fetched from News API: {len(all_articles)}")
    return all_articles

# Function to fetch recent news articles from CryptoPanic API
# Note: Originally set 1000 max_articles but hit API limit at 800.
def fetch_cryptopanic_news(auth_token, days=14, max_articles=600, requests_per_second=5):
    end_date = datetime.now(timezone.utc)
    start_date = end_date - timedelta(days=days)
    base_url = f"https://cryptopanic.com/api/v1/posts/?auth_token={auth_token}&metadata=true"
```

```
    all_articles = []
    current_page = 1

    while len(all_articles) < max_articles:
        url = f"{base_url}&page={current_page}"

        try:
            response = requests.get(url)
            response.raise_for_status()
            articles_data = response.json().get('results', [])

            if not articles_data:
                print(f"No more articles found after fetching
{len(all_articles)} articles.")
                break

            for article in articles_data:
                published_date =
datetime.fromisoformat(article['published_at'][:-
1]).replace(tzinfo=timezone.utc)
                if start_date <= published_date <= end_date:
                    news_data = {
                        "title": article.get('title', 'No Title'),
                        "description": article.get('metadata',
{}).get('description', 'No description available.'),
                        "url": article.get('url'),
                        "publishedAt": article.get('published_at',
datetime.now(timezone.utc).isoformat())
                    }

                    # Optionally add image metadata if needed
                    image_url = article.get('metadata',
{}).get('image', None)
                    if image_url:
                        news_data['urlToImage'] = image_url

                    all_articles.append(news_data)

                    # Stop if the max number of articles reached
                    if len(all_articles) >= max_articles:
                        print(f"Reached the maximum of {max_articles}
articles.")
                        break

            print(f"Page {current_page}: Successfully fetched
{len(articles_data)} articles.")
            current_page += 1

```

```
            # Respect the rate limit by sleeping between requests
            time.sleep(1 / requests_per_second)

            # Check again after processing the page to exit if we've hit the max
            if len(all_articles) >= max_articles:
                break

        except requests.exceptions.HTTPError as http_err:
            print(f"Error during requests to CryptoPanic API on page {current_page}: {http_err}")
            break

        except requests.exceptions.RequestException as req_err:
            print(f"Request error: {req_err}")
            break

    print(f"Total articles fetched from CryptoPanic API: {len(all_articles)}")
    return all_articles

# Function to save the combined data to a JSON file
def save_to_json(data, filename='crypto_news.json'):
    with open(filename, 'w') as json_file:
        json.dump(data, json_file, indent=4)
        print(f"Data saved to {filename}")

# Function to count unique titles in the JSON file and output the count
def count_unique_titles(filename='crypto_news.json'):
    with open(filename, 'r') as json_file:
        articles = json.load(json_file)

    unique_titles = set()
    for article in articles:
        unique_titles.add(article.get('title', 'No Title'))

    print(f"{len(unique_titles)} unique news pieces have been collected.")

# Main function to run the web crawler, RSS feed parser, and API fetcher
if __name__ == "__main__":
    # Crawl CoinTelegraph, CoinDesk, Bitcoin Magazine, Decrypt, and CryptoNews news pages
    telegraph_articles = crawl_news_with_selenium('https://cointelegraph.com', '/news', '.post-card-inline__title')
```

```
    coindesk_articles = crawl_news_with_selenium('https://www.coindesk.com', '/', '.card-title')
    bitcoinmagazine_articles = crawl_news_with_selenium('https://bitcoinmagazine.com', '/articles', 'h2 a')
    decrypt_articles = crawl_news_with_selenium('https://decrypt.co', '/news', '.post-title')
    cryptonews_articles = crawl_news_with_selenium('https://cryptonews.com', '/news', '.article__title')

    webcrawler_articles = (telegraph_articles + coindesk_articles + bitcoinmagazine_articles +
                           decrypt_articles + cryptonews_articles)

    print(f"Total articles fetched from webcrawler: {len(webcrawler_articles)}")

    # Pull articles from RSS feeds
    rss_articles = pull_rss_feeds()
    print(f"Total articles fetched from RSS feeds: {len(rss_articles)}")

    # Fetch recent news articles using News API
    api_key = 'c4121bc7f94a4427ae7b44109ea518c5'  # Your provided API key
    base_url = 'https://newsapi.org/v2'  # Base URL for News API
    news_api_articles = fetch_news_api(api_key, base_url, query='Crypto', days=14, page_size=100, max_calls=1)

    # Fetch recent news articles using CryptoPanic API
    cryptopanic_auth_token = 'e3941a91a81a66a7773b334c8fc9d2fe0272dff4'  # Your provided CryptoPanic API key
    cryptopanic_articles = fetch_cryptopanic_news(cryptopanic_auth_token, days=14)

    # Combine the results
    all_articles = webcrawler_articles + rss_articles + news_api_articles + cryptopanic_articles

    # Save the combined articles to a JSON file
    if all_articles:
        save_to_json(all_articles)

    # Count and output the number of unique titles
    count_unique_titles()
```

```
•
•       # Close the Selenium WebDriver
•       driver.quit()
```

```
# Assessment 3 - Webcrawler and NLP System
# James Ross (14472266)
# PART 2 - Data Preprocessing, Exploratory Analysis, and NLP Models
# For project on GitHub: https://github.com/jimy-r/MA5851_Assessment3

import json
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from bs4 import BeautifulSoup
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import numpy as np
from sklearn.model_selection import KFold
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score
from bertopic import BERTopic
from sklearn.decomposition import PCA

# Download required NLTK data files
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Load the JSON data
with open('raw_data.json', 'r') as f:
    data = json.load(f)

# Extract relevant fields
articles = []
for item in data:
    article = {
        'title': item.get('title', ''),
        'description': item.get('description', ''),
    }
    articles.append(article)

# Convert to DataFrame
df = pd.DataFrame(articles)
```

```
# Text Cleaning
def clean_text(text):
    # Remove HTML tags
    text = BeautifulSoup(text, "html.parser").get_text()
    # Remove special characters and digits
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    return text

df['cleaned_title'] = df['title'].apply(clean_text)
df['cleaned_description'] = df['description'].apply(clean_text)

# Summary Statistics
print("Summary Statistics:")
print(df.describe(include='all'))

# Text Length Analysis
df['title_length'] = df['cleaned_title'].apply(lambda x: len(x.split()))
df['description_length'] = df['cleaned_description'].apply(lambda x:
len(x.split()))

print("\nText Length Analysis:")
print(df[['title_length', 'description_length']].describe())

# Visualize Text Length Distributions
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.histplot(df['title_length'], bins=30, kde=True)
plt.title('Title Length Distribution')
plt.xlabel('Number of Words')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
sns.histplot(df['description_length'], bins=30, kde=True)
plt.title('Description Length Distribution')
plt.xlabel('Number of Words')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

# Save cleaned data to CSV for review
df[['cleaned_title', 'cleaned_description']].to_csv('cleaned_data_new.csv',
index=False)

print("Text cleaning complete. Cleaned data saved to 'cleaned_data_new.csv'.")
```

```
# Remove rows with one-word titles and empty descriptions
df_filtered = df[(df['cleaned_title'].apply(lambda x: len(x.split())) > 1) &
(df['cleaned_description'].apply(lambda x: len(x.strip()) > 0))].copy()

# Save the filtered data to a new CSV file
df_filtered[['cleaned_title',
'cleaned_description']].to_csv('cleaned_updated.csv', index=False)

print("Filtered data saved to 'cleaned_updated.csv'. Summary statistics after
filtering:")

# Summary Statistics after filtering
print(df_filtered.describe(include='all'))

# Text Length Analysis after filtering
print("\nText Length Analysis after filtering:")
print(df_filtered[['title_length', 'description_length']].describe())

# Proceed with Tokenization
df_filtered['tokenized_title'] =
df_filtered['cleaned_title'].apply(word_tokenize)
df_filtered['tokenized_description'] =
df_filtered['cleaned_description'].apply(word_tokenize)

# Word Count Distribution (Before Stopword Removal)
all_words = df_filtered['tokenized_title'].explode().tolist() +
df_filtered['tokenized_description'].explode().tolist()
word_freq = Counter(all_words)

# Convert to DataFrame for analysis
word_freq_df = pd.DataFrame(word_freq.items(), columns=['word', 'count'])

# Top 20 most common words (Before Stopword Removal)
top_words = word_freq_df.nlargest(20, 'count')

# Visualize the Top 20 Words (Before Stopword Removal)
plt.figure(figsize=(12, 6))
sns.barplot(x='count', y='word', data=top_words)
plt.title('Top 20 Most Common Words (Before Stopword Removal)')
plt.xlabel('Frequency')
plt.ylabel('Words')
plt.show()

# Visualize the distribution of word frequencies (Before Stopword Removal)
plt.figure(figsize=(12, 6))
sns.histplot(word_freq_df['count'], bins=50, kde=True)
plt.title('Word Frequency Distribution (Before Stopword Removal)')
```

```
plt.xlabel('Word Frequency')
plt.ylabel('Count of Words')
plt.yscale('log')
plt.show()

# Stopword Removal using NLTK's stopwords list
stop_words = set(stopwords.words('english'))

df_filtered['tokenized_title'] = df_filtered['tokenized_title'].apply(lambda
x: [word for word in x if word not in stop_words])
df_filtered['tokenized_description'] =
df_filtered['tokenized_description'].apply(lambda x: [word for word in x if
word not in stop_words])

# Word Count Distribution (After Stopword Removal)
all_words = df_filtered['tokenized_title'].explode().tolist() +
df_filtered['tokenized_description'].explode().tolist()
word_freq = Counter(all_words)

# Convert to DataFrame for analysis
word_freq_df = pd.DataFrame(word_freq.items(), columns=['word', 'count'])

# Top 20 most common words (After Stopword Removal)
top_words = word_freq_df.nlargest(20, 'count')

# Visualize the Top 20 Words (After Stopword Removal)
plt.figure(figsize=(12, 6))
sns.barplot(x='count', y='word', data=top_words)
plt.title('Top 20 Most Common Words (After Stopword Removal)')
plt.xlabel('Frequency')
plt.ylabel('Words')
plt.show()

# Visualize the distribution of word frequencies (After Stopword Removal)
plt.figure(figsize=(12, 6))
sns.histplot(word_freq_df['count'], bins=50, kde=True)
plt.title('Word Frequency Distribution (After Stopword Removal)')
plt.xlabel('Word Frequency')
plt.ylabel('Count of Words')
plt.yscale('log')
plt.show()

# Lemmatization
lemmatizer = WordNetLemmatizer()

def lemmatize(tokens):
    lemmatized = [lemmatizer.lemmatize(word) for word in tokens]
    return lemmatized
```

```
df_filtered['processed_title'] =
df_filtered['tokenized_title'].apply(lemmatize)
df_filtered['processed_description'] =
df_filtered['tokenized_description'].apply(lemmatize)

# Save the processed data to a final CSV file
df_filtered[['processed_title',
'processed_description']].to_csv('preprocessed_data.csv', index=False)

print("Data preprocessing complete. Processed titles and descriptions saved to
'preprocessed_data.csv'.")

# Combine titles and descriptions for embedding
df_filtered['combined_text'] = df_filtered['processed_title'].apply(' '.join)
+ ' ' + df_filtered['processed_description'].apply(' '.join)

# K-Fold Cross-Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

silhouette_scores = []
davies_bouldin_scores = []
fold = 1

for train_index, test_index in kf.split(df_filtered):
    train_data = df_filtered.iloc[train_index]
    test_data = df_filtered.iloc[test_index]

    # Generate Embeddings with BERTopic
    bertopic_model = BERTopic()
    train_embeddings =
bertopic_model.fit_transform(train_data['combined_text'].tolist())[1]  # Get
the embeddings
    test_embeddings =
bertopic_model.transform(test_data['combined_text'].tolist())[1]

    # Ensure embeddings are 2D arrays
    if train_embeddings.ndim == 1:
        train_embeddings = np.expand_dims(train_embeddings, axis=1)
    if test_embeddings.ndim == 1:
        test_embeddings = np.expand_dims(test_embeddings, axis=1)

    # Cluster Embeddings with KMeans
    n_clusters = 5
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(train_embeddings)

    # Get cluster labels for both train and test data
```

```
    train_labels = kmeans.labels_
    test_labels = kmeans.predict(test_embeddings)

    # Evaluate the Model
    silhouette_avg = silhouette_score(train_embeddings, train_labels)
    davies_bouldin_avg = davies_bouldin_score(train_embeddings, train_labels)

    silhouette_scores.append(silhouette_avg)
    davies_bouldin_scores.append(davies_bouldin_avg)

    print(f"Fold {fold} - Silhouette Score: {silhouette_avg}, Davies-Bouldin
Index: {davies_bouldin_avg}")
    fold += 1

# Average Scores
print(f"Average Silhouette Score across all folds:
{np.mean(silhouette_scores)}")
print(f"Average Davies-Bouldin Index across all folds:
{np.mean(davies_bouldin_scores)}")

# Step 4: Express Cluster Topics in Words
def get_top_words_for_clusters(df, cluster_labels, top_n=10):
    df['cluster'] = cluster_labels
    top_words_per_cluster = {}

    for cluster_num in np.unique(cluster_labels):
        cluster_data = df[df['cluster'] == cluster_num]
        all_words = cluster_data['combined_text'].str.split().explode()
        word_freq = Counter(all_words)
        top_words_per_cluster[cluster_num] =
dict(word_freq.most_common(top_n))

    return top_words_per_cluster

# Use the test data of the last fold for topic extraction
last_fold_test_data = df_filtered.iloc[test_index]  # Get the test data from
the last fold
last_fold_test_labels = test_labels

top_words_per_cluster = get_top_words_for_clusters(last_fold_test_data,
last_fold_test_labels)

# Print and plot top words for each cluster
for cluster_num, words in top_words_per_cluster.items():
    print(f"Cluster {cluster_num}:")
    print(words)
    print()
```

```
# Plot top words for each cluster
for cluster_num, words in top_words_per_cluster.items():
    plt.figure(figsize=(12, 6))
    sns.barplot(x=list(words.values()), y=list(words.keys()))
    plt.title(f'Top Words for Cluster {cluster_num}')
    plt.xlabel('Frequency')
    plt.ylabel('Words')
    plt.show()

# Apply BERTopic for Topic Modeling
bertopic_model_only = BERTopic()
topics, probabilities =
bertopic_model_only.fit_transform(df_filtered['combined_text'].tolist())

# Evaluate the BERTopic model using silhouette and Davies-Bouldin scores

# Extract the embeddings from the BERTopic model
bertopic_embeddings_only =
bertopic_model_only.transform(df_filtered['combined_text'].tolist())[1]

# Ensure the embeddings are 2D arrays
if bertopic_embeddings_only.ndim == 1:
    bertopic_embeddings_only = np.expand_dims(bertopic_embeddings_only,
axis=1)

# Use KMeans labels as the clustering criterion
kmeans_bertopic_only = KMeans(n_clusters=n_clusters, random_state=42)
kmeans_bertopic_only.fit(bertopic_embeddings_only)
bertopic_labels_only = kmeans_bertopic_only.labels_

# Calculate Silhouette Score
silhouette_score_bertopic_only = silhouette_score(bertopic_embeddings_only,
bertopic_labels_only)

# Calculate Davies-Bouldin Index
davies_bouldin_score_bertopic_only =
davies_bouldin_score(bertopic_embeddings_only, bertopic_labels_only)

print(f"BERTopic Only - Silhouette Score: {silhouette_score_bertopic_only}")
print(f"BERTopic Only - Davies-Bouldin Index:
{davies_bouldin_score_bertopic_only}")

# Compare scores with the BERTopic + KMeans model
average_silhouette_kmeans = np.mean(silhouette_scores)
average_davies_bouldin_kmeans = np.mean(davies_bouldin_scores)

print(f"Comparison of Models:")
```

```
print(f"Average Silhouette Score for BERTopic + KMeans:
{average_silhouette_kmeans}")
print(f"Silhouette Score for BERTopic Only: {silhouette_score_bertopic_only}")

print(f"Average Davies-Bouldin Index for BERTopic + KMeans:
{average_davies_bouldin_kmeans}")
print(f"Davies-Bouldin Index for BERTopic Only:
{davies_bouldin_score_bertopic_only}")

# Extract and display the top words for each topic in the BERTopic-only model
# Get the topics and their top words
top_words_per_topic_bertopic_only = bertopic_model_only.get_topics()

# Print and plot top words for each topic
for topic_num, words in top_words_per_topic_bertopic_only.items():
    if topic_num == -1:
        continue  # Skip the outlier topic (-1)

    print(f"Topic {topic_num}:")
    print(dict(words))
    print()

    # Plot the top words for each topic
    plt.figure(figsize=(12, 6))
    sns.barplot(x=[freq for _, freq in words], y=[word for word, _ in words])
    plt.title(f'Top Words for Topic {topic_num} (BERTopic Only)')
    plt.xlabel('Frequency')
    plt.ylabel('Words')
    plt.show()
```