Cybersecurity Attacks: A Statistical Analysis of Descriptions to Determine Severity of the Vulnerabilities

Vrushal Arvind Bhalerao
201675724

Dr. Seppo Virtanen

Submitted in accordance with the requirements for the
module MATH5872M: Dissertation in Data Science and Analitics
as part of the degree of

Master of Science in Data Science and Analystics

The University of Leeds, School of Mathematics

September 2023

School of Mathematics
FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
UNIVERSITY OF LEEDS

Academic integrity statement

You must sign this (typing in your details is acceptable) and include it with each piece of work you submit.

I am aware that the University defines plagiarism as presenting someone else's work, in whole or in part, as your own. Work means any intellectual output, and typically includes text, data, images, sound or performance.

I promise that in the attached submission I have not presented anyone else's work, in whole or in part, as my own and I have not colluded with others in the preparation of this work. Where I have taken advantage of the work of others, I have given full acknowledgement. I have not resubmitted my own work or part thereof without specific written permission to do so from the University staff concerned when any of this work has been or is being submitted for marks or credits even if in a different module or for a different qualification or completed prior to entry to the University. I have read and understood the University's published rules on plagiarism and also any more detailed rules specified at School or module level. I know that if I commit plagiarism I can be expelled from the University and that it is my responsibility to be aware of the University's regulations on plagiarism and their importance.

I re-confirm my consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to monitor breaches of regulations, to verify whether my work contains plagiarised material, and for quality assurance purposes.

I confirm that I have declared all mitigating circumstances that may be relevant to the assessment of this piece of work and that I wish to have taken into account. I am aware of the University's policy on mitigation and the School's procedures for the submission of statements and evidence of mitigation. I am aware of the penalties imposed for the late submission of coursework.

Name Vrushal Arvind Bhalerao

Student ID 201675724

Abstract

In this modern age of big data, where technology has been advancing constantly, making more data available every day than we know what to do with it. But we still have to ensure that the data does not fall into the wrong hands and get misused. Although a lot of security measures have been developed to safeguard one's data, there are still vulnerabilities in the system that get exploited more than once in a while. This exploitation of vulnerabilities has been rising throughout the years, one can say that crime has also evolved along with the technology. The National Vulnerability Database, operated by the National Institute of Standards and Technology, is a vast repository of data that collects information about vulnerabilities from various sources and is used by researchers as a source of data for security-related studies, including vulnerability analysis, trends, and patterns. Using this invaluable data, the primary objective of this research is to predict the severity of a vulnerability along with the keyword associated with the severity. This dual approach will be performed using the natural language processing topic modelling method Latent Dirichlet Allocation, which will be used as an input to several machine learning techniques such as; logistic regression, decision tree, random forest and support vector machines, and a type of recurrent neural network, long short-term memory. The use of these diverse tools will help us understand how each model is affected by the topics present in the descriptions of the vulnerability. The use of data science in the field of cybersecurity will not only fortify the management of vulnerabilities but also strengthen the system and network resilience in an ever-evolving technological landscape.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In our world were technology has been advancing rapidly, where data is the lifeblood of businesses and individuals, the importance of safeguarding the data from falling into the wrong hands cannot be overstated. As the interconnected of systems increases so do the chances of the data getting exploited. Cybersecurity attacks which were once considered rare occurrences have now become much more common, targeting organizations and individuals alike. In this day and age, the consequences of a successful cybersecurity attack can be catastrophic, ranging from a substantial financial losses and data breaches to destroyed network and reputational damage.

One of the primary objective of the cybersecurity domain is the identification and amending the vulnerabilities within computer systems and networks, which can be exploited to perform a cybersecurity. Vulnerabilities are like cracks in the systems that get exploited by malicious attackers. Identifying these vulnerabilities before they can be leveraged by attackers is a critical preventive measure in safeguarding sensitive information and maintaining integrity of systems.

National Vulnerability Databse is a U.S repository were vulnerabilities are recorded. The primary objective of this research is to utilize the data available on the National Vulnerabilities Database to predict the severity levels of the vulnerabilities. In the ever-evolving world of technology, accessing and understanding the potential impact of vulnerabilities is of great importance. By employing natural language processing techniques mainly Latent Dirichlet Allocation (LDA) along with machine learning and deep learning techniques, this research aims to make models that classify vulnerabilities into different severity levels based on the keywords or descriptions. Furthermore, given that keywords or descriptions are to be used for the predictions, this research also aims to generate keywords based on the severity level predicted by the model, offering an understanding of the nature of the vulnerability. This dual approach aims to enhance the efficiency of vulnerability management, ultimately contributing to cybersecurity practices.

The analysis is mainly based on the descriptions of the vulnerability, to accomplish this, a type of topic model method known as Latent Dirichlet Allocation will be used. The aim of this model is to find the underlying themes that are present in the text, which will be particularly useful to predict the severity as every severity will have a different textual theme. The changes due to the different severity levels can be found by using this topic model on the data for each individual severity level, additionally, this same approach can be used to see the changes in different years in the counts and severity of the vulnerability. This model was validated using the metrics coherence score and perplexity, where coherence explains how often the terms co-occur together which is more useful for

interoperability and perplexity explains how well the model describes the texts. Another topic model called the Hierarchical Dirichlet Process (HDP) was used to determine the optimal number of topics for the LDA model and t-Distributed Stochastic Neighbour Embedding (t-SNE) was used to visualize the topics generated by the LDA model, in a 2-D format.

To perform the analysis, machine learning models such as logistic regression with L1, L2 and L1-L2 penalty, decision tree, random forest and support vector machines, and a type of recurrent neural network, Long Short-Term Memory (LSTM) were all integrated with the LDA model. These models take the topic distributions generated by the LDA as the input to train the model. In addition to predicting the severity, the LSTM network also generates a keyword that the model thinks is associated with the severity, thus helping us understand the nature of the vulnerability.

The ability to predict the severity of a vulnerability and generate relevant keywords is not merely an academic pursuit; it is a proactive defense mechanism in the ongoing battle against cybersecurity attacks. By understanding the potential scope and impact of the vulnerability, security experts can prioritize their resources and address the most critical issues first.

In summary, this research stands at the intersection of datascience and cybersecurity, with the goal of fortifying our networks. Vulnerability prediction along with keyword generation, security experts can stay one step ahead of potential attacks by identifying potential vulnerabilities. This predictive approach is particularly crucial in this day and age as new vulnerabilities are discovered regularly, and attackers are quick to exploit them.

Chapter 2

Background

2.1 Aim

Within the broader concept of cybersecurity, this research focuses on the vulnerabilities, the cracks in the system that get exploited by attackers. Vulnerabilites have real world consequences when exploited, and are classified into different severity. This data is available of the National Vulnerabilities Database, and each vulnerability comes with a description, describing the said vulnerability. The primary objective of this research is the predict the severity of a vulnerability based on the description. This can be achieved using machine learning and deep learning techniques along with natural language processing techniques. Additionally, using deep deep learning techniques along with natural language processing techniques will also allow the generation of keyword for the predicted severity, thus highlighting the nature of the vulnerability.

This section will explain the data source National Vulnerabilities Database and the data obtained from this data source which will be utilized in this research. This section will also discuss the relevant papers.

2.2 Data

The National Vulnerabilities Database (NVD) is the U.S. government repository of standards-based vulnerability management data [1]. Originally created in 1999 (called the Internet - Categorization of Attacks Toolkit or ICAT), the NVD has undergone multiple iterations and improvements and will continue to do so in order to deliver its services. NVD performs analysis on Common Vulnerabilities and Exposures (CVEs) which have been published in the CVE dictionary. NVD does not actually analyse these CVEs but relies on vendors, third-party security researchers and vulnerability coordinators to provide information that is then used to assign these attributes, these third-party organisations are called CVEs Numbering Authorities (CNAs). CVE-ID is allocated to a CVE once it is considered to be a vulnerability, it is assigned by the CNAs. Basically, the CNAs are given possible candidates of CVEs, based on their scope and ability to timely vet each one. Once a reporter (typically the original individual or organization(s) that discovered the bug) contacts the CVE Assignment Team or an appropriate CNA to request a CVE ID.

Each CVE must include a description that is either provided by the reporter. This description

Variable Name	Description	Type
CVE-ID	The ID assigned to a common vulernability and exposure after it has been confirmer to be a vulnerability	Object
Description	The description provided by the vendor for the vulernability	Object
Base Severity V3	The severity assigned to the vulernability based on the V3 metric	Object
Published Date	The date the vulernability was published	Object

Table 2.1: Table Explaining the Variables in the Data

includes the type of vulnerability, the product's vendor, and the affected code base(s). Reporters can provide further information, such as the expected impact, attack vectors, or state of remediation. Once the vetting process is completed, a CVE ID is assigned.

RESERVED tags are used when CVE IDs have been assigned or potentially assigned to vulnerabilities which need further details before they can be finalized. Should the vulnerability be unsuitable for publication, it will be denied a CVE ID and tagged REJECTED by the CNA. This may occur due to a lack of qualifying factors, irregularities in the reporting process, or a request to be withdrawn by the original reporter.

A CVE ID also may be given a DISPUTED tag should the vendor or other authoritative entity challenge the validity of the vulnerability. This can occur before or after the National Vulnerability Database publishes their analysis. The website has data feeds for all the CVEs which are available in various formats, but for this research paper the data feeds in JSON format were taken. There is data available for CVEs of each year. The JSON files contain the CVE-IDs, the severity of the attack, the description and the time stamp when the CVE was published.

The severity of a CVE is calculated using Common Vulnerability Scoring System (CVSS). There are two metrics to calculate the severity of the CVE, CVSS V2 and CVSS V3, for this analysis, the V3 metric will be considered as V2 was no longer used since 12th July 2022.

The data consists of various variables like CVE-ID, CVE-Problemtype, DatePublished, DateModified, etc. only a select few variables will be used for the purpose of this analysis. The table 2.1 explains the variables that are going to be used in the analysis of the data.

2.3 Literature Review

This section will discuss all the papers that have influenced this research, introduced the primary tool of this research and utilized similar techniques used in this research. These papers have contributed to the theoretical framework, methodologies, and tools that underpin this research, making it essential to explore their contributions.

The foundation of this research is based on the work of David Blei, Andrew Ng, and Michael Jordan, presented in their paper "Latent Dirichlet Allocation" (LDA) (2003) [2]. This paper is recognised as the original source for LDA, the LDA model was introduced to address the challenges in the traditional topic modelling techniques. LDA is an unsupervised learning technique and does not require manual selection of topics, unlike the techniques previous used in topic modelling. It also addressed the problem of overfitting on large data by introducing a probabilistic generative model that helped regularize topic assignments. This made it a much more suitable model for any topic-modelling technique and made it a cornerstone in the field of natural language processing. While the primary focus of this research is the use of LDA, this paper served as a guiding reference for understanding the theoretical and core concepts of LDA, making it much easier to integrate with several models.

The integration of the Recurrent Neural Network (RNN) model LSTM and topic models was first proposed in the paper titled ”TopicRNN: A Recurrent Neural Network with Long-Range Semantic Dependency” [3]. The aim of this integration was to cover the shortcomings of both LSTM and topic models. LSTM is good for capturing local word structures, but cannot handle long-range dependencies, on the other hand topic models are able to capture the global structure of the document but do not account for the ordering of the words. TopicRNNs were designed to capture both local and long-range semantic dependencies in sequential data, making them particularly suitable for tasks such as language modelling. For this analysis, a similar approach was utilized by combining LDA and LSTM, but this model was utilized to predict the severity along with keyword generation, demonstrating the versatility of a hybrid LDA-LSTM model.

A paper in 2020 titled ”A Hybrid LDA-LSTM Model for Understanding Health and Medical Text” [4], presented the approach of integrating LDA with LSTM to analyze health and medical text data. This paper aimed to harness the power of user comments and reviews through advanced topic modelling and neural network techniques, ultimately enhancing the quality of recommendations and improving our understanding of user-generated data in the context of crowdfunding campaigns. Unlike this research, this model takes multiple inputs, including words, topic embeddings, temporal details, and probability distributions from Latent Dirichlet Allocation (LDA) to make a personalized recommendation based on the topics discovered.

Chapter 3

Methodology

The methodology for this research is designed to explore the domain of cybersecurity, specifically to predict the severity of the vulnerabilities based on their descriptions. To perform the said task, the approach in this research utilizes techniques such as Latent Dirichlet Allocation along with various classification models and an RNN model.The overarching goal of this research is to unearth concealed patterns and insights hidden within the data. Along with predicting the severity of a vulnerability, the use of RNN model LSTM will also generate keywords for the said vulnerability. This chapter endeavours to bridge the gap between the theoretical framework and practical application along with discussing the limitation faced by this research.

3.1 Latent Dirichlet Allocation (LDA)

LDA [2] is a generative statistical model used for topic modelling. Topic modelling is a type of statistical model used in natural language processing to discover topics from a document. The goal of topic modelling is to uncover underlying semantic structures in a large textual corpus, it does so by taking a collection of words that frequently co-occur together. LDA addressed several challenges in traditional topic modelling like scalability, interpretability, flexibility; and the handling of complex linguistic phenomena like polysemy. Another advantage of LDA is that it is an unsupervised classification technique, which means that it needs labelled data. It is based on the principle that each document is made up of a mixture of topics, which in turn are defined by a mixture of words. The word Latent refers to the hidden structures (topics) in the document, while Dirichlet refers to the distribution that is used to model the variability about the distribution of topics in documents as well as the distribution of words in a topic and Allocations is assigned the topics to the documents. So, it models documents as a mixture of latent topics. Documents are a probability distribution over latent topics and a document was created by selecting words from a mixture of topics and topics are probability distribution over words.

3.1.1 Generative Process

The algorithm assumes that each document is generated through the following process:

- Count the number of words in a document.

- Chose a topic mixture (θ) for the document (d) using Dirichlet distribution, parameterized by (α), i.e., $\Theta \sim \text{Diri}(\alpha)$.

- For each word in the document:

  - Choose a topic (z) from the topic mixture (θ).
  - Choose a word (w) from the topic's (z) word distribution (ϕ), which is also drawn from a Dirichlet distribution, parameterized by (β), $\Phi \sim \text{Diri}(\beta)$.
  - Add word (w) to this document (d).

3.1.2 Joint Distribution

The joint distribution of a topic mixture θ, a set of topics Z, and a set of N words W is given by:

$$p(\theta, Z, W|\alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^{N} p(z_n|\theta)p(w_n|z_n, \beta)$$

3.1.3 Components of Joint Distribution

- $p(\theta|\alpha)$ is the Dirichlet distribution for the topic mixture θ parameterized by α.

- $p(z_n|\theta)$ is the probability of choosing a topic z_n from the topic mixture θ.

- $p(w_n|z_n, \beta)$ is the probability of choosing a word w_n from the topic z_n, parameterized by β.

3.1.4 Inference Methods

The primary aim of LDA is to reverse this generative process to infer hidden topic structures in a document. This is also known as the inference step. In this, its objective is to estimate the posterior distribution of the variables in a document. There are methods that LDA uses for inference:

- Gibbs Sampling: This is a Markov Chain Monte Carlo (MCMC) method, it constructs a Markov chain and uses its stationary distribution to approximate the posterior distribution of the topics. This method goes through each variable or set of variables in the model and samples a new value, conditioned on the current values of all the other variables, It does so for each variable until it produces a sample close to the posterior distribution

- Variational Inference: This is an optimization technique that turns the inference problem into an optimization problem. The idea behind this is to simplify the complex posterior distribution with a simpler distribution, and then tweak the simpler distribution to be close to the complex posterior distribution. This method makes use of Jenson's inequality to find an adjustable lower bound on the log-likelihood. It tries to maximize the lower bound to make simpler distributions which are closely approximate to the complex posterior distribution.

3.1.5 Parameter Estimation

After the inference, the next step is to estimate the parameters of the model α (for topics per document) and β (for words per topic). This is generally done using the Variational Expectation-Maximization (EM) algorithm, which is an extension of the traditional EM algorithm adapted to work with the variational approximations of the posterior distributions. In this step, it finds the variational parameters that maximize the lower bound and then using these parameters it maximizes the lower bound with respect to the model parameters α and β.

In summary, LDA is an effective tool which helps us find latent topics from any document and the algorithm's foundation in Bayesian statistics and its use of variational inference makes it particularly useful for handling large and complex databases. Compared to other natural language processing techniques, LDA is easily interpretable and the ability to find topics on each document makes it really suitable for analysis of textual data. Therefore, the use of LDA aligns with the research objective as it will serve as a pivotal tool to uncover the latent thematic tones within the descriptions present in my database. The algorithm's output will help us understand which topics dominate which will in turn help us identify the trends in the data.

3.2 Perplexity

Perplexity is a metric used to evaluate the quality of probabilistic models, like LDA. It was used as a metric in the paper "Latent Dirichlet Allocation" (2003) [2]. It serves as a measure of how well the probability distribution predicted by the model aligns with the actual distribution of the words in the documents. Lower perplexity values indicate better generalization performance on unseen data, thus suggesting a more accurate model. The mathematical formula for perplexity is:

$$\text{Perp}(D) = \exp\left\{-\frac{\sum_{d=1}^{M} \log p(\mathbf{w}_d)}{\sum_{d=1}^{M} N_d}\right\} \tag{3.1}$$

where M is the number of documents, $\mathbf{w}_d$ is the d-th document in the corpus, and N_d is the number of words in document d. The term $p(\mathbf{w}_d)$ is the likelihood of the document $\mathbf{w}_d$ given the model.

Lower perplexity implies that the model is certain about its word predictions, indicating that the quality of the model is good.

In the context of LDA, perplexity is calculated using the inferred topic-word distributions (ϕ) and the document-topic distributions (θ). Specifically, the likelihood $p(w_d)$ for each document is computed based on these distributions, and then the perplexity is calculated using the formula mentioned above.

3.3 Hierarchical Dirichlet Process (HDP)

HDP is a non-parametric extension of LDA, it was introduced in the year 2006 and was published in the "Journal of the American Statistical Association" [5]. Like LDA, it helps us identify the topics in a collection of documents but, unlike LDA, HDP does not need a specific number of topics to be specified in advance so it allows for potentially infinite topics, making it a much more flexible approach. At the core, HDP uses the Dirichlet process to model the uncertainty for the number of

topics. It adds a hierarchical structure on top of the Dirichlet process, such that data is shared across different levels of hierarchy, which is particularly useful for a complex and diverse database where the ideal number of topics is not clear.

It creates a set of global topics which can be used across all the documents, they're all general categories that the words can belong to. It picks up from these global topics and decides which ones are the most relevant for each individual document, so each document gets a mixture of its own topics. Then just like LDA, each word in the document is then assigned to one of the topics. It then utilizes statistical methods, like Gibbs sampling and variational inference, to find the best topics and how they appear in the documents. It adjusts itself such that it finds the most coherent and representative topics for the whole corpus. Like LDA, HDP also has two primary parameters, α and γ. α works the same way as in LDA, that is, it controls the distribution of topics within each document, a higher value of α makes each document contain a mixture of topics whereas a low value of α makes each document focus on fewer topics. γ, which is a parameter unique to HDP, controls the distribution of topics over the entire corpus and it influences the number of topics the model will learn from the data, a higher value of γ will encourage the model to choose a large number of topics, whereas a smaller value will return fewer topics. In summary, HDP serves as a robust and flexible tool as an extension to LDA. Its ability to automatically determine the number of topics serves really well for a complex database. The model's hierarchical structure allows a more nuanced relationship between topics.

In this study a hybrid approach was utilized, where both LDA and HDP were used to achieve a comprehensive topic model, This approach also serves as a safeguard against overfitting and underfitting. Initially, HDP was used to determine the optimal number of topics, which was confirmed using coherence scores and graphs. This helped make a LDA model, where the optimal number of topics had been found through HDP, allowing for a more target and computationally efficient exploration of topic distributions. The combination of these two techniques enhances the robustness of the methodology and also ensures a more distinct understanding of the topics within the entire corpus.

3.4 Coherence scores

Coherence scores are a metric that is used to evaluate the topics generated by topic models. They are based on the co-occurrence of words within topics and across the entire corpus. They measure the degree to which words that co-occur in the corpus make up a topic, a higher coherence score indicates that the words that make up the topic co-occur more frequently thus they are likely to represent the topic. There are various methods used to calculate the coherence score, like UMass, c_v, and c_{npmi}, etc. The coherence score will be calculated from the HDP model, as it does not require a set number of topics thus allowing us to calculate the coherence score for each set of topics. These scores for different topics can be plotted to visualize where the scores peak, the number of topics that have the highest score is considered as the optimal number. Using HDP's ability to discover the latent topics in the corpus we calculate the scores, and we find the optimal number of topics required for an LDA model.

3.5 t-Distributed Stochastic Neighbour Embedding (t-SNE)

t-SNE is a type of machine learning algorithm which is used for dimensionality reduction, it was first introduced in the year 2008 in the "Journal of Machine Learning Research" [6]. It is well-suited for visualization of high-dimensional databases. Its primary objective is to map high-dimensional data points to a lower-dimensional space such that similar points in high-dimensional space remain close to each other in a lower-dimensional space. It utilizes a probabilistic approach to model the similarity between data points. The algorithm uses gradient descent to minimize the cost function to measure the difference between the high-dimensional and low-dimensional distribution of points. t-SNE is used for topic modelling to visualize the distribution of topics and their similarities with each other. By reducing the dimensionality of the topic model's output, it makes it easier to understand the relationship between topics, it is particularly useful for complex models like HDP and LDA.

3.6 Multinomial Logistic Regression

Multinomial Logistic Regression [7] is an extension of binary logistic regression, It is a supervised machine learning algorithm, that has been designed to handle more than two discrete outcomes, thus offering a more flexible approach to a multi-class problem. Its objective is to model the probabilities of multiple classes for the dependent variable. It uses the SoftMax function to generalize the logistic function for multiple classes C.

3.6.1 Formula

The formula for Multinomial Logistic Regression can be represented as:

$$\text{SoftMax}(c) = \frac{e^{w_c \cdot x}}{\sum_{k=1}^{C} e^{w_k \cdot x}}$$

where w_c is the weight vector for a class c and x is the feature vector.

3.6.2 Likelihood Maximization

The model aims to find the weight vectors w_c that maximize the likelihood of the data.

3.6.3 Likelihood Formula

The likelihood formula can be represented as:

$$\text{Likelihood} = \prod_{i=1}^{N} \frac{e^{w_{y_i} \cdot x_i}}{\sum_{k=1}^{C} e^{w_k \cdot x_i}}$$

3.6.4 Regularization

Without regularization, the model focuses only on maximizing the likelihood, which makes it easier to read but prone to overfitting on high-dimensional data. To prevent overfitting on high dimensional data by the model, various regularization techniques are applied:

L1 Penalty (Lasso Regression)

Introduced in the year 1996 in "Journal of the Royal Statistical Society: Series B (Statistical Methodology)" [8], this penalty aims to shrink some of the regression coefficients to zero, performing feature selection. The optimization equation becomes:

$$\min_w \frac{1}{N} \sum_{i=1}^{N} \log(1 + \exp(-y_i w^T x_i)) + \lambda \|w\|_1$$

L2 Penalty (Ridge Regression)

Introduced in the Journal of "Technometrics" in the year 1970 [9], this penalty aims to shrink the regression coefficients towards zero, not zero, which is useful in handling multicollinearity. The optimization equation becomes:

$$\min_w \frac{1}{N} \sum_{i=1}^{N} \log(1 + \exp(-y_i w^T x_i)) + \lambda \|w\|_2^2$$

L1-L2 Penalty (Elastic Net)

This hybrid approach was introduced in the "Journal of the Royal Statistical Society: Series B (Statistical Methodology)", 2005 [10] combines both the L1 and L2 regularization and the optimization equation becomes:

$$\min_w \frac{1}{N} \sum_{i=1}^{N} \log(1 + \exp(-y_i w^T x_i)) + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2$$

The Multinomial Logistic Regression model serves as an excellent statistical technique suited for a multi-classification problem. By utilizing the SoftMax function, it generalizes logistic regression for multiple discrete outcomes which in turn helps us understand the dependent variable's categorical nature. The model without regularization can serve as a baseline to signify the impact of regularization techniques.

In this study, a hybrid approach was employed where LDA was integrated with Multinomial Logistic Regression to predict the categorical variable. With this approach, the aim is to use the feature engineering capabilities of the LDA and the classification capabilities of the Multinomial Logistic Regression model. Initially, LDA is applied to the whole corpus so that each document in the corpus is mapped to a lower-dimensional space defined by the topics identified by LDA, which transforms the original high-dimensional data into a manageable form. Then the topic distributions are combined with other feature extraction techniques like Term Frequency-Inverse Document Frequency (TF-IDF), to create a composite feature set, By combining these two the model can capture both the semantic structure of the text (from LDA) and the importance of individual terms (from TF-IDF). Finally, these combined features are then used as the inputs for the Multinomial Logistic Regression model and this model is then trained to predict the categorical variable.

3.7 Decision Tree

Decision Tree, as explained in "Classification and Regression Trees (CART)" [11], is a supervised machine learning algorithm, which uses a tree-like model of decisions to make decisions or predictions based on the input features. In a multiclassification problem, the tree-like model is used to navigate through a series of tests on the input features, to get to a class label. It starts with a root node, which contains the whole database. It then divides this node based on certain criteria, like information gain or Gini impurity, creating child nodes. At each of the nodes, it selects the feature that maximizes the criterion to get the best split and then uses this feature to make subsets of the data. This process continues until the model reaches a stopping criterion, which can be the maximum tree depth or minimum number of samples per leaf. The terminal nodes are called leaves and contain the final predictions and are determined by a majority vote in the sample in the leaf. To make a prediction for a new data point, the model starts at the root node and travels down the tree until it reaches the terminal node or leaf, the class label for the terminal node is then the prediction for the new data point.

3.7.1 Splitting Criterion

The splitting criterion is calculated using two methods:

Information Gain

The formula for Information Gain is given by:

$$\text{Information Gain} = \text{Entropy(Parent)} - \sum_{i=1}^{n} \frac{|C_i|}{|Parent|} \times \text{Entropy}(C_i)$$

Gini Impurity

The formula for Gini Impurity is given by:

$$\text{Gini Impurity} = 1 - \sum_{i=1}^{n} p_i^2$$

Decision Tree's hierarchal, rule-based structure serves as a robust and flexible model for a multiclassification problem. Like Multinomial Logistic Regression, LDA was integrated with the Decision Tree model where the combined features obtained from the LDA are the input features in the model.

3.8 Random Forest

Random Forest, which was introduced in the "Machine Learning" journal [12], is an ensemble learning method which constructs multiple Decision Trees during training. The final prediction is determined by taking a vote in every tree in the forest for each class and choosing the majority vote. It uses bagging as the fundamental ensemble method, where multiple subsets of the original database are created using random sampling with replacement and each subset is used to make a Decision Tree, all these Decision Trees make up the Random Forest model. To ensure that each individual decision

tree is decorrelated, it randomly selects a subset of features for each split during the construction of the tree. For any given input, each tree in the model makes an independent prediction and the final prediction is determined by taking the majority vote of the final prediction of all the trees in the forest.

3.8.1 Ensemble Prediction

For any given input, each tree in the model makes an independent prediction. The ensemble prediction for a given input in a multiclassification problem is given by:

$$\text{Ensemble Prediction} = \frac{1}{N}\sum_{i=1}^{N}\text{Prediction}_i$$

By utilizing an ensemble of decision trees, Random Forest models have a lower risk of overfitting and the algorithm's ability to handle multiclassification problems makes it suitable for complex problems. Similar to the previous models, this model was also integrated with LDA and takes the combined features of the LDA as an input to train the Decision Trees, thus adding a layer of complexity that would return better predictions.

3.9 Support Vector Machines

SVM, was introduced in the "Machine Learning" Journal [13], is a type of supervised machine learning model. Their primary aim is to find the hyperplane that separates the data into different classes. The model tries to find the hyperplane that maximizes the margin between different classes in the feature space. The margin is defined as the distance between the closest points (support vectors) of different classes to the hyperplane. In a multiclassification problem, SVM uses either a one-vs-one or one-vs-all strategy. In the one-vs-one strategy, SVM is trained for every pair of classes, thus giving $\binom{n}{2}$classifiers for n classes and in the one-vs-all strategy, each class is trained against all the other classes giving n classifiers. Just like the Decision Trees in a Random Forest, each class makes an independent class prediction and the final class prediction is chosen by a majority vote among the classifiers. SVMs employ kernel functions to map the feature space into a high-dimensional space where it becomes linearly separable. Common kernel functions include polynomial, radial basis function (RBF), and sigmoid kernels. The optimization problem here is to find the hyperplane that maximizes the margin between the classes while minimizing the classification error.

$$\text{Optimization Problem: } \min_{w,b,\xi}\frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i$$

subject to the constraints

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

By utilizing strategies like one-vs-one or one-vs-all, SVM extends its binary classification capabilities to effectively handle multiclassification. The algorithm's flexibility in using kernel functions allows for the accommodation of non-linearly separable data, enhancing its applicability across a range of complex classification tasks. Like every machine learning model before, a hybrid approach was employed where LDA was integrated with SVM. The combined features are used to train the model and SVM tries to find the hyperplane that separates the classes in a high-dimensional feature

space. The algorithm uses kernel functions to transform the feature space if the data is not linearly separable.

3.10 Long Short-Term Memory (LSTM)

LSTM was first introduced in the journal of "Neural Computation" [14]. LSTM networks are a type of Recurrent Neural Network (RNN) that are designed to learn long-term dependencies in sequence data, they were first introduced in the year 1997 in the Neural Information Processing Systems (NeurIPS) conference [15]. They are particularly effective for tasks involving time-series data, natural language processing and other sequential data.

3.10.1 Cell State and Gates

The core part of LSTM is the cell state that acts as a memory that the network can read from, write in and update, which allows the network to store and manage information over long sequences effectively. The network uses three types of gates to regulate the flow of information in the cell state:

- Input gate: Regulate what information is stored in the cell state.
- Forget gate: Decide what information is removed from the cell state.
- Output gate: Decide what information to output based on the cell state and the input.

3.10.2 Cell State Update

The LSTM unit updates its cell state C_t and hidden state h_t based on the current input x_t, previous hidden state h_{t-1}, and previous cell state C_{t-1} as follows:

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\
h_t &= o_t \odot \tanh(C_t).
\end{aligned}
$$

3.10.3 Backpropagation Through Time

LSTMs use a special form of backpropagation called backpropagation through time to update the network weights, this allows the network to learn from long sequences without facing the problem of vanishing or exploding gradients.

3.10.4 Multiclassification

For a multiclassification problem, the output network of the LSTM generally uses a SoftMax activation function, where the raw output scores for each class are converted into a probability and the class with the highest probability is chosen as the final prediction. The optimization goal is to minimize a

loss function, typically the categorical cross-entropy loss, concerning the network parameters. This is often done using optimization algorithms like Adam or RMSprop.

The final output y_t for multiclass classification is computed using a softmax activation function as follows:

$$y_t = \text{softmax}(W_{\text{out}} \cdot h_t + b_{\text{out}})$$

LSTM is really efficient in addressing both sequence-based tasks and multiclassification problems. The integration of the softmax in the output layer and the use of advanced optimization techniques further enhance the network's predictability accuracy. Like the machine learning models, LSTM was also integrated with LDA such that the combined features from the LDA are used as the inputs for LSTM. The first stage in this network involves transforming the raw text into a dense vector representation, this is crucial for capturing the semantic overtone of the words. These dense vectors are then processed by an LSTM layer, which captures the temporal and sequential dependencies in the text. Parallelly, an LDA model generates a topic distribution for each document. This distribution captures the thematic structure of the text. The output from the LSTM layer and the LDA topic distribution are fused into a single, combined feature vector. This feature vector's aim is to capture both the sequential and the thematic information in the text. This combined feature vector is processed through fully connected layers to perform the final classification. The network produces multiple outputs, including severity levels and keyword predictions. This fusion provides a richer feature set and improves the accuracy and the inclusion of LDA adds a layer of interpretability that makes it easier to understand the context of the classification.

3.11 Limitations

While the integration of LDA with LSTM and machine learning techniques offers a unique approach to text classification and topic modelling, the study is limited by a lack of a deeper understanding of the subject matter. This could potentially affect the interpretation of results and the identification of relevant features. Another limitation is computational power, as models like LSTM and Random Forest require more computational power, integrating LDA with them further increases the computational resources they need, The study was limited by the available computational power, which has an impact on the optimization of hyperparameters and the overall performance of the models. Due to this restraint, an extensive search for the hyperparameters was not feasible which affects the general predictive power of the models. While LDA adds a layer of interpretability, models like LSTM and Random Forest and Support Vector Machines, are generally considered "black box" models. This poses a challenge to the interpretability of the results. As the models are being utilized to solve a multiclassification problem, which is inherently more complex than binary classification, the limitations of understanding the precision of multiclassification could affect the reliability of the results. A significant limitation of the study was the imbalance of the classes present in the database used for training and testing. This imbalance introduced bias in the model, making it more sensitive to the majority class and less sensitive to the other classes. This skewness in class distribution reduced the accuracy of the models. In summary, while the methods used in this study are robust and flexible there are several challenges due to the limitations faced. These range from a lack of a deeper understanding of the subject matter to computational constraints, algorithmic complexities, and a sig-

nificant data imbalance. Each of these limitations individually or collectively impacts the reliability, generalizability, and interpretability of the study's findings.

Chapter 4

Analysis

4.1 Data Pre-processing

The initial dataframe is in JSON format which contains nested structures that require normalization before any analysis can be performed on them. The "pd.json_normalize" function from the Pandas library is utilized for this purpose. The primary objective is to flatten the nested JSON object into a tabular form. The "CVE_Items" field contains all the data of interest and is normalized to create the initial dataframe. Additionally, the "CVE_data_timestamp" is retained as metadata. The "cve.description.description_data" field contains nested lists that are then flattened into a new dataframe, This dataframe is then concatenated with the initial dataframe and the now redundant "cve.description.description_data" field is dropped. This results in a fully flattened and normalized dataframe, which is then renamed for a better understanding of the data, and can be used for further analysis. This process is done for the data of years 2022, 2021 , 2020 and 2019. The null and duplicated values are checked for the dataframes for all 4 years. There are no duplicate values in any of the dataframes, but there are 732 null values in the 2022 dataframe, 698 in the 2021 dataframe, 1629 in the 2020 dataframe and 1415 in the 2019 dataframe, These null values are dropped as they were all present in the column "BaseSeverity_V3", which indicates that these vulnerabilities where rejected, i.e., they are not considered as vulnerabilities. The next step was to fix the column with the timestamp, which combines the date and time of the publication of the vulnerability in a string of the format ”YYYY-MM-DDTHH:MM:SS” and for the purpose of this research, only the year and month were required. First, the string is split into two components date and time and both of these components are stored in separate lists. Then the list containing the date component is further processed to remove the data and retain only the year and the month, which results in the format “YYYY-MM”. This list is then added as a new column in the dataframe. The final step was to merge the dataframes from each year into a single, unified dataframe. This was done based on the column named "CVE_ID", using the “concat” function of the Pandas library, which ensured that all the values were distinct and allowed for a seamless merging process.

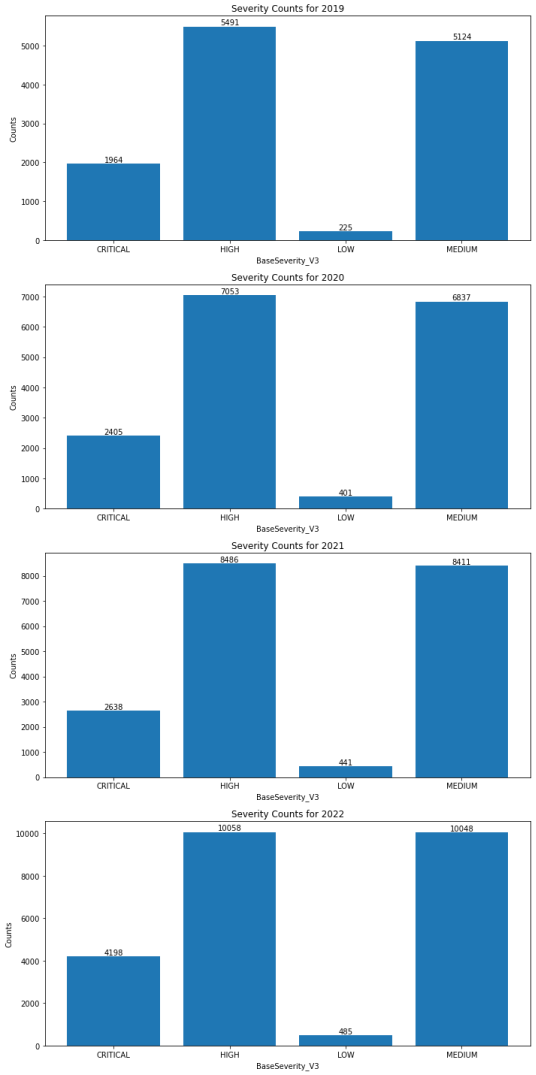

Figure 4.1: Severity Counts for all Four Years

4.2 Exploratory Data Analysis

To gain insights into the severity levels over four years, the bar chart visualization technique was utilized. The data was grouped by each year and then by the severity levels, using the "groupby" method from the Pandas library. Four subplots were created one for each year, using Matplotlib's "subplots" method. Each of these subplots is titled "Severity Counts for 2019", "Severity Counts for 2020", "Severity Counts for 2021" and "Severity Counts for 2020" for each year. The x-axis represents the different severity levels, while the y-axis represents the count for each severity level. Each bar in the chart is labelled with its height value, representing the count of occurrences for each severity level.

As seen in 4.1 the severity level distribution shows a steady increase over the years, although most vulnerabilities each year are of the severity "Medium" and "High" with "Low" being the lowest for each year. We can see that the count has increased substantially from the year 2019 to 2022, where the count has almost doubled, As seen in the graph, the count for the severity level "High" has

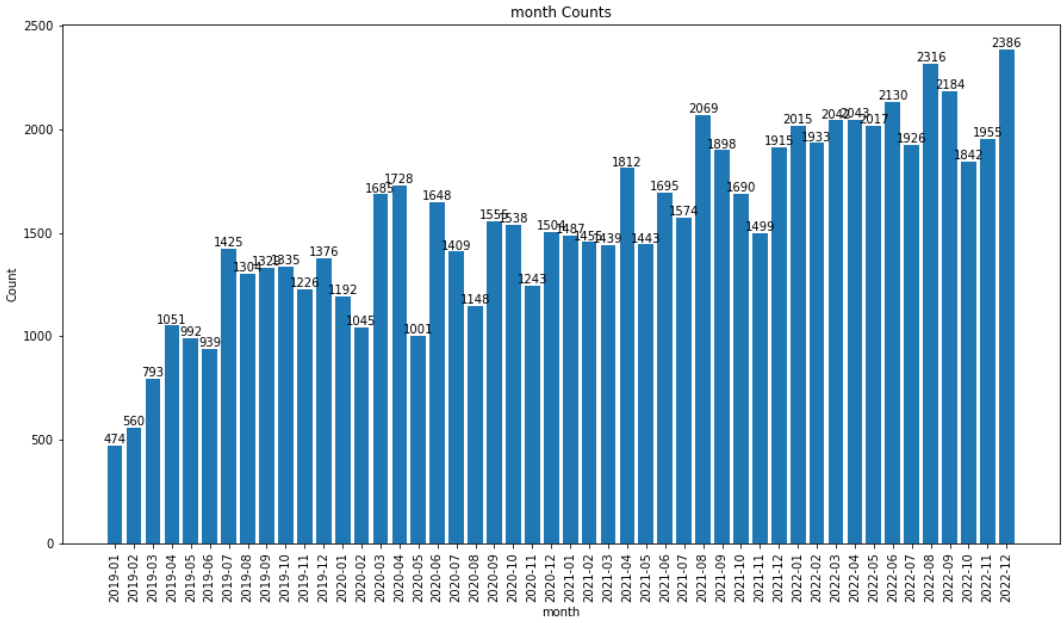

Figure 4.2: Counts in Every Month for all Four Years

gone from 5,491 in the year 2019 to 10,058 in the year 2022. Similarly, the count for the severity levels "Medium", "Critical" and "Low" has gone from 5124, 1,964 and 225 respectively to 10,048, 4,198 and 485. Additionally, to gain insights into the counts of vulnerabilities every month of each year, the bar chart visualization technique was utilized. This graph provides a detailed view of the counts of vulnerabilities every month of each year. The purpose of this graph is to highlight the trends and provide a summary of the data. The x-axis represents the months of the year, while the y-axis represents their counts. Each bar in the chart is labelled with its height value, representing the count of occurrences for each month.

Through the graph in figure 4.2, we can see that there has been a steady increase in the number of vulnerabilities over the years. If we compare the years 2022 and 2019, we can see that the lowest count occurs in the month of January for the year with the count being 474 for 2019 and 1,842 for 2022 in the month of October. On the other hand, the highest count of 1,425 for the year 2019 is in the month of July and the highest count for the year 2022 is 2,386 for the month of December. We can also compare the count of number of attacks for each year to compare how the number of vulnerabilities has increased each year, There were 12,804 vulnerabilities in total for the year 2019, and it rose to 16,696 vulnerabilities in the year 2020, which further increased to 19,976 vulnerabilities in the year 2021 and finally, there were 24,789 vulnerabilities in the year 2022. This suggests that there is a steady and consistent increase in the number of vulnerabilities every year, this also suggests that the number of vulnerabilities will increase as more time passes on.

Now to gain insights into the overall distribution of severity levels for the dataframe which contained the data from all four years, the bar chart visualization technique was utilized. First, the data was grouped by the severity level and then the counts were aggregated. A bar chart is plotted using Matplotlib, where the x-axis represents the different severity levels and the y-axis represents the counts for each severity level.

The figure 4.3, titled "Severity Counts" is a bar graph where each bar in the chart is labelled with

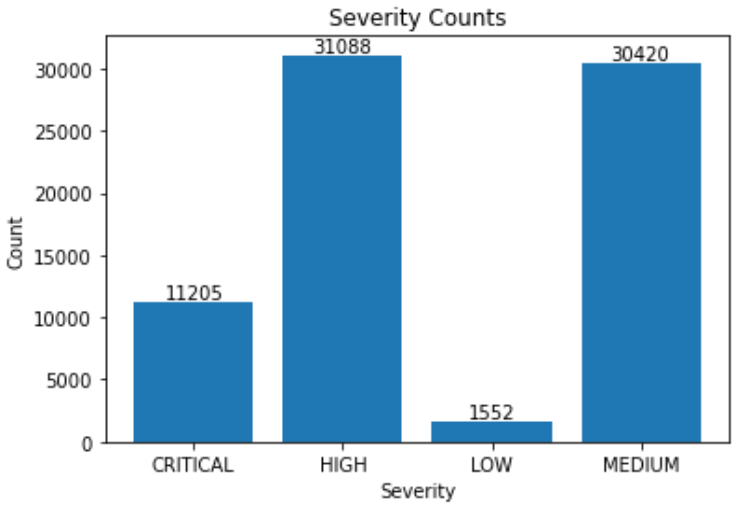

Figure 4.3: Severity Counts all Four Years

its height value, which represents the count for each severity level. The resulting graph shows us that the count for the severity level "High" is the highest with the count being 31,088. The count for severity levels varies significantly across the other categories: the "Medium" severity level has the second highest frequency with 30,420 counts, followed by the "Critical" level with 11,205 instances, and the "Low" severity level has the least frequency, with only 1,552.

The predominance of "High" and "Medium" severity cases indicate that the majority of the vulnerabilities are of high or moderate concern. This implies that the majority of vulnerabilities should not be ignored and most of them need to be addressed as soon as possible. The number of "Critical" cases is also significant and cannot be overlooked, as these vulnerabilities likely require immediate attention to prevent severe consequences. The lesser count of "Low" cases suggests that minor issues occur less frequently or are not reported often. The distribution of severity counts also suggests that the data is imbalanced are certain classes occur more frequently which might skew the analysis of the data.

4.3 Text Pre-Processing

The initial text processing is done using the function "`preprocess_text`", which performs several operations to clean and prepare the textual data for analysis. It performs the following operations:

- Stop word removal: A set of common English stop words is used to filter out all the words that do not contribute anything. Additionally, the word "could" was added to this set as this word was identified to be irrelevant to the topics of interest.
- Case: All the characters are converted to lowercase to ensure uniformity.
- Alphanumeric filtering: Except for alphabets and numbers everything was removed from the text.
- Specific term removal: The word "vulnerability" was removed as it is a common word in the text and does not help in distinguishing between them.

- Tokenization: The cleaned text is tokenized into individual words.
- Frequency-based filtering: A frequency count is performed on tokens so that any token occurring less than 3 times is removed.

This function returns the cleaned text as a string and returns a list of cleaned tokens. This function is applied to the dataframe, thus obtaining a column with the cleaned text and another column with a list of tokens.

Another function called "`preprocess_text_2`" is defined as a variant of the above function, "`preprocess_text`", made specifically to return clean, readable sentences. This function follows steps similar to the previous function, except it does not filter out tokens that occur less than three times, thus making the output more readable. This function is also applied to the dataframe and returns a column with clean, readable text. This column is used for calculating the coherence score, and used in other natural language processing visualization techniques.

By utilizing these pre-processing steps, the text data is cleaned and prepared for the visualization, natural language processing tasks and machine and deep learning techniques that follow.

The final step of pre-processing involved transforming the categorical variable "`BaseSeverity_V3`" into a numerical format. This transformation is crucial because most machine learning algorithms and neural network architectures require numerical input features for model training and prediction. The transformation was carried out using the "`LabelEncoder`" class from the scikit-learn library. The "`LabelEncoder`" class assigns a unique integer to each category in the "`BaseSeverity_V3`" column. Specifically, the encoder sorts the unique categories in the column and assigns an integer label starting from 0 to $N - 1$, where N is the number of unique categories. In this case, the "`BaseSeverity_V3`" column contains four unique categories: "`Critical,`" "`High,`" "`Low,`" and "`Medium,`" which are encoded into 0, 1, 2, and 3, respectively. These resulting numerical labels are stored in a new column called "`label`" in the dataframe. This label-encoded column serves as the target variable for the machine learning classification models and the LSTM model, enabling these algorithms to perform tasks like training, validation, and prediction on the dataset.

4.4 Textual Analysis

In the initial phase of the textual analysis, the entire dataframe was analysed, which serves as a baseline for understanding the general themes and patterns present in the data. Subsequently, the data was grouped into two distinct groups based on the two variables: Severity Levels and Year of Publication. This approach allowed for an understanding of how the textual data varied in severity and how it evolved over time. This approach was not utilized for the machine and deep learning techniques, to ensure a robust and comprehensive modelling approach the analysis was performed on the whole dataframe.

4.4.1 The Entire Dataframe

The first step in the analysis is the application of LDA on the textual data. This will serve as a foundational pillar of the analysis and is the main tool used in the analysis. Using this will allow us to uncover the topics hidden in the corpus and the underlying thematic structure of the text.

Two functions were defined to perform Latent Dirichlet Allocation (LDA), each with its own advantages. The first function, "LDA_gen," is designed to execute LDA on a given corpus of documents and accepts four parameters:

- "desc": the corpus of text documents.
- "num": the number of topics to generate.
- "num_words": the number of words to generate for each topic.
- "vectorizer_type": The type of vectorizer to be used, either "count" for CountVectorizer or "tfidf" for TfidfVectorizer (the default is "count").

This function begins by selecting the appropriate vectorizer and constructs a document-term matrix based on the vectorizer. The document-term matrix is then converted to a Gensim corpus format, and LDA is performed using the Gensim library. The function returns the LDA model, the terms most likely to appear in each topic, the corpus in Gensim format and the mapping from term IDs to terms.

"LDA_gen_2" is a function similar to the previous function, but it is optimized for larger datasets. It accepts the same parameters as the previous function, but instead of using a vectorizer, it directly tokenizes the text and constructs a Gensim Dictionary and corpus. This makes it more memory-efficient and faster for larger datasets.

These two functions are applied to the column containing the cleaned descriptions of the dataframe. Upon applying these functions, a set of terms associated with the 10 topics are generated, these terms try to provide the essence of the topics.

From the output from "LDA_gen", Topic 0 is characterized by terms like ”libreoffice,” ”outside,” and ”technology,” with respective weights of 0.074, 0.042, and 0.036. This suggests that the topic might be related to open-source office software and its external technologies. Topic 1, on the other hand, is heavily weighted towards ”windows” and ”user,” with weights of 0.285 and 0.148, respectively, possibly indicating a focus on user interactions within Windows operating systems. Topic 2 is particularly interesting, with ”attacker” and ”affected” carrying significant weights of 0.221 and 0.078. This topic seems to be centred around security vulnerabilities and potential threats. Topic 4 is dominated by the term ”snapdragon” with a weight of 0.244, followed by ”file” and ”sd,” which could imply a focus on mobile hardware specifications. Topic 8 is overwhelmingly characterized by the term ”earlier,” with a weight of 0.344. This could indicate a focus on version histories or temporal aspects of the subject matter, especially when considered alongside other terms like ”oracle” and ”access.”

Now from the output of "LDA_gen_2", Topic 0 is dominated by terms such as ”snapdragon” and ”sd,” with weights of 0.236 and 0.183, respectively. This suggests that the topic may be focused on hardware specifications, particularly those related to mobile devices, which is similar to Topic 4 from the previous model. Topic 1, on the other hand, is characterized by the term ”prior” with a weight of 0.329, followed by ”user” and ”magento,” indicating a possible focus on versioning or updates in cloud-based services or e-commerce platforms. Topic 2 is particularly noteworthy, with ”attacker” and ”affected” having significant weights of 0.255 and 0.099, respectively, which is similar to topic 2 from the previous model the only difference being the weight of the terms. Topic 3 is overwhelmingly characterized by the term ”versions,” with a weight of 0.366, suggesting a focus on software or firmware versions across different applications or devices. Topic 8 stands out with the

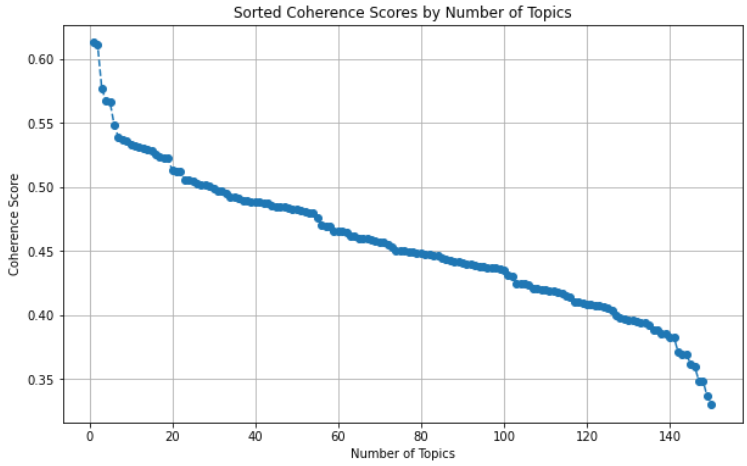

Figure 4.4: sorted Coherence Graph for the HDP model

term ”earlier” having a weight of 0.409, which similar to topic 8 of the previous could indicate a focus on version histories or temporal aspects, especially when considered alongside other terms like ”java” and ”se.”

Both models identify overlapping themes but with varying degrees of emphasis. For example, the term “snapdragon” appears in both the models but is associated with different terms, which suggests that the models capture the core topics, but offer different perspectives on their contextual relationship. `"LDA_gen"` produces terms that are more specific (”libreoffice”, ”tibco”, ”cvss”), whereas `"LDA_gen_2"` produces more general terms (”data”, ”user”, ”issue”), this is due to the use of different vectorization techniques which affect the granularity of the topics. Moreover, `"LDA_gen_2"` has a lower perplexity score of 68.4283 as compared to `"LDA_gen"` whose score is 473.8953, which suggests that `"LDA_gen_2"` is much better suited for this dataframe.

After the application of LDA, HDP was applied to this data to calculate the coherence scores. The goal here is to find the optimal number of topics by plotting a graph for the coherence scores and identifying the elbow, which indicates a drop in the coherence score. The HDP model was initialized using the corpus (`"lda_corpus_dec"`) and the dictionary (`"lda_gensim_dict"`) generated from the preceding LDA model. The model inferred a total of 150 topics. This feature of HDP to infer the topics without pre-defining the number of topics makes it suitable to use when the aim is to find the optimal number of topics.

To validate the optimal number of topics for the LDA model, the coherence scores were calculated for the 150 topics and plotted in descending order. The x-axis represents the number of topics, while the y-axis represents the coherence score. From the graph in figure 4.4, we can see that there is an elbow at 10 topics, beyond which the coherence scores begin to decline. This serves as an indicator for the optimal number of topics for the LDA, as this decline suggests that additional topics may introduce noise. Based on this evidence, 10 was chosen as the optimal number of topics for the analysis of the dataframe.

The next step was to identify the dominant topic for each document. The function ”`format_topic_sentences`” was made for this purpose, this function operates by iterating through each document in the corpus and identifying the topic that has the highest contribution to that document, effectively labelling it as the ’dominant topic’. It then creates a new dataframe, which contains the original text along with the

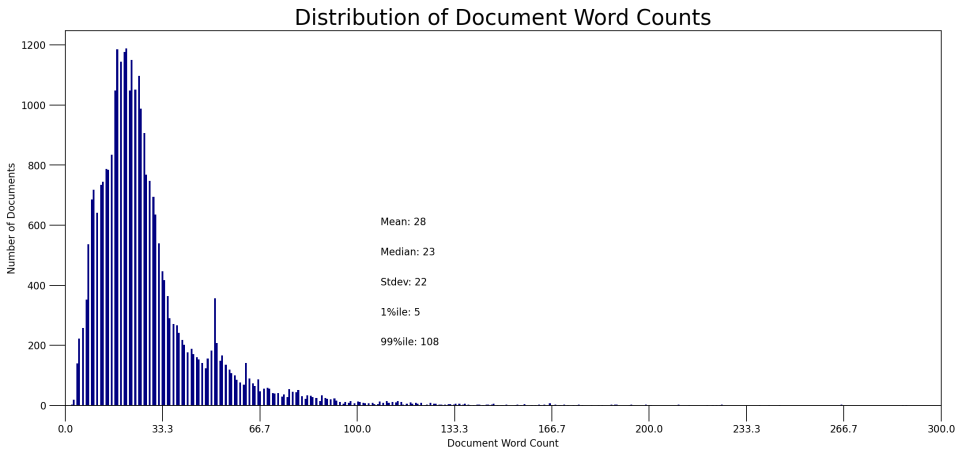

Figure 4.5: Distribution of Document Word Counts

identifier for its dominant topic. This dataframe serves as the foundational structure for plotting the distribution of document word counts both globally and by dominant topic.

From the graph in figure 4.5 focuses on the global distribution of document word counts across the entire corpus. First, the document lengths are calculated by splitting each document into individual words and counting them. Various statistical measures such as the mean, median, standard deviation, and the 1st and 99th percentiles were calculated using the document lengths. The x-axis represents the document word count and the y-axis represents the number of documents. These measures provide a comprehensive understanding of the central tendency, spread, and outliers in the distribution of document lengths. This plot serves as an initial exploration of the corpus, offering insights into its complexity and variability. From the graph we can see that the mean is 28, that is the average length of a description is 28 words. Although the median, which is less sensitive to outliers, is slightly lower which suggests that a few documents have a really high word count. A standard deviation of 23 signifies a moderate level of dispersion around the mean. In other words, while most documents hover around the average length, there is a substantial number that deviates significantly, either being much shorter or much longer. This is further corroborated by the 1st and 99th percentiles, which stand at 5 and 111, respectively. This wide range highlights the diversity in the lengths of documents within the corpus.

The second set of plots provides a more granular view by showing the distribution of document word counts for each dominant topic. First, the lengths of the documents for each dominant topic were calculated and various statistical measures such as the mean, median, standard deviation, and the 1st and 99th percentiles were calculated using these document lengths. The x-axis represents the document word count, while the y-axis shows the number of documents. Additionally, a Gaussian Kernel Density Estimation (KDE) curve is plotted in black to provide a smoothed representation of the distribution. These plots allow for a topic-specific analysis of document complexity, aiding in the interpretation of each topic's scope and depth within the corpus. From the graphs in figure 4.6, we can see that all the topics have a similar distribution to the global distribution of word counts. Topics 0, 2, 3, 6 and 5 have almost the same mean and median value which suggests that most of the documents have a similar word count. Topics 4,6 and 9 have a significantly higher standard deviation, signifying a higher level of dispersion around the mean. This highlights the diversity in the lengths

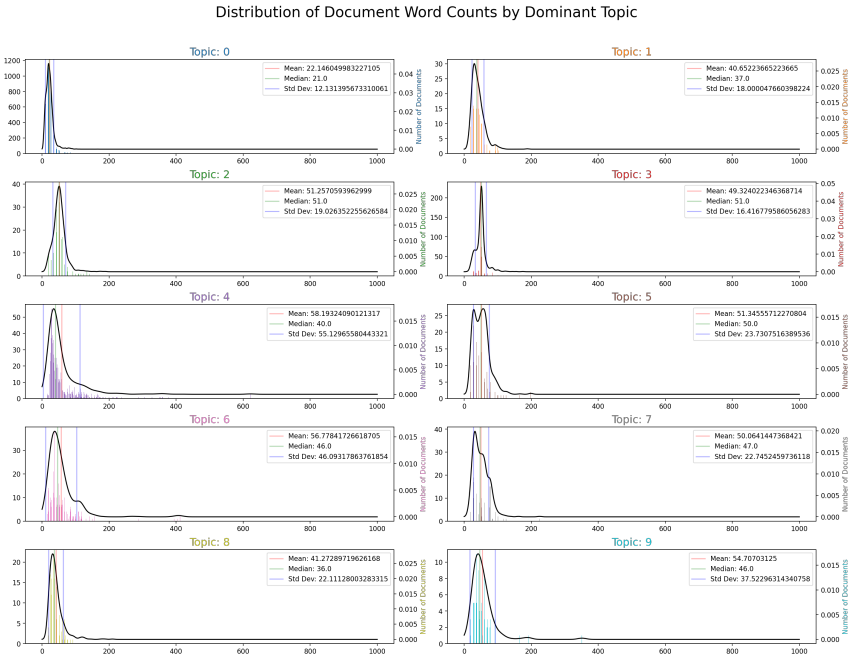

Figure 4.6: Distribution of Document Word Counts by Dominant Topic

of documents for different topics

The next visualization is a series of subplots based on the topics identified by the LDA model. The aim of this plot is to signify the importance of the keywords for each topic. The importance of each keyword is calculated from the model, which is the weight assigned to the word by the model and the word count is derived from the frequency of each keyword in the entire corpus. The graph provides the visualization for the word count and keyword importance for each topic. The x-axis represents the keywords and the y-axis represents the word count and the importance. The word counts bars are plotted on top of the importance bars of the keywords, this provides an understanding of the relevance and the prevalence of the keywords.

From the graphs in figure 4.7, we can interpret that the keywords that appear the most are not always considered as important according to the LDA model. This suggests that while the keyword is frequent in the corpus, it might not be indicative of the specific topic. For example, in Topic 1, the word count for the keyword “attacker” is high but the importance of the keyword “image” is high even though it has a significantly lower word count.

In the final step of the textual analysis, we will perform a t-sne model on the data to visualize the topics generated by the LDA model. To accomplish this, first, we will calculate the topic weights for each document filter those with a dominant topic weight greater than 0.35, and store that in an array. Then we will apply the t-sne model to this array of topic weights, which will reduce the dimensionality of the data while preserving the pair-wise distance between high dimensional data points, resulting in a 2D representation of the topics.

As we can see from the graph in figure 4.8, the data has been properly divided into 10 topics, each with its own cluster. Some clusters are overlapping suggesting that there are some topics that the model cannot distinguish between.

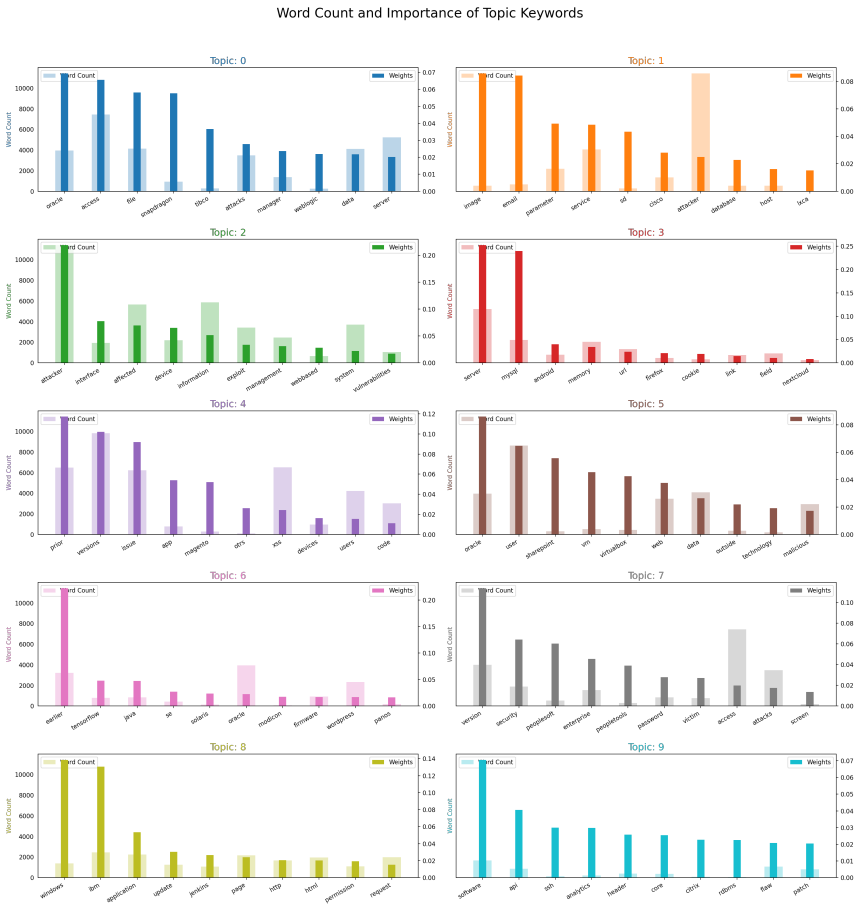

Figure 4.7: Word Count and Importance of Topic Keywords

4.4.2 Segmented Dataframe

To discover the effect of the different severity levels and the years on the textual themes, the data will be grouped by the severity and the years. The techniques used in the textual analysis of the combined dataframe will be applied to each group to understand how the themes vary over severity and time. As these are a subset of the much bigger combined dataframe, the size of these dataframes gets significantly smaller, therefore the function `\LDA_gen"` will be applied to this data, to provide more accurate results.

Severity

Given that there are four severity levels, the textual analysis will be performed on all four levels to understand how the themes vary for the levels Critical, High, Low and Medium.

Critical:

The first step, as before, is the application of LDA. The output consists of 10 distinct topics. Topic 0 is predominantly defined by the term ”earlier,” which has a weight of 0.533. This is followed by terms such as ”application,” ”docker,” and ”system,” with respective weights of 0.035, 0.033, and 0.026, This is potentially indicating a focus on temporal aspects of systems and applications. Similarly, Topic 1 appears to concentrate on hardware and software technologies, with terms like

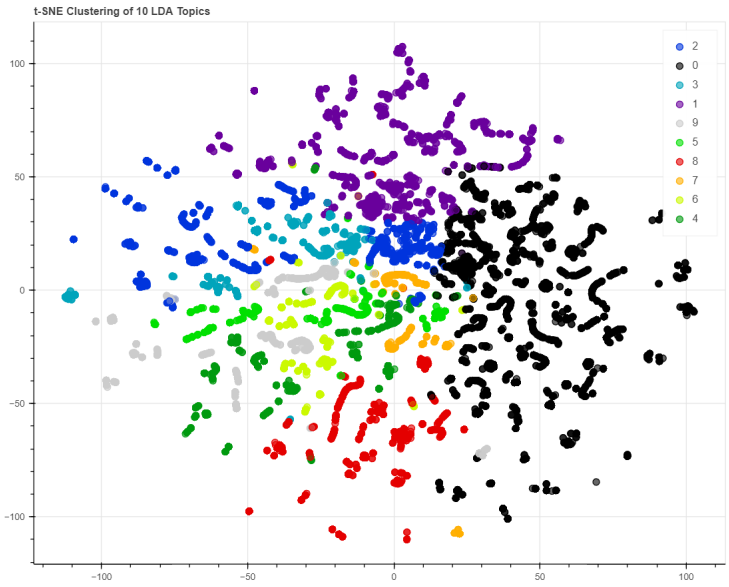

Figure 4.8: t-SNE Clustering of 10 LDA Topics

”snapdragon,” ”sd,” and ”iot” carrying weights of 0.336, 0.122, and 0.061, respectively. This could imply that the topic is likely related to technological specifications or features. It is imperative to note that topic 7 contains terms such as ”attacker,” ”firefox,” and ”affected,” this could be a sign of a cybersecurity attack, in addition to being a vulnerability in the system.

The next step was to identify the dominant topic for each document. The function ”`format_topic_sentences`” was utilized for this purpose. The dataframe produced by this serves as the foundational structure for plotting the distribution of document word counts both globally and by dominant topic. The first graph in figure 4.9, focuses on the global distribution of document word counts across the entire corpus, for the severity level critical. This plot serves as an initial exploration of the corpus, offering insights into its complexity and variability. From the graph we can see that the mean is 25, that is the average length of a description is 25 words. Although the median, which is less sensitive to outliers, is slightly lower which suggests that a few documents have a really high word count. A standard deviation of 21 signifies a moderate level of dispersion around the mean. In other words, while most documents hover around the average length, there is a substantial number that deviates significantly, either being much shorter or much longer. This is further corroborated by the 1st and 99th percentiles, which stand at 5 and 99, respectively. This wide range highlights the diversity in the lengths of documents within the corpus.

In the final step of the textual analysis, we will perform a t-sne model on the data to visualize the topics generated by the LDA model for critical severity. To accomplish this, we will follow similar steps as before and calculate the topic weights for each document filter those with a dominant topic weight greater than 0.35, and store that in an array. Then we will apply the t-sne model to this array of topic weights, which will reduce the dimensionality of the data while preserving the pair-wise distance between high dimensional data points, resulting in a 2D representation of the topics.

As we can see through the graph in figure 4.10, the data has been divided into 10 topics, each with its own cluster. Some clusters are overlapping suggesting that there are some topics that the model cannot distinguish between. It should be noted that the graph indicates that more clusters can

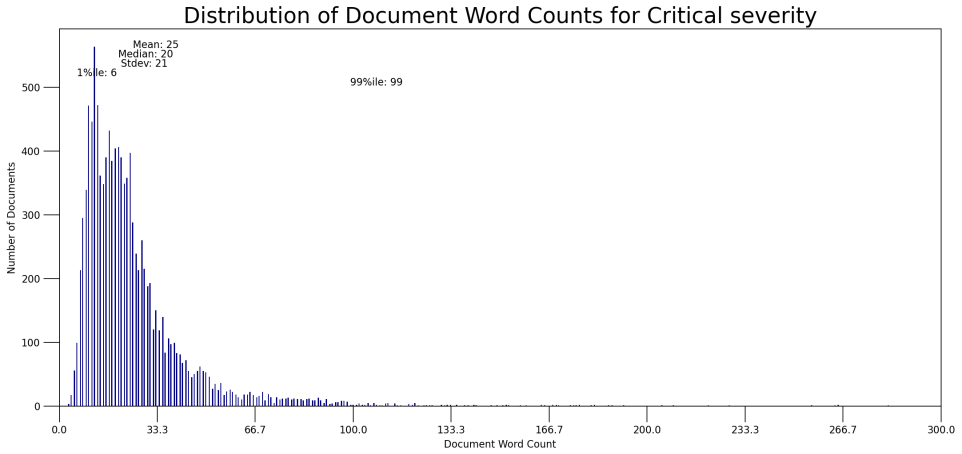

Figure 4.9: Distribution of Document Word Counts for Critical Severity

be formed, i.e., the number of topics should be increased for this model so as to accurately capture the themes of the text. It is important to note that the purpose of this model is not for detailed topic categorization, rather the model serves to provide insights into variations according to severity. As such increasing the number of topics is not in the scope of this analysis.

High:

After the application of LDA, the model produces 10 topics. Topic 0 is characterized by terms like ”issue” and ”tensorflow,” with weights of 0.074 and 0.066, respectively. This suggests that the topic may be focused on challenges or issues related to machine learning frameworks, possibly TensorFlow. Topic 1 is dominated by terms such as ”enterprise” and ”ibm,” indicating a focus on enterprise-level software or systems, potentially those provided by IBM. Topic 4 is particularly striking, with the term ”earlier” having a weight of 0.388. This could imply a focus on version histories or updates, especially when considered alongside terms like ”oracle” and ”access.” Topic 5, with ”attacker” and ”device” having weights of 0.216 and 0.098, respectively, appears to be centred around security vulnerabilities and affected devices. Topic 8 is overwhelmingly characterized by the term ”windows,” with a weight of 0.306, suggesting a focus on Windows operating systems or related technologies. Topic 9, with ”snapdragon” having a weight of 0.271, could be focusing on hardware, possibly mobile device chipsets.

The next step was to identify the dominant topic for each document. The function ”`format_topic_sentences`” was utilized for this purpose. The dataframe produced by this serves as the foundational structure for plotting the distribution of document word counts both globally and by dominant topic. The first graph as seen in figure 4.11 focuses on the global distribution of document word counts across the entire corpus, for the severity level high. This plot serves as an initial exploration of the corpus, offering insights into its complexity and variability. From the graph we can see that the mean is 30, that is the average length of a description is 30 words. Although the median, which is less sensitive to outliers, is slightly lower which suggests that a few documents have a really high word count. A standard deviation of 24 signifies a moderate level of dispersion around the mean. In other words, while most documents hover around the average length, there is a substantial number that deviates significantly, either being much shorter or much longer. This is further corroborated by the 1st and 99th percentiles, which stand at 5 and 116, respectively. This wide range highlights the diversity in

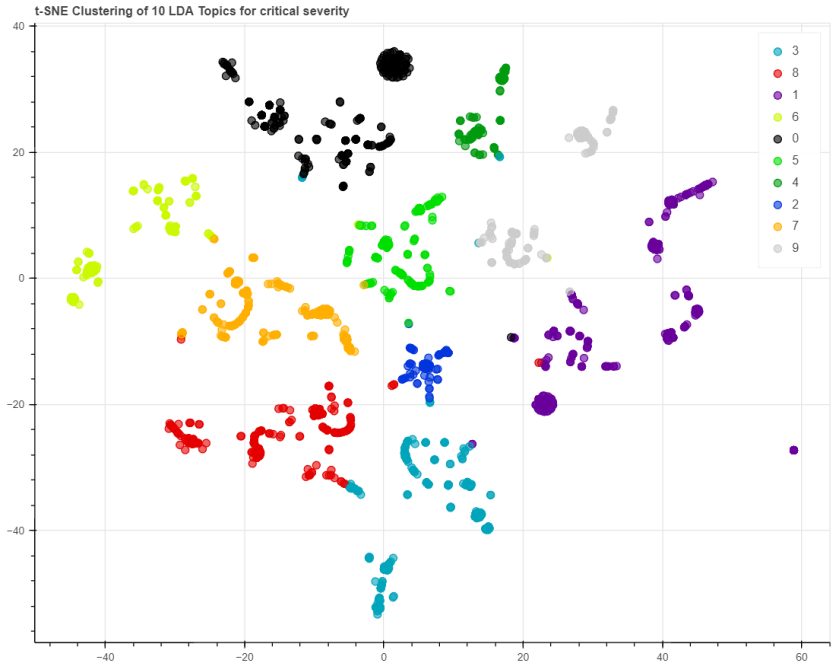

Figure 4.10: t-SNE Clustering of 10 LDA Topics for Critical Severity

the lengths of documents within the corpus.

As we can see from the graph in figure 4.12, the data has been divided into 10 topics, each with its own cluster. We can see that there are no clear clusters, so many of them are overlapping suggesting that there are some topics that the model cannot distinguish between.

Low:

After the application of LDA, the model produces 10 topics. Topic 0 is dominated by the term ”imagemagick” with a weight of 0.118, suggesting a focus on image processing or manipulation, possibly in a security context given the presence of the term ”attacker.” Topic 1 features terms like ”access” and ”jenkins,” indicating a focus on access control or permissions. Topic 2 is characterized by ”oracle” and ”earlier,” with weights of 0.155 and 0.122 respectively, suggesting a focus on Oracle technologies, possibly in the context of versioning or updates. Topic 3, with dominant terms like ”prior” and ”cluster,” seems to focus on versioning and clustering technologies, possibly in a MySQL environment given the term’s weight of 0.100. Topic 4 is heavily weighted towards ”server” and ”mysql,” indicating a focus on server-side technologies and MySQL databases. Topic 5, featuring ”information” and ”ibm,” could be centred around information management or analytics, possibly in an IBM software environment. Topic 6, with terms like ”issue” and ”versions,” appears to focus on software issues and versioning, possibly in a specialized field like visualization given the term’s weight of 0.030. Topic 7 is overwhelmingly characterized by the term ”app,” with a weight of 0.271, suggesting a focus on application development or security, possibly in an Android environment. Topic 8, dominated by ”java” and ”se,” likely focuses on Java Standard Edition, possibly in the context of embedded systems. Topic 9, featuring terms like ”data” and ”file,” appears to focus on data protection and file security, as indicated by terms like ”vault” and ”attacker.”

The next step was to identify the dominant topic for each document. The function ”`format_topic_sentences`” was utilized for this purpose. The dataframe produced by this serves as the foundational structure for

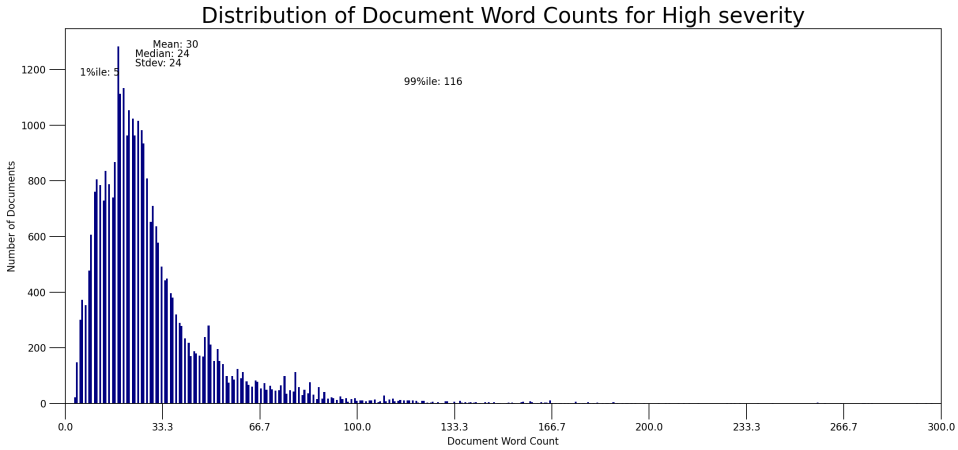

Figure 4.11: Distribution of Document Word Counts for High Severity

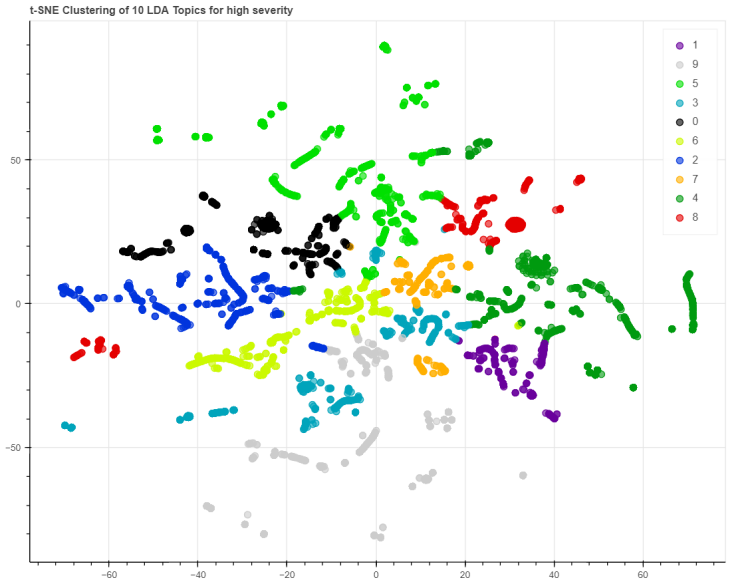

Figure 4.12: t-SNE Clustering of 10 LDA Topics for High Severity

plotting the distribution of document word counts both globally and by dominant topic. The first graph as seen in figure 4.13 focuses on the global distribution of document word counts across the entire corpus, for the severity level high. This plot serves as an initial exploration of the corpus, offering insights into its complexity and variability. From the graph we can see that the mean is 34, that is the average length of a description is 34 words. Although the median, which is less sensitive to outliers, is slightly lower which suggests that a few documents have a really high word count. A standard deviation of 23 signifies a moderate level of dispersion around the mean. In other words, while most documents hover around the average length, there is a substantial number that deviates significantly, either being much shorter or much longer. This is further corroborated by the 1st and 99th percentiles, which stand at 7 and 113, respectively. This wide range highlights the diversity in the lengths of documents within the corpus.

As we can see through the graph in figure 4.14, the data has been divided into 10 distinct topics,

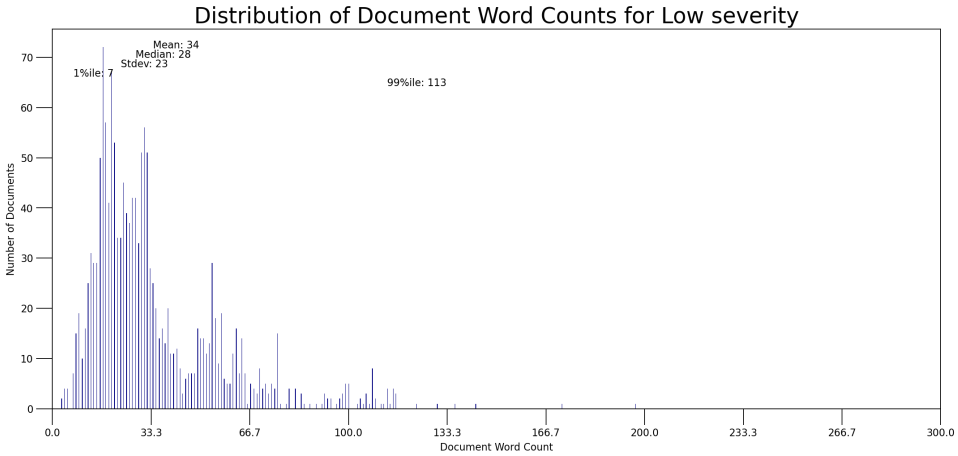

Figure 4.13: Distribution of Document Word Counts for Low Severity

each with its own cluster. Only one cluster is overlapping at one place, but all the other topics have their own separate distinct clusters, showing that the model is able to distinguish between them properly. It should be noted that all the clusters are not close to each other thus making it easier for the model.

Medium

Topic 0 seems to focus on Oracle technologies and access control mechanisms, particularly in the context of file management. This is evident from the dominant terms ”oracle,” ”access,” and ”file.” Topic 1 appears to be centred around image processing or manipulation, possibly in the context of email services. This is indicated by the prominence of terms like ”image” and ”email.” Topic 2 likely deals with security vulnerabilities, especially those that could be exploited through device interfaces. The dominant terms ”attacker,” ”interface,” and ”affected” support this interpretation. Topic 3 is overwhelmingly concerned with server-side technologies and MySQL databases. The term ”android” also appears, suggesting that the topic may extend to mobile database management. Topic 4 seems to focus on software versioning and issues, possibly in a Magento or OTRS environment. This is indicated by the terms ”prior,” ”versions,” and ”issue.” Topic 5 likely focuses on Oracle technologies and Microsoft SharePoint, possibly in the context of virtual machines, as suggested by terms like ”oracle,” ”user,” and ”sharepoint.” Topic 6 appears to be concerned with earlier versions of technologies like TensorFlow and Java, as indicated by the dominant term ”earlier.” Topic 7 seems to focus on versioning and security in enterprise software, particularly PeopleSoft, as suggested by terms like ”version,” ”security,” and ”peoplesoft.” Topic 8 could be centered around Windows operating systems and IBM technologies, possibly in the context of application updates, as indicated by the terms ”windows” and ”ibm.” Topic 9 appears to focus on software development and application programming interfaces, possibly with a focus on secure shell (SSH) protocols, as suggested by terms like ”software,” ”api,” and ”ssh.”

The next step was to identify the dominant topic for each document. The function ”`format_topic_sentences`” was utilized for this purpose. The dataframe produced by this serves as the foundational structure for plotting the distribution of document word counts both globally and by dominant topic. The first graph seen in figure 4.15 focuses on the global distribution of document word counts across the entire corpus, for the severity level medium. This plot serves as an initial exploration of the corpus,

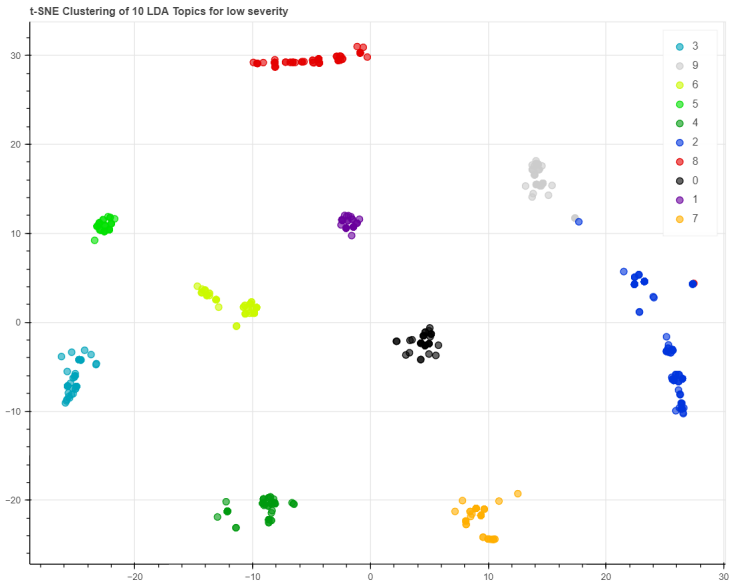

Figure 4.14: t-SNE Clustering of 10 LDA Topics for Low Severity

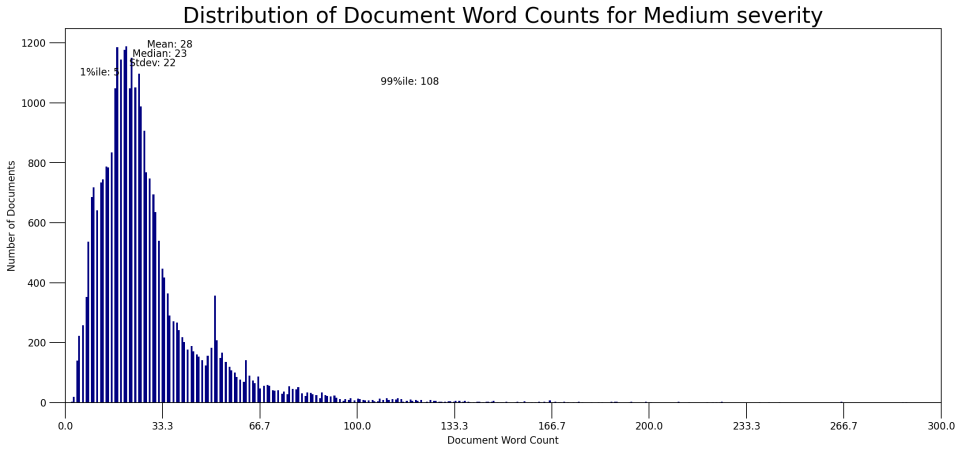

Figure 4.15: Distribution of Document Word Counts for Medium Severity

offering insights into its complexity and variability. From the graph we can see that the mean is 28, that is the average length of a description is 28 words. Although the median, which is less sensitive to outliers, is slightly lower which suggests that a few documents have a really high word count. A standard deviation of 22 signifies a moderate level of dispersion around the mean. In other words, while most documents hover around the average length, there is a substantial number that deviates significantly, either being much shorter or much longer. This is further corroborated by the 1st and 99th percentiles, which stand at 5 and 108, respectively. This wide range highlights the diversity in the lengths of documents within the corpus.

As we can see from the graph in figure 4.16, the data has been divided into 10 topics, each with its own cluster. We can see that there are no clear clusters, so many of them are overlapping suggesting that there are some topics that the model cannot distinguish between.

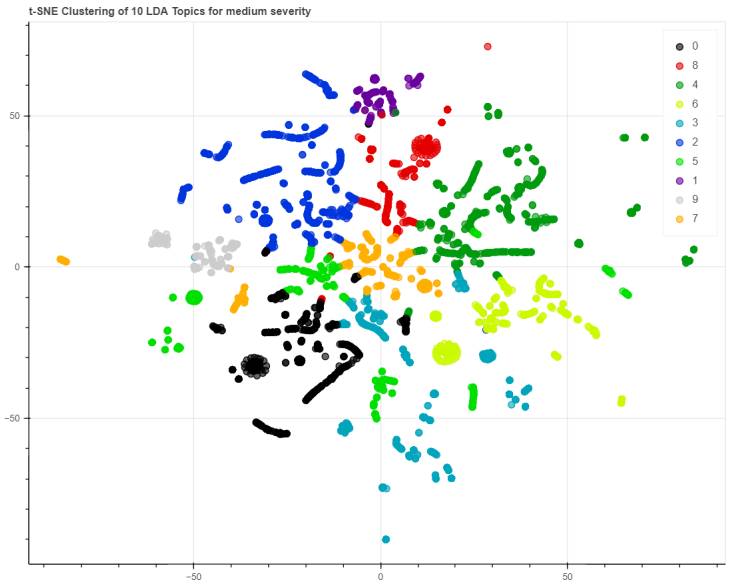

Figure 4.16: t-SNE Clustering of 10 LDA Topics for Medium Severity

Years

Given that there are four years, the textual analysis will be performed on all four years to understand how the themes vary for the years 2019, 2020, 2021 and 2022.

2019

Topic 0 appears to focus on core system functionalities and relational database management systems (RDBMS), as indicated by terms like ”core,” ”system,” and ”rdbms.” Topic 1 seems to be concerned with file management and password security, particularly in the context of the OTRS system. This is supported by the terms ”file,” ”password,” and ”otrs.” Topic 2 is predominantly about security vulnerabilities and exploits, with a focus on email-based attacks. The terms ”attacker,” ”email,” and ”exploit” are particularly telling. Topic 3 is overwhelmingly concerned with earlier versions of software or systems, possibly in the context of web-based management interfaces, as indicated by the dominant term ”earlier.” Topic 4 appears to focus on service configurations, particularly VPN services, as suggested by terms like ”service,” ”page,” and ”vpn.” Topic 5 seems to deal with image processing and Cross-Site Request Forgery (CSRF) vulnerabilities, possibly in containerized environments, as indicated by terms like ”image,” ”simatic,” and ”csrf.” Topic 6 is likely centered around functions and XML processing, possibly in the context of session management, as suggested by terms like ”function,” ”xml,” and ”session.” Topic 7 appears to focus on issues related to Snapdragon processors and Internet of Things (IoT) devices, as indicated by the terms ”issue,” ”snapdragon,” and ”iot.” Topic 8 seems to be concerned with SD cards and certificates, possibly in the context of Jenkins, as suggested by terms like ”sd,” ”snapdragon,” and ”certificate.” Topic 9 likely focuses on firmware security and versioning, possibly in the context of EOS systems, as indicated by terms like ”version,” ”firmware,” and ”security.” Topic 10 appears to be centered around server technologies, particularly MySQL and Windows, as suggested by terms like ”server,” ”mysql,” and ”windows.” Topic 11 seems to focus on Magento and software versioning, particularly prior versions, as indicated by terms like ”prior,” ”magento,” and ”versions.” Topic 12 likely deals with Oracle technologies and access control,

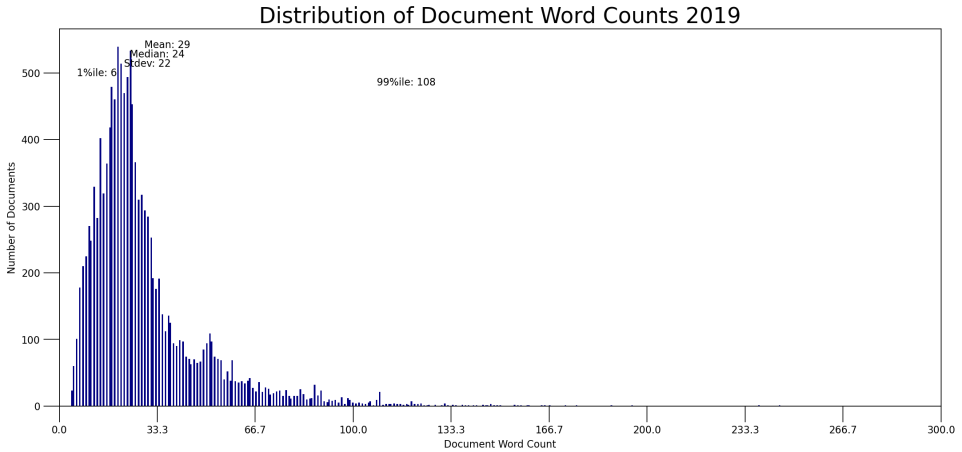

Figure 4.17: Distribution of Document Word Counts for 2019

possibly in the context of device security, as suggested by terms like ”oracle,” ”access,” and ”device.” Topic 13 appears to focus on software and remote services, possibly in the context of SharePoint, as indicated by terms like ”software,” ”sharepoint,” and ”remote.” Topic 14 seems to be concerned with user interactions, possibly in virtual environments or IP-based systems, as suggested by terms like ”user,” ”virtual,” and ”ip.”

The next step was to identify the dominant topic for each document. The function ”`format_topic_sentences`” was utilized for this purpose. The dataframe produced by this serves as the foundational structure for plotting the distribution of document word counts both globally and by dominant topic. The first graph in figure 4.17 focuses on the global distribution of document word counts across the entire corpus, for the year 2019. This plot serves as an initial exploration of the corpus, offering insights into its complexity and variability. From the graph we can see that the mean is 29, that is the average length of a description is 29 words. Although the median, which is less sensitive to outliers, is slightly lower which suggests that a few documents have a really high word count. A standard deviation of 22 signifies a moderate level of dispersion around the mean. In other words, while most documents hover around the average length, there is a substantial number that deviates significantly, either being much shorter or much longer. This is further corroborated by the 1st and 99th percentiles, which stand at 6 and 108, respectively. This wide range highlights the diversity in the lengths of documents within the corpus.

As we can see through the graph in figure 4.18, the data has been divided into 10 topics, each with its own cluster. Some clusters are overlapping suggesting that there are some topics that the model cannot distinguish between. It should be noted that the graph indicates that more clusters can be formed, i.e., the number of topics should be increased for this model so as to accurately capture the themes of the text. It is important to note that the purpose of this model is not for detailed topic categorization, rather the model serves to provide insights into variations according to the year. As such increasing the number of topics is not in the scope of this analysis.

2020

Topic 0 appears to focus on Oracle technologies and access control, with an emphasis on data management and Docker containers. The terms ”oracle,” ”access,” and ”data” are particularly indicative. Topic 1 seems to be concerned with security vulnerabilities and issues, with a focus on

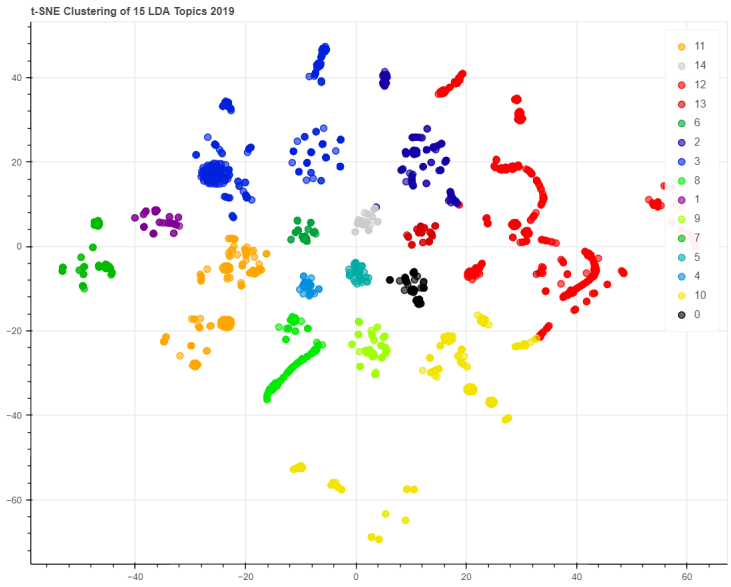

Figure 4.18: t-SNE Clustering of 10 LDA Topics for 2019

attackers exploiting earlier versions of software. The terms ”attacker,” ”earlier,” and ”issue” are noteworthy. Topic 2 likely deals with server technologies and user interfaces, particularly in the context of MySQL databases. The terms ”server,” ”user,” and ”mysql” are indicative of this focus. Topic 3 appears to be centred around device security and potential attacks, possibly involving Java and Linux systems. The terms ”device,” ”attacks,” and ”java” are particularly telling. Topic 4 seems to focus on file systems and Internet of Things (IoT) devices, possibly in the context of SharePoint and Tibco systems. The terms ”file,” ”iot,” and ”system” are indicative. Topic 5 is likely concerned with software versioning and virtualization technologies, particularly Oracle’s VirtualBox. The terms ”version,” ”information,” and ”vm” are noteworthy. Topic 6 appears to focus on IBM technologies and security measures, possibly involving cookies and Citrix systems. The terms ”ibm,” ”security,” and ”cookie” are indicative of this focus. Topic 7 seems to be centred around Windows operating systems, particularly in enterprise environments, with a focus on memory management and password security. The terms ”windows,” ”enterprise,” and ”memory” are particularly telling. Topic 8 likely deals with Snapdragon processors and software versioning, particularly prior versions. The terms ”snapdragon,” ”versions,” and ”prior” are indicative of this focus. Topic 9 appears to focus on device security, particularly in the context of Firefox and Android systems, possibly involving Cross-Site Scripting (XSS) vulnerabilities. The terms ”devices,” ”firefox,” and ”android” are noteworthy.

The next step was to identify the dominant topic for each document. The function ”`format_topic_sentences`” was utilized for this purpose. The dataframe produced by this serves as the foundational structure for plotting the distribution of document word counts both globally and by dominant topic. IMAGE The first graph as seen in figure 4.19 focuses on the global distribution of document word counts across the entire corpus, for the year 2020. This plot serves as an initial exploration of the corpus, offering insights into its complexity and variability. From the graph we can see that the mean is 30, that is the average length of a description is 30 words. Although the median, which is less sensitive to outliers, is slightly lower which suggests that a few documents have a really high word count. A standard deviation of 22 signifies a moderate level of dispersion around the mean. In other words,

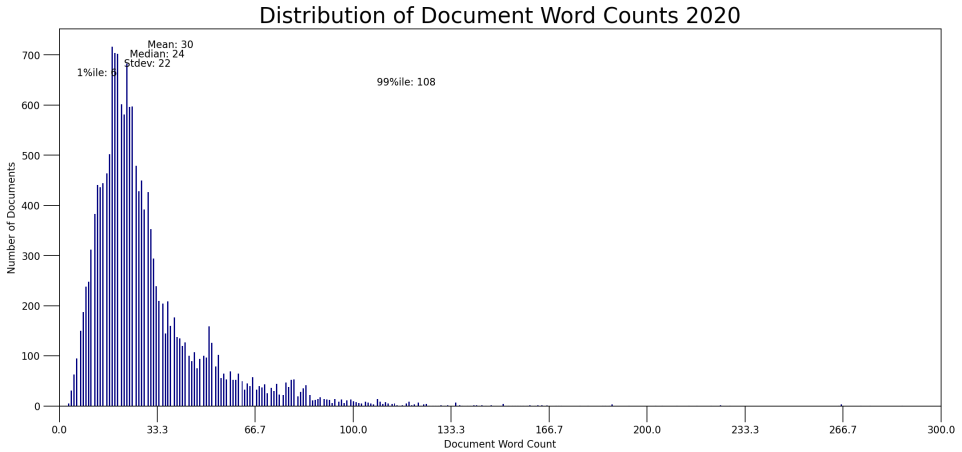

Figure 4.19: Distribution of Document Word Counts for 2020

while most documents hover around the average length, there is a substantial number that deviates significantly, either being much shorter or much longer. This is further corroborated by the 1st and 99th percentiles, which stand at 6 and 108, respectively. This wide range highlights the diversity in the lengths of documents within the corpus.

As we can see from the graph in figure 4.20, the data has been divided into 10 topics, each with its own cluster. Some clusters are overlapping suggesting that there are some topics that the model cannot distinguish between. It should be noted that the graph indicates that more clusters can be formed, i.e., the number of topics should be increased for this model so as to accurately capture the themes of the text.

2021

Topic 0 appears to focus on web technologies, particularly HTTP protocols and Cross-Site Scripting (XSS) vulnerabilities. The terms ”http,” ”software,” and ”xss” are indicative of this focus. Topic 1 seems to be concerned with file systems and Oracle technologies, particularly in the context of virtual machines (VMs) and VirtualBox. The terms ”file,” ”oracle,” and ”vm” are noteworthy. Topic 2 likely deals with user authentication and memory management, possibly involving Tibco systems. The terms ”user,” ”memory,” and ”tibco” are indicative of this focus. Topic 3 appears to be centred around device security, with an emphasis on vulnerabilities and potential exploits. The terms ”attacker,” ”device,” and ”vulnerabilities” are particularly telling. Topic 4 seems to focus on software versioning, particularly prior versions and may involve Aruba technologies. The terms ”versions,” ”prior,” and ”aruba” are indicative. Topic 5 is likely concerned with Snapdragon processors and server technologies, particularly MySQL databases and IoT applications. The terms ”snapdragon,” ”server,” and ”mysql” are noteworthy. Topic 6 appears to focus on access control and data security, possibly involving Oracle technologies and password management. The terms ”access,” ”data,” and ”oracle” are indicative of this focus. Topic 7 seems to be centred around IBM technologies and user management, possibly involving SQL databases. The terms ”version,” ”ibm,” and ”users” are particularly telling. Topic 8 likely deals with TensorFlow technologies and earlier versions of software, possibly involving policy management. The terms ”tensorflow,” ”earlier,” and ”policy” are indicative of this focus. Topic 9 appears to focus on system issues, particularly in the context of Firefox and Nextcloud technologies. The terms ”system,” ”issue,” and ”firefox” are noteworthy.

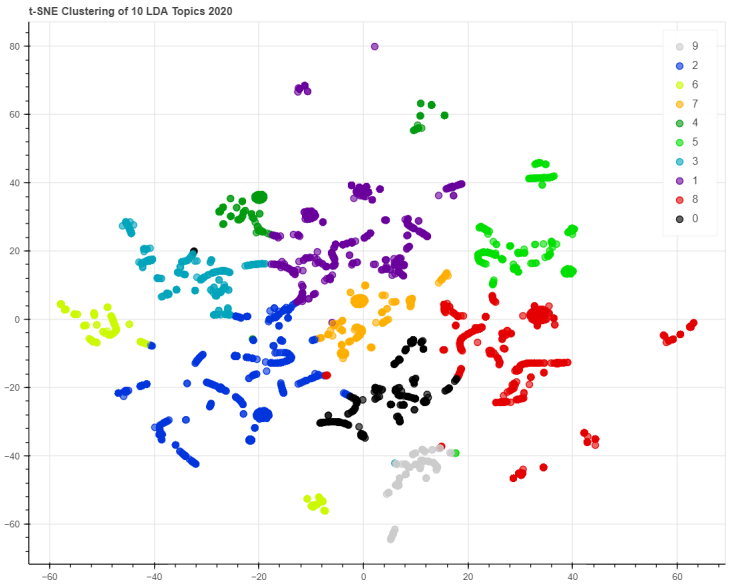

Figure 4.20: t-SNE Clustering of 10 LDA Topics for 2020

The next step was to identify the dominant topic for each document. The function "`format_topic_sentences`" was utilized for this purpose. The dataframe produced by this serves as the foundational structure for plotting the distribution of document word counts both globally and by dominant topic. The first graph as seen in figure 4.21 focuses on the global distribution of document word counts across the entire corpus, for the year 2021. This plot serves as an initial exploration of the corpus, offering insights into its complexity and variability. From the graph we can see that the mean is 29, that is the average length of a description is 29 words. Although the median, which is less sensitive to outliers, is slightly lower which suggests that a few documents have a really high word count. A standard deviation of 22 signifies a moderate level of dispersion around the mean. In other words, while most documents hover around the average length, there is a substantial number that deviates significantly, either being much shorter or much longer. This is further corroborated by the 1st and 99th percentiles, which stand at 5 and 116, respectively. This wide range highlights the diversity in the lengths of documents within the corpus.

As we can see from the graph in figure 4.22, the data has been divided into 10 topics, each with its own cluster. Some clusters are overlapping suggesting that there are some topics that the model cannot distinguish between.

2022

Topic 0 appears to focus on software flaws and functions, possibly in the context of Blue Prism and Apache Airflow technologies. The terms "flaw," "function," and "blue" are indicative of this focus. Topic 1 seems to be concerned with issues related to authentication and system keys, possibly involving message protocols. The terms "issue," "authentication," and "key" are noteworthy in this context. Topic 2 likely deals with TensorFlow technologies and software versioning, particularly prior versions. The terms "versions," "prior," and "tensorflow" are indicative of this focus. Topic 3 appears to be centred around SQL databases and memory management, possibly involving Firefox and file directories. The terms "sql," "memory," and "firefox" are particularly telling. Topic 4 seems to focus on service-related issues, possibly involving denial-of-service attacks and build packs. The terms

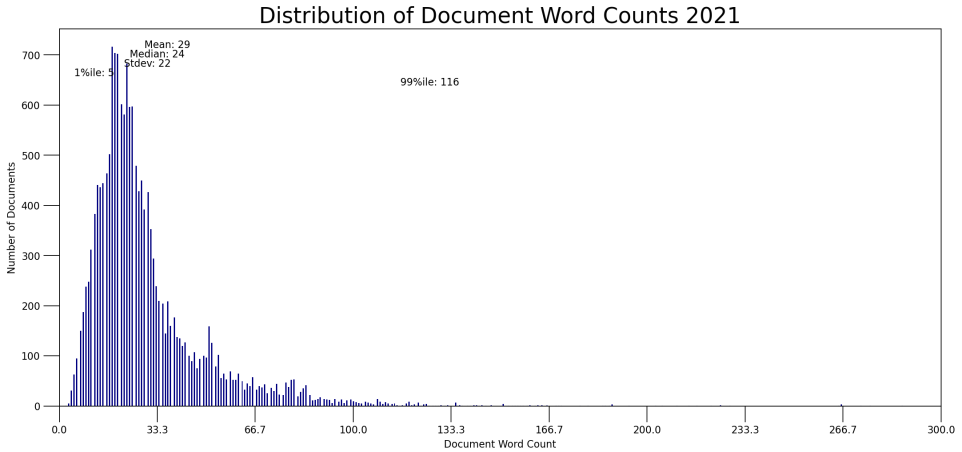

Figure 4.21: Distribution of Document Word Counts for 2021

”service,” ”may,” and ”denial” are indicative. Topic 5 is likely concerned with web-based attacks and information security, possibly involving Apache technologies and password management. The terms ”attacker,” ”web,” and ”information” are noteworthy. Topic 6 appears to focus on Snapdragon processors and file systems, possibly involving Jenkins and buffer overflows. The terms ”snapdragon,” ”file,” and ”jenkins” are indicative of this focus. Topic 7 seems to be centred around user management and access control, possibly involving IBM technologies. The terms ”versions,” ”version,” and ”user” are particularly telling. Topic 8 likely deals with earlier versions of server technologies, possibly involving Oracle and MySQL databases. The terms ”earlier,” ”server,” and ”oracle” are indicative of this focus. Topic 9 appears to focus on operating systems like macOS and Windows, possibly involving Tibco and Nextcloud technologies. The terms ”macos,” ”command,” and ”tibco” are noteworthy.

The next step was to identify the dominant topic for each document. The function ”`format_topic_sentences`” was utilized for this purpose. The dataframe produced by this serves as the foundational structure for plotting the distribution of document word counts both globally and by dominant topic. The first graph as seen in figure 4.23 focuses on the global distribution of document word counts across the entire corpus, for the year 2022. This plot serves as an initial exploration of the corpus, offering insights into its complexity and variability. From the graph we can see that the mean is 26, that is the average length of a description is 26 words. Although the median, which is less sensitive to outliers, is slightly lower which suggests that a few documents have a really high word count. A standard deviation of 24 signifies a moderate level of dispersion around the mean. In other words, while most documents hover around the average length, there is a substantial number that deviates significantly, either being much shorter or much longer. This is further corroborated by the 1st and 99th percentiles, which stand at 5 and 113, respectively. This wide range highlights the diversity in the lengths of documents within the corpus.

As we can see from the graph in figure 4.24, the data has been divided into 10 topics, each with its own cluster. Some clusters are overlapping suggesting that there are some topics that the model cannot distinguish between.

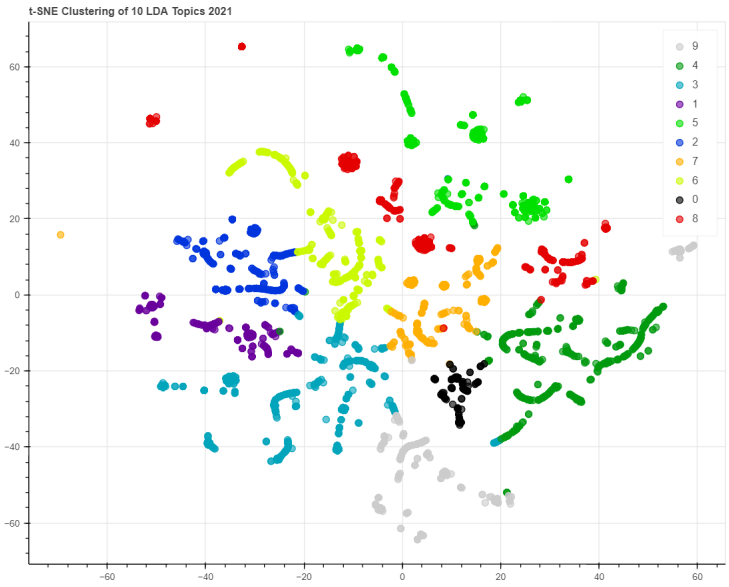

Figure 4.22: t-SNE Clustering of 10 LDA Topics for 2021

4.5 Analysis based on machine learning and deep learning

In this section, LDA will be used to engineer features that capture the topical essence of each description and use it for predicting the severity. Through the integration of LDA with machine learning models, like Logistic Regression, Decision Tree, Random Forest and Support Vector Machines, and the deep learning network Long Short-Term Memory.

4.5.1 Machine Learning Algorithms with LDA

The first step is to split the data into train and test data, where 70% of the data is used for training and 30% is used for testing. Stratification is used during splitting to ensure that the distribution of labels in the test set is similar to that in the training set. A function "`get_topic_distribution_for_descriptions`" is defined to obtain the topic distribution from each description, this function first converts the descriptions into a document-term matrix, transforms this document-term matrix into a form suitable for LDA and calculates topic probabilities for each document based on the trained LDA model. To integrate LDA with our machine learning pipelines, the "LDATransformer" class is defined. The purpose of this class is to preprocess text data by extracting topic distributions from text documents and transforming them into feature vectors that can be used for machine learning tasks. The class takes two inputs during initialization, the LDA model and the number of topics, If an LDA model is not provided the class trains its own LDA model. Once the LDA model is trained or provided, it takes a collection of documents as the input and applies the LDA model to the documents, extracting the topic distributions for each document. The topic distributions are returned as dense arrays, which can be used as features in subsequent machine-learning models.

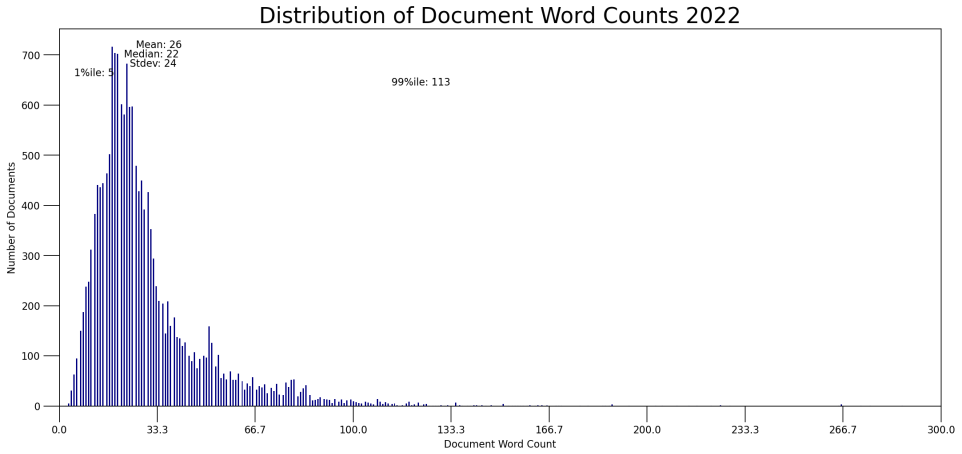

Figure 4.23: Distribution of Document Word Counts for 2022

Dummy Classifier

A simple dummy classifier was used in order to establish a baseline to evaluate the performance of the models. This classifier employs the basic strategy of predicting the most frequent class in the training data, effectively mimicking a model that makes predictions with no consideration for input features. From the classification report for the dummy classifier in figure 4.25, we can see that the overall accuracy is 42%, accuracy alone might not be a reliable metric, as the classes are imbalanced. The precision for class 1 is 0.42, which means that when the model predicts class 1, it is correct approximately 42% of the time. The recall for class 1 is 1, suggesting that the model is good at identifying instances of class 1 when they are present in the test data. However, it's important to note that recall for other classes (0, 2, and 3) is 0, which indicates that the model struggles to identify these classes. The F1-score for class 1 is 0.59, however, the F1-score for other classes is 0 indicating poor performance. Support represents the number of actual occurrences of each class in the test dataset. Classes 1 and 3 have higher support, while classes 0 and 2 have lower support. The macro-average F1-score and weighted-average F1-score provide an overall view of model performance across all classes. In this case, both macro and weighted averages are low, indicating that the model's performance is generally poor across all classes.

Logistic Regression

A logistic Regression model was used as the initial classifier model along with its variations by incorporating the penalty terms such as L1(Lasso), L2(Ridge) and L1-L2, to evaluate their impact on the model's performance.

From the classification report for Logistic Regression in figure 4.26, we can see that the overall accuracy is 45%, It suggests that the model's performance is somewhat limited in correctly classifying instances across all classes. Class 0 (Label 0) has a relatively low precision and recall, indicating that the model has difficulty correctly identifying instances of this class, often leading to false positives and false negatives. Class 1 (Label 1) exhibits good precision and recall, indicating that the model performs well in correctly classifying instances of this class. Class 2 (Label 2) has a high precision but low recall, suggesting that while the model's predictions for this class are accurate, it misses

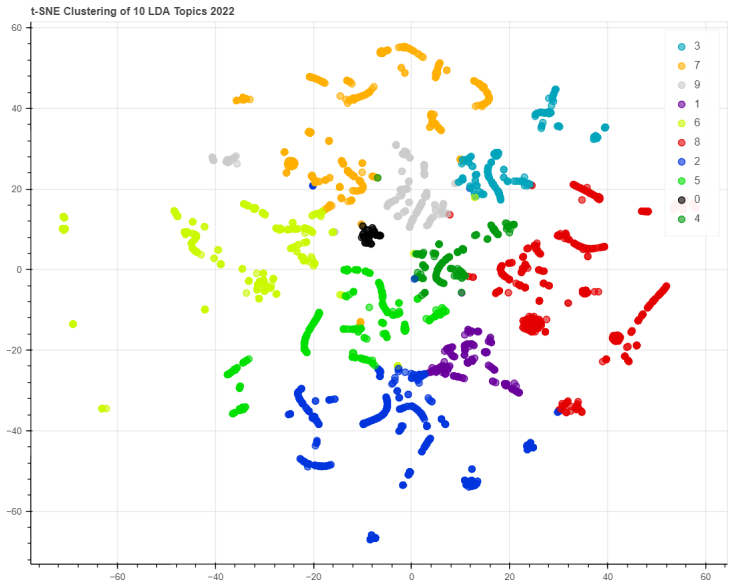

Figure 4.24: t-SNE Clustering of 10 LDA Topics for 2022

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2241
1	0.42	1.00	0.59	6218
2	0.00	0.00	0.00	310
3	0.00	0.00	0.00	6084
accuracy			0.42	14853
macro avg	0.10	0.25	0.15	14853
weighted avg	0.18	0.42	0.25	14853

Figure 4.25: Classification report from the Dummy Classifier

a significant number of actual instances. Class 3 (Label 3) shows moderate precision and recall, indicating some ability to identify instances but also some misclassification. The F1-score is highest for Class 1, indicating the best overall performance for this class. The macro-average F1-score and precision are lower than the weighted averages, indicating that the performance is not consistent across all classes. The weighted average gives more weight to the larger classes (e.g., Class 1 and Class 3) due to their higher support, while the macro-average treats all classes equally. Class 1 has significantly more samples than the other classes. This imbalance may affect the model's ability to correctly classify minority classes. In summary, this classification model performs relatively well for Class 1 but struggles with the other classes, particularly Class 0 and Class 2.

As there is an imbalance in class sizes, K-fold stratified cross-validation was implemented to help tackle this problem. Stratified cross-validation is a technique that ensures each fold of the data maintains the same class distribution as the original dataset, potentially helping the model perform better when classes are imbalanced.

The cross-validation results for the logistic regression model indicate a consistent pattern across the folds. The accuracy scores for each fold range from approximately 45.29% to 45.82%. The precision scores range from around 0.4977 to 0.5274. Higher precision suggests fewer false positives, but we should also consider the trade-off with recall. The recall scores exhibit consistency with values between 0.4529 and 0.4582. The F1 scores vary between approximately 0.3543 and 0.3629. Overall,

	precision	recall	f1-score	support
0	0.38	0.02	0.04	2241
1	0.44	0.92	0.59	6218
2	0.78	0.05	0.09	310
3	0.60	0.16	0.25	6084
accuracy			0.45	14853
macro avg	0.55	0.29	0.24	14853
weighted avg	0.50	0.45	0.36	14853

Figure 4.26: Classification report from Logistic Regression

	precision	recall	f1-score	support
0	0.41	0.02	0.05	2241
1	0.44	0.93	0.59	6218
2	0.75	0.02	0.04	310
3	0.61	0.15	0.24	6084
accuracy			0.45	14853
macro avg	0.55	0.28	0.23	14853
weighted avg	0.51	0.45	0.35	14853

Figure 4.27: Classification report from Logistic Regression with L1 Penalty

while the logistic regression model doesn't exhibit dramatic fluctuations in performance across folds, the relatively low F1 scores indicate room for improvement.

Logistic Regression with L1 Penalty

From the classification report for Logistic Regression with L1 penalty in figure 4.27, we can see that the overall accuracy of the model is still 0.45, indicating that the overall performance has not changed significantly. Class 0 (Label 0) still has relatively low precision and recall, indicating challenges in correctly identifying instances of this class. Class 1 (Label 1) continues to exhibit good precision and recall, suggesting reliable performance for this class. Class 2 (Label 2) maintains a high precision but low recall, similar to the previous results. Class 3 (Label 3) shows moderate precision and recall, indicating some success in identifying instances. The F1-score remains highest for Class 1, indicating consistent performance for this class. Both the macro-average and weighted average F1 scores are similar to the previous results. The macro-average emphasizes the lower-performing classes, while the weighted average gives more weight to the larger classes. The class imbalance remains a factor, with Class 1 having significantly more samples than the other classes. In summary, the updated classification model still demonstrates good performance for Class 1 but struggles with other classes, particularly Class 0 and Class 2.

The cross-validation results for the logistic regression with the L1-penalty model indicate a consistent pattern across the folds. The accuracy scores for each fold range from approximately 44.82% to 45.50%. The precision scores range from around 0.4979 to 0.5215. Higher precision suggests fewer false positives, but we should also consider the trade-off with recall. The recall scores exhibit consistency with values between 0.4482 and 0.4550. The F1 scores vary between approximately 0.34850 and 0.3584.

	precision	recall	f1-score	support
0	0.38	0.02	0.04	2241
1	0.44	0.92	0.59	6218
2	0.78	0.05	0.09	310
3	0.60	0.16	0.25	6084
accuracy			0.45	14853
macro avg	0.55	0.29	0.24	14853
weighted avg	0.50	0.45	0.36	14853

Figure 4.28: Classification report from Logistic Regression with L2 Penalty

Logistic Regression with L2 Penalty

From the classification report for Logistic Regression with L2 penalty in figure 4.28, we can see that the overall accuracy of the model is 0.45, indicating that the model's performance is moderate. Class 0 (Label 0) has very low precision and recall, indicating significant challenges in correctly identifying instances of this class. Class 1 (Label 1) exhibits a relatively high precision and recall, suggesting good performance for this class. Class 2 (Label 2) shows a high precision but low recall, which could indicate that the model is conservative in predicting this class. Class 3 (Label 3) demonstrates moderate precision and recall, indicating some success in identifying instances. The F1 score varies across classes, with Class 1 having the highest F1 score, followed by Class 3, Class 2, and Class 0. The macro-average F1-score is 0.24, reflecting the average performance across all classes. The weighted average F1-score is 0.36, giving more weight to classes with larger support. In summary, the classification model appears to perform well for Class 1 but struggles with Class 0, Class 2, and to some extent, Class 3.

The cross-validation results for the logistic regression with the L1-penalty model indicate a consistent pattern across the folds. The accuracy scores for each fold range from approximately 45.28% to 45.81%. The precision scores range from around 0.4966 to 0.5274. Higher precision suggests fewer false positives, but we should also consider the trade-off with recall. The recall scores exhibit consistency with values between 0.4528 and 0.4581. The F1 scores vary between approximately 0.3542 and 0.3629.

Given that, K-fold stratified cross-validation did not improve the results of the model, I decided not to utilize this for the subsequent machine learning techniques.

Logistic Regression with L1-L2 Penalty

From the classification report for Logistic Regression with L1-L2 penalty in figure 4.29, we can see that the overall accuracy of the model is 0.45, which is consistent with the previous set of results, indicating that the model's performance is moderate. Class 0 (Label 0) has very low precision and recall, indicating significant challenges in correctly identifying instances of this class. Class 1 (Label 1) exhibits a relatively high precision and recall, suggesting good performance for this class. Class 2 (Label 2) shows a high precision but low recall, which could indicate that the model is conservative in predicting this class. Class 3 (Label 3) demonstrates moderate precision and recall, indicating some success in identifying instances. The F1 score varies across classes, with Class 1 having the highest F1 score, followed by Class 3, Class 2, and Class 0. The macro-average F1-score is 0.24, reflecting the average performance across all classes. The weighted average F1-score is 0.36, giving more weight to classes with larger support.

	precision	recall	f1-score	support
0	0.38	0.02	0.04	2241
1	0.44	0.92	0.59	6218
2	0.78	0.05	0.09	310
3	0.60	0.16	0.25	6084
accuracy			0.45	14853
macro avg	0.55	0.29	0.24	14853
weighted avg	0.50	0.45	0.36	14853

Figure 4.29: Classification report from Logistic Regression with L1-L2 Penalty

	precision	recall	f1-score	support
0	0.25	0.04	0.06	2241
1	0.43	0.91	0.58	6218
2	0.11	0.02	0.04	310
3	0.53	0.11	0.18	6084
accuracy			0.43	14853
macro avg	0.33	0.27	0.21	14853
weighted avg	0.44	0.43	0.33	14853

Figure 4.30: Classification report from Decision Tree

The analysis for this set of results is similar to the previous one, with Class 1 performing well and the model struggling with Class 0, Class 2, and to some extent, Class 3.

Decision Tree

From the classification report for Decision Tree in figure 4.30, we can see that the overall accuracy of the model is 0.43, Although lower than the previous model it is still higher than the dummy model, which suggests that the model's performance is moderate. Class 0 (Label 0) has a low precision and recall, indicating challenges in correctly identifying instances of this class. The model often misses class 0 and has a relatively high false-negative rate. Class 1 (Label 1) exhibits a relatively high precision and recall, suggesting good performance for this class. The model is successful at both correctly identifying class 1 instances and avoiding false positives. Class 2 (Label 2) shows low precision and recall, indicating that the model struggles to correctly predict this class. Class 3 (Label 3) demonstrates moderate precision and recall, indicating some success in identifying instances of this class. The F1 scores vary across classes, with Class 1 having the highest F1 score, followed by Class 3, Class 0, and Class 2. The macro-average F1-score is 0.21, reflecting the average performance across all classes. It is relatively low, primarily due to the challenges in Class 0 and Class 2. The weighted average F1-score is 0.33, giving more weight to classes with larger support. In summary, this classification model performs relatively well for Class 1, but it struggles with the other classes, particularly Class 0 and Class 2.

Random Forest

From the classification report for Random Forest in figure 4.31, we can see that the overall accuracy of the model is 0.44, which suggests that the model's performance is moderate. Class 0 (Label 0) has a low precision and recall, indicating challenges in correctly identifying instances of this class. The model often misses class 0 and has a relatively high false-negative rate. Class 1 (Label 1) exhibits a relatively high precision and recall, suggesting good performance for this class. The model is

	precision	recall	f1-score	support
0	0.46	0.01	0.02	2241
1	0.43	0.91	0.58	6218
2	0.00	0.00	0.00	310
3	0.53	0.15	0.23	6084
accuracy			0.44	14853
macro avg	0.35	0.27	0.21	14853
weighted avg	0.46	0.44	0.34	14853

Figure 4.31: Classification report from Random Forest

	precision	recall	f1-score	support
0	0.42	0.03	0.05	2241
1	0.44	0.93	0.59	6218
2	0.83	0.02	0.03	310
3	0.62	0.15	0.24	6084
accuracy			0.45	14853
macro avg	0.58	0.28	0.23	14853
weighted avg	0.52	0.45	0.36	14853

Figure 4.32: Classification report from Support Vector Machines

successful at both correctly identifying class 1 instances and avoiding false positives. Class 2 (Label 2) shows low precision and recall, indicating that the model struggles to correctly predict this class. Class 3 (Label 3) demonstrates moderate precision and recall, indicating some success in identifying instances of this class. The F1 scores vary across classes, with Class 1 having the highest F1 score, followed by Class 3, Class 0, and Class 2. The macro-average F1-score is 0.21, reflecting the average performance across all classes. It is relatively low, primarily due to the challenges in Class 0 and Class 2. The weighted average F1-score is 0.34, giving more weight to classes with larger support. In summary, this classification model performs relatively well for Class 1, but it struggles with the other classes, particularly Class 0 and Class 2.

Support Vector Machines

From the classification report for Support Vector Machines, we can see that the overall accuracy of the model is 0.45, which suggests that the model's performance is moderate. Class 0 (Label 0) shows low precision and recall, indicating challenges in correctly identifying instances of this class. The model often misses class 0 and has a relatively high false-negative rate. Class 1 (Label 1) demonstrates a relatively high precision and recall, suggesting good performance for this class. The model is successful at both correctly identifying class 1 instances and avoiding false positives. Class 2 (Label 2) exhibits a high precision but very low recall, indicating that the model predicts this class sparingly, and when it does, it is mostly accurate. Class 3 (Label 3) shows moderate precision and recall, indicating some success in identifying instances of this class. The F1 scores vary across classes, with Class 1 having the highest F1 score, followed by Class 3, Class 0, and Class 2. The macro-average F1-score is 0.23, reflecting the average performance across all classes. It is relatively low, primarily due to the challenges in Class 0 and Class 2. The weighted average F1-score is 0.36, giving more weight to classes with larger support. In summary, this classification model performs relatively well for Class 1, but it struggles with the other classes, particularly Class 0 and Class 2.

In conclusion, the machine learning models, integrated with LDA displayed an accuracy of around

45%. It is essential to note that this accuracy is due to the fact that there is a significant imbalance in the classes. The imbalanced distribution of labels, with one class dominating the others, posed a significant challenge for the models. Even after utilizing techniques like stratified k-fold cross-validation and experimenting with various algorithms and hyperparameters to improve classification performance, the accuracy remained the same.

With the understanding that further improvements may be limited due to the inherent class imbalance in the dataframe, the next step was to utilize deep learning techniques to try and improve the accuracy.

4.5.2 Deep Learning Algorithm with LDA

LSTM

Before integrating LDA with the LSTM network, we first need to prepare the textual data by performing tokenization and padding operations to transform the raw text into a format suitable for the model. These preparatory steps are essential for the LSTM network to effectively understand and learn from the complex patterns and structures inherent in textual data.

First, we use the trained LDA model for the topic distributions. These topic distributions are stored in a matrix format, where each row represents a document, and each column corresponds to a specific topic. This document-topic matrix captures the underlying thematic structures within our text data. Next, we perform tokenization and padding, where the tokens from the documents will be converted into numerical representation and padding ensures that all the sequences have the same length, which is crucial for any deep learning model. These tokenized and padded sequences, along with the document-topic distributions are the input features for the LSTM-LDA network. By combining topic information and token sequences, the aim is to leverage both semantic and syntactic aspects of the text data to enhance the performance of the deep learning model.

Before performing any analysis on the data, the first step is to split the data into train and test sets. To do this, "StratifiedShuffleSplit" was utilized as this ensures both stratification and randomness. A single split was performed on the data to create a test set that was 20% of the data. The training set which consists of 80% of the data, was further split using the same method into training and validation sets, where the validation set comprised 25% of the data and the training set consisted of 75% of the data. All three sets include both text data, represented as tokenized and padded sequences, and corresponding labels. Additionally, the LDA topic distributions were also incorporated, which provide insights into the latent topics within the text.

After splitting our dataset into appropriate subsets for training, validation, and testing, the next step was to prepare the data for integration into the deep learning model. This involved creating custom datasets and data loaders tailored to our specific requirements. Three separate datasets were created: the training dataset, the validation dataset, and the test dataset. Each dataset was equipped with the necessary components: the text data, the corresponding labels, and the LDA topic distributions. To efficiently feed these datasets into the model during training and evaluation, we established data loaders. These data loaders divide the datasets into batches, allowing for batch-wise processing, which is particularly useful when for large volumes of data. For training, the batch size was set to 64 and enabled the shuffling of the training data to promote randomness during each epoch. In contrast, during validation and testing, shuffling was disabled to ensure consistent and reproducible evaluations.

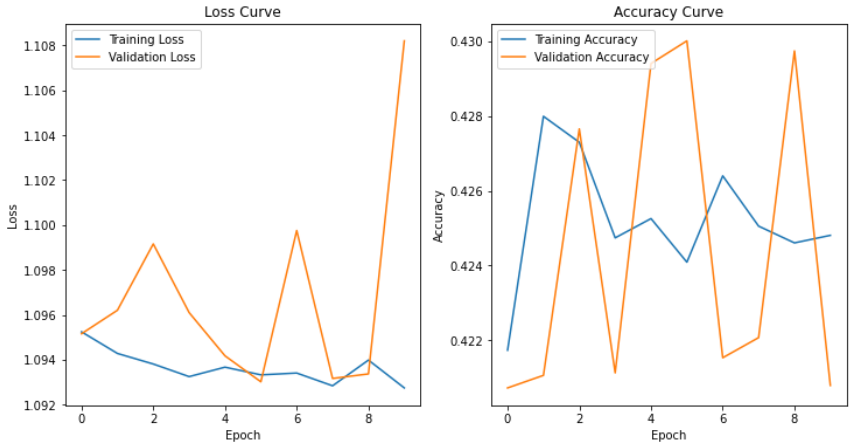

Figure 4.33: Graphs for Loss and Accuracy of the LDA-LSTM Model over the Epochs

The LDA-LSTM model is designed to leverage both the semantic information embedded in text data and the contextual information provided by LDA topic distributions. This fusion allows the model to capture intricate patterns and relationships within the text, making it particularly well-suited for tasks involving text classification and keyword generation. The model begins with an embedding layer which transforms the input text into dense vectors of fixed dimensions. This layer learns to represent words in a continuous space, enabling the model to understand the contextual meaning of words in the input text. The LSTM layer processes the embedded text data. LSTM networks are well-suited for sequential data like text. In this model, the LSTM layer captures the sequential information present in the text. After processing the text data, the LSTM layer produces an output. This output is concatenated with the LDA topic distributions. Combining these two allows the model to consider both the linguistic context and the topics present in the text. The model has two output layers, The first output layer predicts the severity of the text, classifying it into one of four classes and the second output layer generates keywords related to the input text. The SoftMax function is applied to produce a probability distribution over a predefined set of keywords. The model is trained to minimize a cross-entropy loss function for the severity prediction task. For optimization, the Adam optimizer is employed with a learning rate of 0.001625 and specific beta values, (0.9, 0.999), as these provided the best results. During inference, text is passed into the model, which outputs the severity prediction and a set of keywords related to the text.

The LDA-LSTM model was then trained for 10 epochs on the training and validation set. After each epoch, the accuracy and loss for both the training set and the validation set were calculated and stored in separate lists, Using these lists the accuracy over each epoch and loss over each epoch graphs were plotted for both the training set and the validation set. As the graphs in figure 4.33 indicate, the accuracy for the validation set fluctuates but still does not exceed the highest accuracy of 43%, which it reaches to on the 6th epoch. Similarly, for the training set the accuracy reached 42.8%, which is the highest, at the 2nd epoch. The training loss, although it fluctuates eventually decreases and goes down to 1.0927 on the 10th epoch. The validation loss, on the other hand, fluctuates a lot and reached its lowest value, 1.09301 at the 6th epoch but by the 10th epoch has increased to 1.10820. After conducting rigorous hyperparameter tuning and training, the present results are the best results obtainable by the current model on this data. The class imbalance within the dataset posed

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2241
1	0.52	0.06	0.11	6218
2	0.00	0.00	0.00	310
3	0.42	0.97	0.58	6084
accuracy			0.42	14853
macro avg	0.23	0.26	0.17	14853
weighted avg	0.39	0.42	0.29	14853

Figure 4.34: Classification Report for the LDA-LSTM Model

a significant challenge, and despite the efforts to mitigate its impact through various techniques, the achieved accuracy stagnated at this level.

Now we will predict the trained LDA-LSTM network using the test set. First, the severity will be predicted followed by the prediction of the keywords. To predict the severity on the trained model is used to predict the severity from the test set, where the severity is predicted based on the text. From the classification report in figure 4.34, we can see that the overall accuracy of the model is 42%, which means that the model correctly classified 42% of the total samples in the test set. Class 0 (label 0) and Class 2 (label 2) both have 0 precision and recall scores, indicating that the model did not correctly identify any samples belonging to these classes. Class 1 (label 1) has a precision of 52%, indicating that when the model predicts samples as class 1 (label 1), it is correct 52% of the time. However, there is still a significant number of false positives for this class. Class 1 (label 1) has a recall of 6%, meaning that only 6% of the true class 1 (label 1) samples were correctly identified by the model. There is a high rate of false negatives for this class as well. Class 3 (label 3) has a precision of 42%, suggesting that when the model predicts samples as class 3 (label 3), it is correct 42% of the time. Class 3 (label 3) has a recall of 97%, indicating that the model correctly identified 97% of the true class 3 (label 3) samples. The F1 score for class 0 (label 0) and class 2 (label 2) is 0, which is the lowest possible value. This reflects the model's inability to correctly classify these classes. Class 1 (label 1) has an F1 score of 0.11, indicating a low balance between precision and recall for this class. Class 3 (label 3) has the highest F1 score at 0.58, suggesting a relatively better balance between precision and recall. The macro-average F1-score (0.17) is the average of the F1-scores for all classes. It gives equal weight to each class, regardless of class size. The weighted average F1-score (0.29) considers class imbalance by giving more weight to classes with more samples. In summary, this classification report indicates that the model has challenges in correctly classifying samples, particularly for classes 0, 1, and 2. Class 3 has the highest recall and F1 score, suggesting that the model performs relatively better on this class.

Finally, the Receiver Operating Characteristic (ROC) curve graph, figure 4.35, is plotted to visualize the trade-off between the True Positive Rate and the False Positive Rate for different classification thresholds. As we can see from the graph, the area under the curve (AUC) value for class 0 (label 0) and class 2 (label 2) is 0.50 and the AUC value for class 1 (label 1) and class 3 (label 3) is 0.51, this is no better than random guessing, in other words the model's ability to distinguish between the true positives and false positives for these classes is limited. The micro average AUC has a value of 0.61, which indicates that considering all the classes together, the model's distinguishing power is slightly better than random guessing. However, it's still not very high, suggesting that the model's performance in distinguishing between different classes is limited.

The LDA-LSTM model returns the severity label and keyword prediction. Now to find the key-

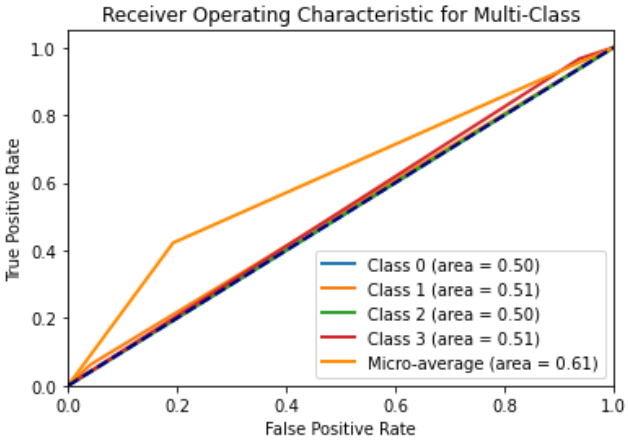

Figure 4.35: ROC Curve for LDA-LSTM Model

```
Unique Values in Order: ['cloud', 'wildcard', 'control', 'add', 'try', 'popup', 'signaling', 'traffic', 'malicious', 'create',
'vcenter', 'expired', 'victims', 'backports', 'neighbor', 'packets']
```

Figure 4.36: Keywords Predicted by the LDA-LSTM Model

word based on the severity it predicted, we have to map the token id back to its corresponding token to make it human-readable. Figure 4.36 shows the unique tokens for the severity predicted, and are in the order they appeared. Similarly, we can view the predicted severity and the keyword associated in figure 4.37, although this only makes predictions for 64 values as the size of our test data loader is 64. This pair of prediction and severity is stored in a dictionary and as we can see it predicts everything of severity 3, which is due to the class imbalance and the keyword it predicts the keyword ”cloud” the most number of times, and the only other keyword it is predicting is the word ”wildcard”,

```
{0: {'severity': 3, 'keywords': 'cloud'},
1: {'severity': 3, 'keywords': 'cloud'},
2: {'severity': 3, 'keywords': 'cloud'},
3: {'severity': 3, 'keywords': 'cloud'},
4: {'severity': 3, 'keywords': 'cloud'},
                .
                .
                .

25: {'severity': 3, 'keywords': 'cloud'},
26: {'severity': 3, 'keywords': 'cloud'},
27: {'severity': 3, 'keywords': 'wildcard'},
28: {'severity': 3, 'keywords': 'cloud'},
29: {'severity': 3, 'keywords': 'cloud'},
                .
                .
                .
60: {'severity': 3, 'keywords': 'cloud'},
61: {'severity': 3, 'keywords': 'cloud'},
62: {'severity': 3, 'keywords': 'cloud'},
63: {'severity': 3, 'keywords': 'wildcard'}}
```

Figure 4.37: Predicted Severity with Associated Keyword

4.6 Conclusion

In conclusion, the extensive exploration of various models has demonstrated their capability to effectively classify cybersecurity data, albeit with certain limitations. A persistent challenge that emerged in the analysis was the imbalance in the severity classes. Despite employing a range of techniques and exhaustive hyperparameter tuning efforts, it was observed that achieving higher accuracy is hindered by this class imbalance. This is where the quintessential nature of this research intertwines with the broader cybersecurity narrative.

Cybersecurity by its very nature is marked by its constant evolution and the asymmetric nature of the threats it combats. Vulnerabilities in digital systems emerge unpredictably and attackers relentlessly seek to exploit them. In such an environment, a model's accuracy and effectiveness can mean the difference between safeguarding sensitive data and suffering network breaches.

One striking revelation from this analysis is the undeniable upwards trend in the frequency and severity of vulnerabilities. Attacks that were once considered rare have increased due to the rise in these vulnerabilities. These attacks have evolved from being nuisances to large-scale assaults on critical infrastructure and sensitive data repositories. This analysis underscored the reality that while our models exhibit moderate performance in predicting the severity, their true potential is restricted by the class imbalance.

As a result, the analysis is concluded with the realization that cybersecurity attacks are a threat that continue to increase over the years and will likely do so. Although the models do not return high accuracy, they still provide keywords associated with the severity level, which is useful in understanding the nature of the vulnerability. The class imbalance in the data is not merely a hurdle but a cybersecurity challenge themselves. This serves as a valuable insight for future work, significance of addressing class imbalances which will allow for a the models to reach their full potential and help in fortifying the defences in the systems to prevent any cybersecurity attacks.

Bibliography

[1] *National Vulnerability Database*. (accessed: 01-09-2023).

[2] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[3] Adji B Dieng et al. "Topicrnn: A recurrent neural network with long-range semantic dependency". In: *arXiv preprint arXiv:1611.01702* (2016).

[4] Wafa Shafqat et al. "A Hybrid Approach for Topic Discovery and Recommendations Based on Topic Modeling and Deep Learning". PhD thesis. 2020.

[5] Yee Whye Teh et al. "Hierarchical Dirichlet Processes". In: *Journal of the American Statistical Association* 101.476 (2006), pp. 1566–1581. (Visited on 09/02/2023).

[6] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[7] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[8] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58.1 (1996), pp. 267–288.

[9] Arthur E Hoerl and Robert W Kennard. "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 12.1 (1970), pp. 55–67.

[10] Hui Zou and Trevor Hastie. "Regularization and variable selection via the elastic net". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 67.2 (2005), pp. 301–320.

[11] Wei-Yin Loh. "Classification and regression trees". In: *Wiley interdisciplinary reviews: data mining and knowledge discovery* 1.1 (2011), pp. 14–23.

[12] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[13] Vladimir N. Vapnik and Corinna Cortes. "Support-Vector Networks". In: *Machine Learning* 20.3 (1995), pp. 273–297.

[14] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[15] Jürgen Schmidhuber and AI Blog. "1. 1997 LSTM journal paper". In: ().

Appendix

```
import os
import numpy as np
import json
import random
import math
import string
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib.ticker import FuncFormatter

from collections import Counter
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
    recall_score
from sklearn.metrics import f1_score as f1_metric
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import FeatureUnion
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.manifold import TSNE
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import label_binarize

from bokeh.plotting import figure, output_notebook, show
from bokeh.models import ColumnDataSource, ColorBar, HoverTool
from bokeh.transform import linear_cmap

from scipy.stats import gaussian_kde

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline as ImbPipeline

from gensim.models import LdaModel
from gensim.models import HdpModel
from gensim.matutils import Sparse2Corpus
from gensim.corpora.dictionary import Dictionary
from gensim.models.coherencemodel import CoherenceModel

import torch
from torch.utils.data import Dataset, DataLoader
from torch.utils.data import random_split
import torch.nn as nn
import torch.nn.functional as F

### 2022

with open("C:/V/Notes/Dissertation/nvdcve-1.1-2022/nvdcve
    -1.1-2022.json", 'r', encoding = 'utf-8') as json_file:

    data = json.load(json_file)

df_flat = pd.json_normalize(data, record_path = ['CVE_Items'],
    meta = ['CVE_data_timestamp']) #normalizing it, atleast trying

#problemtype and description still had lists inside columns so had
```

```
    to normalize it
problemtype_data = df_flat['cve.problemtype.problemtype_data']
problemtype_flattened = []
for item in problemtype_data:
    if isinstance(item, list):
        for subitem in item:
            problemtype_flattened.append(subitem)
    else:
        problemtype_flattened.append(item)

df_problemtype = pd.json_normalize(problemtype_flattened)
df_problemtype.columns = [f'problemtype.{col}' for col in
    df_problemtype.columns]

description_data = df_flat['cve.description.description_data']
description_flattened = []
for item in description_data:
    if isinstance(item, list):
        for subitem in item:
            description_flattened.append(subitem)
    else:
        description_flattened.append(item)

df_description = pd.json_normalize(description_flattened)
df_description.columns = [f'description.{col}' for col in
    df_description.columns]

df_flat = pd.concat([df_flat.drop(['cve.problemtype.
    problemtype_data', 'cve.description.description_data'], axis =
    1), df_problemtype, df_description], axis = 1)

cols = ["publishedDate", "lastModifiedDate", "cve.CVE_data_meta.ID
    ", "problemtype.description", "description.value", "impact.
    baseMetricV3.cvssV3.baseScore", "impact.baseMetricV3.cvssV3.
    baseSeverity"]
df_super = df_flat.loc[:, cols].copy() #took only the columns i
    needed, though i did not end up using problemtype and basescore

problemtype_data_new = df_super['problemtype.description']
problemtype_flattened_new = []
for item in problemtype_data_new:
    if isinstance(item, list) and len(item) > 0:
        problemtype_flattened_new.append(item[0])
```

```
    else:
        problemtype_flattened_new.append({})

df_problemtype_new = pd.json_normalize(problemtype_flattened_new)
df_problemtype_new.columns = [f'problemtype.{col}' for col in
    df_problemtype_new.columns]

df_super['problemtype.description'] = df_super['problemtype.
    description'].apply(lambda x: x[0]['value'] if isinstance(x,
    list) and len(x) > 0 else '')

df_problemtype_new = pd.DataFrame(df_super['problemtype.
    description'], columns = ['problemtype.description'])

df_super = df_super.rename(columns = {'impact.baseMetricV3.cvssV3.
    baseScore': 'BaseScore_V3'})
df_super = df_super.rename(columns = {'impact.baseMetricV3.cvssV3.
    baseSeverity': 'BaseSeverity_V3'})
df_super = df_super.rename(columns = {'description.value': '
    description'})
df_super = df_super.rename(columns = {'problemtype.description': '
    problemtype'})
df_super = df_super.rename(columns = {'cve.CVE_data_meta.ID': '
    CVE_ID'})

df_super.isnull().sum() #so there are 732 nulls

df_super.dropna(inplace = True) #dropped

df_super.duplicated().sum() #no duplicates

#fixing the columns with date-time
pub_date = df_super["publishedDate"]

date_list = []
time_list = []

for x in pub_date:
    dat = x.split('T')[0]
    tim = x.split("T")[1]
    date_list.append(dat)
    time_list.append(tim)
```

```
df_super["Published_Date"] = date_list

mod_date = df_super["lastModifiedDate"]

date_list_1 = []
time_list_1 = []

for x in mod_date:
    dat_1 = x.split('T')[0]
    tim_1 = x.split("T")[1]
    date_list_1.append(dat_1)
    time_list_1.append(tim_1)

df_super["Modified_Date"] = date_list_1

df_super = df_super.drop(["publishedDate", "lastModifiedDate"],
   axis = 1)

exp_1 = df_super["Published_Date"]

date_month = []
for x in exp_1:
    y = x[:-3]
    date_month.append(y)

df_super["Published_date"]= date_month

exp_2 = df_super["Published_Date"]

date_month_2 = []
for x in exp_2:
    y = x[:-3]
    date_month_2.append(y)

df_super["Modified_date"]= date_month_2

df_super = df_super.drop(["Modified_Date", "Published_Date"], axis
    = 1)

df_super = df_super.drop(["Modified_date"], axis = 1)

df_super.head() #cleaned dataframe
```

```
print(df_super.shape)

df_super_v = df_super.drop(df_super[df_super["Published_date"].
    astype(str).str.contains("2023")].index)

print(df_super_v.shape)

severity_counts_v = df_super_v['BaseSeverity_V3'].value_counts()
severity_counts_v = severity_counts_v.sort_index()

plt.bar(severity_counts_v.index, severity_counts_v)
plt.xlabel('Severity')
plt.ylabel('Count')
plt.title('Severity Counts')

for index, count in enumerate(severity_counts_v):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()

month_counts_v = df_super_v['Published_date'].value_counts()
month_counts_v = month_counts_v.sort_index()

plt.figure(figsize = (10, 6))
plt.bar(month_counts_v.index, month_counts_v)
plt.xlabel('month')
plt.ylabel('Count')
plt.title('month Counts')
plt.xticks(rotation=90)

for index, count in enumerate(month_counts_v):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()

### 2021

with open("C:/V/Notes/Dissertation/nvdcve-1.1-2021/nvdcve
    -1.1-2021.json", 'r', encoding = 'utf-8') as json_file:
    data1 = json.load(json_file)
```

```
df_flat_21 = pd.json_normalize(data1, record_path = ['CVE_Items'],
    meta = ['CVE_data_timestamp'])

problemtype_data = df_flat_21['cve.problemtype.problemtype_data']
problemtype_flattened = []
for item in problemtype_data:
    if isinstance(item, list):
        for subitem in item:
            problemtype_flattened.append(subitem)
    else:
        problemtype_flattened.append(item)

df_problemtype = pd.json_normalize(problemtype_flattened)
df_problemtype.columns = [f'problemtype.{col}' for col in
    df_problemtype.columns]

description_data = df_flat_21['cve.description.description_data']
description_flattened = []
for item in description_data:
    if isinstance(item, list):
        for subitem in item:
            description_flattened.append(subitem)
    else:
        description_flattened.append(item)

df_description = pd.json_normalize(description_flattened)
df_description.columns = [f'description.{col}' for col in
    df_description.columns]

df_flat_21 = pd.concat([df_flat_21.drop(['cve.problemtype.
    problemtype_data', 'cve.description.description_data'], axis =
    1), df_problemtype, df_description], axis = 1)

cols = ["publishedDate", "lastModifiedDate", "cve.CVE_data_meta.ID
    ", "problemtype.description", "description.value", "impact.
    baseMetricV3.cvssV3.baseScore", "impact.baseMetricV3.cvssV3.
    baseSeverity"]
df_super_21 = df_flat_21.loc[:, cols].copy()

problemtype_data_new = df_super_21['problemtype.description']
problemtype_flattened_new = []
for item in problemtype_data_new:
    if isinstance(item, list) and len(item) > 0:
```

```
            problemtype_flattened_new.append(item[0])
        else:
            problemtype_flattened_new.append({})

df_problemtype_new = pd.json_normalize(problemtype_flattened_new)
df_problemtype_new.columns = [f'problemtype.{col}' for col in
    df_problemtype_new.columns]

df_super_21['problemtype.description'] = df_super_21['problemtype.
    description'].apply(lambda x: x[0]['value'] if isinstance(x,
    list) and len(x) > 0 else '')

df_problemtype_new = pd.DataFrame(df_super_21['problemtype.
    description'], columns=['problemtype.description'])

df_super_21 = df_super_21.rename(columns = {'impact.baseMetricV3.
    cvssV3.baseScore': 'BaseScore_V3'})
df_super_21 = df_super_21.rename(columns = {'impact.baseMetricV3.
    cvssV3.baseSeverity': 'BaseSeverity_V3'})
df_super_21 = df_super_21.rename(columns = {'description.value': '
    description'})
df_super_21 = df_super_21.rename(columns = {'problemtype.
    description': 'problemtype'})
df_super_21 = df_super_21.rename(columns = {'cve.CVE_data_meta.ID'
    : 'CVE_ID'})

df_super_21.isnull().sum()

df_super_21.dropna(inplace = True)

df_super_21.duplicated().sum()

pub_date_21 = df_super_21["publishedDate"]

date_list_21 = []
time_list_21 = []

for x in pub_date_21:
    dat = x.split('T')[0]
    tim = x.split("T")[1]
    date_list_21.append(dat)
    time_list_21.append(tim)
```

```
df_super_21["Published_Date"] = date_list_21

mod_date_21 = df_super_21["lastModifiedDate"]

date_list_1_21 = []
time_list_1_21 = []

for x in mod_date_21:
    dat_1 = x.split('T')[0]
    tim_1 = x.split("T")[1]
    date_list_1_21.append(dat_1)
    time_list_1_21.append(tim_1)

df_super_21["Modified_Date"] = date_list_1_21

df_super_21 = df_super_21.drop(["publishedDate", "lastModifiedDate
    "], axis = 1)

exp_1_21 = df_super_21["Published_Date"]

date_month_21 = []
for x in exp_1_21:
    y = x[:-3]
    date_month_21.append(y)

df_super_21["Published_date"]= date_month_21

df_super_21 = df_super_21.drop(["Modified_Date", "Published_Date"
    ], axis = 1)

df_super_21_v = df_super_21[df_super_21['Published_date'].str.
    startswith('2021')]

print(df_super_21.shape)

print(df_super_21_v.shape)

severity_counts_21_v = df_super_21_v['BaseSeverity_V3'].
    value_counts()
severity_counts_21_v = severity_counts_21_v.sort_index()

plt.bar(severity_counts_21_v.index, severity_counts_21_v)
plt.xlabel('Severity')
```

```
plt.ylabel('Count')
plt.title('Severity Counts')

for index, count in enumerate(severity_counts_21_v):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()

month_counts_21_v = df_super_21_v['Published_date'].value_counts()
month_counts_21_v = month_counts_21_v.sort_index()

plt.figure(figsize = (10, 6))
plt.bar(month_counts_21_v.index, month_counts_21_v)
plt.xlabel('month')
plt.ylabel('Count')
plt.title('month Counts')
plt.xticks(rotation=90)

for index, count in enumerate(month_counts_21_v):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()

### 2020

with open("C:/V/Notes/Dissertation/nvdcve-1.1-2020/nvdcve
    -1.1-2020.json", 'r', encoding = 'utf-8') as json_file:
    data2 = json.load(json_file)

df_flat_20 = pd.json_normalize(data2, record_path = ['CVE_Items'],
    meta = ['CVE_data_timestamp'])

problemtype_data = df_flat_20['cve.problemtype.problemtype_data']
problemtype_flattened = []
for item in problemtype_data:
    if isinstance(item, list):
        for subitem in item:
            problemtype_flattened.append(subitem)
    else:
        problemtype_flattened.append(item)
```

```
df_problemtype = pd.json_normalize(problemtype_flattened)
df_problemtype.columns = [f'problemtype.{col}' for col in
    df_problemtype.columns]

description_data = df_flat_20['cve.description.description_data']
description_flattened = []
for item in description_data:
    if isinstance(item, list):
        for subitem in item:
            description_flattened.append(subitem)
    else:
        description_flattened.append(item)

df_description = pd.json_normalize(description_flattened)
df_description.columns = [f'description.{col}' for col in
    df_description.columns]

df_flat_20 = pd.concat([df_flat_20.drop(['cve.problemtype.
    problemtype_data', 'cve.description.description_data'], axis =
    1), df_problemtype, df_description], axis = 1)

cols = ["publishedDate", "lastModifiedDate", "cve.CVE_data_meta.ID
    ", "problemtype.description", "description.value", "impact.
    baseMetricV3.cvssV3.baseScore", "impact.baseMetricV3.cvssV3.
    baseSeverity"]
df_super_20 = df_flat_20.loc[:, cols].copy()

problemtype_data_new = df_super_20['problemtype.description']
problemtype_flattened_new = []
for item in problemtype_data_new:
    if isinstance(item, list) and len(item) > 0:
        problemtype_flattened_new.append(item[0])
    else:
        problemtype_flattened_new.append({})

df_problemtype_new = pd.json_normalize(problemtype_flattened_new)
df_problemtype_new.columns = [f'problemtype.{col}' for col in
    df_problemtype_new.columns]

df_super_20['problemtype.description'] = df_super_20['problemtype.
    description'].apply(lambda x: x[0]['value'] if isinstance(x,
    list) and len(x) > 0 else '')
```

```
df_problemtype_new = pd.DataFrame(df_super_20['problemtype.
    description'], columns = ['problemtype.description'])

df_super_20 = df_super_20.rename(columns = {'impact.baseMetricV3.
    cvssV3.baseScore': 'BaseScore_V3'})
df_super_20 = df_super_20.rename(columns = {'impact.baseMetricV3.
    cvssV3.baseSeverity': 'BaseSeverity_V3'})
df_super_20 = df_super_20.rename(columns = {'description.value': '
    description'})
df_super_20 = df_super_20.rename(columns = {'problemtype.
    description': 'problemtype'})
df_super_20 = df_super_20.rename(columns = {'cve.CVE_data_meta.ID'
    : 'CVE_ID'})

df_super_20.isnull().sum()

df_super_20.dropna(inplace = True)

df_super_20.duplicated().sum()

pub_date_20 = df_super_20["publishedDate"]

date_list_20 = []
time_list_20 = []

for x in pub_date_20:
    dat = x.split('T')[0]
    tim = x.split("T")[1]
    date_list_20.append(dat)
    time_list_20.append(tim)

df_super_20["Published_Date"] = date_list_20

df_super_20 = df_super_20.drop(["publishedDate", "lastModifiedDate
    "], axis = 1)

exp_1_20 = df_super_20["Published_Date"]

date_month_20 = []
for x in exp_1_20:
    y = x[:-3]
    date_month_20.append(y)
```

```
df_super_20["Published_date"] = date_month_20

df_super_20 = df_super_20.drop(["Published_Date"], axis = 1)

df_super_20_v = df_super_20[df_super_20['Published_date'].str.
    startswith('2020')]

print(df_super_20.shape)

print(df_super_20_v.shape)

severity_counts_20_v = df_super_20_v['BaseSeverity_V3'].
    value_counts()
severity_counts_20_v = severity_counts_20_v.sort_index()

plt.bar(severity_counts_20_v.index, severity_counts_20_v)
plt.xlabel('Severity')
plt.ylabel('Count')
plt.title('Severity Counts')

for index, count in enumerate(severity_counts_20_v):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()

month_counts_20_v = df_super_20_v['Published_date'].value_counts()
month_counts_20_v = month_counts_20_v.sort_index()

plt.figure(figsize = (10, 6))
plt.bar(month_counts_20_v.index, month_counts_20_v)
plt.xlabel('month')
plt.ylabel('Count')
plt.title('month Counts')
plt.xticks(rotation = 90)

for index, count in enumerate(month_counts_20_v):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()

### 2019
```

```
with open("C:/V/Notes/Dissertation/nvdcve-1.1-2019/nvdcve
    -1.1-2019.json", 'r', encoding = 'utf-8') as json_file:
    data3 = json.load(json_file)

df_flat_19 = pd.json_normalize(data3, record_path = ['CVE_Items'],
    meta = ['CVE_data_timestamp'])

problemtype_data = df_flat_19['cve.problemtype.problemtype_data']
problemtype_flattened = []
for item in problemtype_data:
    if isinstance(item, list):
        for subitem in item:
            problemtype_flattened.append(subitem)
    else:
        problemtype_flattened.append(item)

df_problemtype = pd.json_normalize(problemtype_flattened)
df_problemtype.columns = [f'problemtype.{col}' for col in
    df_problemtype.columns]

description_data = df_flat_19['cve.description.description_data']
description_flattened = []
for item in description_data:
    if isinstance(item, list):
        for subitem in item:
            description_flattened.append(subitem)
    else:
        description_flattened.append(item)

df_description = pd.json_normalize(description_flattened)
df_description.columns = [f'description.{col}' for col in
    df_description.columns]

df_flat_19 = pd.concat([df_flat_19.drop(['cve.problemtype.
    problemtype_data', 'cve.description.description_data'], axis =
    1), df_problemtype, df_description], axis = 1)

cols = ["publishedDate", "lastModifiedDate", "cve.CVE_data_meta.ID
    ", "problemtype.description", "description.value", "impact.
    baseMetricV3.cvssV3.baseScore", "impact.baseMetricV3.cvssV3.
    baseSeverity"]
df_super_19 = df_flat_19.loc[:, cols].copy()
```

```
problemtype_data_new = df_super_19['problemtype.description']
problemtype_flattened_new = []
for item in problemtype_data_new:
    if isinstance(item, list) and len(item) > 0:
        problemtype_flattened_new.append(item[0])
    else:
        problemtype_flattened_new.append({})

df_problemtype_new = pd.json_normalize(problemtype_flattened_new)
df_problemtype_new.columns = [f'problemtype.{col}' for col in
    df_problemtype_new.columns]

df_super_19['problemtype.description'] = df_super_19['problemtype.
    description'].apply(lambda x: x[0]['value'] if isinstance(x,
    list) and len(x) > 0 else '')

df_problemtype_new = pd.DataFrame(df_super_19['problemtype.
    description'], columns = ['problemtype.description'])

df_super_19 = df_super_19.rename(columns = {'impact.baseMetricV3.
    cvssV3.baseScore': 'BaseScore_V3'})
df_super_19 = df_super_19.rename(columns = {'impact.baseMetricV3.
    cvssV3.baseSeverity': 'BaseSeverity_V3'})
df_super_19 = df_super_19.rename(columns = {'description.value': '
    description'})
df_super_19 = df_super_19.rename(columns = {'problemtype.
    description': 'problemtype'})
df_super_19 = df_super_19.rename(columns = {'cve.CVE_data_meta.ID'
    : 'CVE_ID'})

df_super_19.isnull().sum()

df_super_19.dropna(inplace = True)

df_super_19.duplicated().sum()

pub_date_19 = df_super_19["publishedDate"]

date_list_19 = []
time_list_19 = []

for x in pub_date_19:
```

```
    dat = x.split('T')[0]
    tim = x.split("T")[1]
    date_list_19.append(dat)
    time_list_19.append(tim)

df_super_19["Published_Date"] = date_list_19

df_super_19 = df_super_19.drop(["publishedDate", "lastModifiedDate
    "], axis = 1)

exp_1_19 = df_super_19["Published_Date"]

date_month_19 = []
for x in exp_1_19:
    y = x[:-3]
    date_month_19.append(y)

df_super_19["Published_date"] = date_month_19

df_super_19 = df_super_19.drop(["Published_Date"], axis = 1)

df_super_19_v = df_super_19[df_super_19['Published_date'].str.
    startswith('2019')]

print(df_super_19.shape)

print(df_super_19_v.shape)

severity_counts_19_v = df_super_19_v['BaseSeverity_V3'].
    value_counts()
severity_counts_19_v = severity_counts_19_v.sort_index()

plt.bar(severity_counts_19_v.index, severity_counts_19_v)
plt.xlabel('Severity')
plt.ylabel('Count')
plt.title('Severity Counts')

for index, count in enumerate(severity_counts_19_v):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()
```

```
month_counts_19_v = df_super_19_v['Published_date'].value_counts()
month_counts_19_v = month_counts_19_v.sort_index()

plt.figure(figsize = (10, 6))
plt.bar(month_counts_19_v.index, month_counts_19_v)
plt.xlabel('month')
plt.ylabel('Count')
plt.title('month Counts')
plt.xticks(rotation = 90)

for index, count in enumerate(month_counts_19_v):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()

fig, axes = plt.subplots(4, 2, figsize = (20, 24))

month_data = [month_counts_v, month_counts_21_v, month_counts_20_v
    , month_counts_19_v]
severity_data = [severity_counts_v, severity_counts_21_v,
    severity_counts_20_v, severity_counts_19_v]
titles_months = ['Month Counts 22', 'Month Counts 21', 'Month
    Counts 20', 'Month Counts 19']
titles_severity = ['Severity Counts 22', 'Severity Counts 21', '
    Severity Counts 20', 'Severity Counts 19']

for i in range(4):

    axes[i, 0].bar(month_data[i].index, month_data[i])
    axes[i, 0].set_title(titles_months[i])
    axes[i, 0].set_xlabel('Month')
    axes[i, 0].set_ylabel('Count')
    axes[i, 0].tick_params(axis = 'x', rotation = 90)

    for index, count in enumerate(month_data[i]):
        axes[i, 0].text(index, count, str(count), ha = 'center',
            va = 'bottom', fontsize = 10)



    axes[i, 1].bar(severity_data[i].index, severity_data[i])
    axes[i, 1].set_title(titles_severity[i])
    axes[i, 1].set_xlabel('Severity')
```

```
    axes[i, 1].set_ylabel('Count')
    axes[i, 1].tick_params(axis = 'x')

    for index, count in enumerate(severity_data[i]):
        axes[i, 1].text(index, count, str(count), ha = 'center',
            va = 'bottom', fontsize = 10)

plt.tight_layout()
plt.show()

### together

df_mega = pd.concat([df_super, df_super_21, df_super_20,
    df_super_19], ignore_index = True)

print(df_mega.shape)

df_mega = df_mega.drop(df_mega[df_mega["Published_date"].astype(
    str).str.contains("2023")].index)

df_mega.info()

df_mega.head()

df_mega_2 = df_mega.copy()

df_mega_2['Year'] = pd.DatetimeIndex(df_mega_2['Published_date']).
    year


grouped_by_year_severity = df_mega_2.groupby(['Year', '
    BaseSeverity_V3']).size().reset_index(name = 'Counts')


df_mega_2['Month'] = pd.DatetimeIndex(df_mega_2['Published_date'])
    .month
grouped_by_year_month = df_mega_2.groupby(['Year', 'Month']).size
    ().reset_index(name = 'Counts')

unique_years = grouped_by_year_severity['Year'].unique()

fig, axes = plt.subplots(len(unique_years), 1, figsize = (10, 5 *
    len(unique_years)))
```

```
for i, year in enumerate(unique_years):
    ax = axes[i]
    subset_data = grouped_by_year_severity[
        grouped_by_year_severity['Year'] == year]
    bars = ax.bar(subset_data['BaseSeverity_V3'], subset_data['
        Counts'])
    ax.set_title(f'Severity Counts for {year}')
    ax.set_xlabel('BaseSeverity_V3')
    ax.set_ylabel('Counts')

    for bar in bars:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2, yval, int(yval),
            ha = 'center', va = 'bottom')

plt.tight_layout()
plt.show()

fig, axes = plt.subplots(len(unique_years), 1, figsize=(10, 5 *
    len(unique_years)))

for i, year in enumerate(unique_years):
    ax = axes[i]
    subset_data = grouped_by_year_month[grouped_by_year_month['
        Year'] == year]
    bars = ax.bar(subset_data['Month'], subset_data['Counts'])
    ax.set_title(f'Month Counts for {year}')
    ax.set_xlabel('Month')
    ax.set_ylabel('Counts')

    for bar in bars:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2, yval, int(yval),
            ha = 'center', va = 'bottom')

plt.tight_layout()
plt.show()



severity_counts_mega = df_mega['BaseSeverity_V3'].value_counts()
severity_counts_mega = severity_counts_mega.sort_index()
```

```
plt.bar(severity_counts_mega.index, severity_counts_mega)
plt.xlabel('Severity')
plt.ylabel('Count')
plt.title('Severity Counts')

for index, count in enumerate(severity_counts_mega):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()

month_counts_mega = df_mega['Published_date'].value_counts()
month_counts_mega = month_counts_mega.sort_index()

plt.figure(figsize = (15, 8))
plt.bar(month_counts_mega.index, month_counts_mega)
plt.xlabel('month')
plt.ylabel('Count')
plt.title('month Counts')
plt.xticks(rotation = 90)

for index, count in enumerate(month_counts_mega):
    plt.text(index, count, str(count), ha = 'center', va = 'bottom
        ', fontsize = 10)

plt.show()

df_mega_2 = df_mega.copy()

### text

def preprocess_text(text):
    stop_words = set(stopwords.words('english'))

    stop_words.add('could') #because it came up as one of the
        topics

    text = text.lower()

    text = ''.join(char for char in text if char.isalnum() or char
        .isspace())
```

```
    text = text.replace('vulnerability', '')

    tokens = word_tokenize(text)
    tokens = [token for token in tokens if token.isalpha() and
        token not in stop_words]

    word_freq = Counter(tokens)

    cleaned_tokens = [token for token in tokens if word_freq[token
        ] >= 3] # reducing the noise

    cleaned_text = ' '.join(cleaned_tokens)

    return cleaned_text, cleaned_tokens

def preprocess_text_2(text):   #just to give me clean sentences,
    like readable clean ones
    stop_words = set(stopwords.words('english'))

    stop_words.add('could')

    text = text.lower()

    text = ''.join(char for char in text if char.isalnum() or char
        .isspace())

    text = text.replace('vulnerability', '')

    tokens = word_tokenize(text)

    tokens = [token for token in tokens if token.isalnum() and
        token not in stop_words]

    cleaned_text = ' '.join(tokens)
    return cleaned_text

df_mega_2["clean_description"],df_mega_2["tokens"]  = zip(*
    df_mega_2["description"].apply(preprocess_text))

df_mega_2["clean_desc"] = df_mega_2["description"].apply(
    preprocess_text_2)

label_encoder = LabelEncoder()
```

```
df_mega_2['label'] = label_encoder.fit_transform(df_mega_2['
    BaseSeverity_V3'])

df_mega_2.head()

## Grouping

df_mega_2['Year'] = pd.to_datetime(df_mega_2['Published_date']).dt
    .year

grouped_by_year = df_mega_2.groupby("Year")

dataframes_by_year = [group for _, group in grouped_by_year]

dataframe_by_severity = df_mega_2.groupby("BaseSeverity_V3")



def LDA_gen(desc, num, num_words = 6, vectorizer_type = 'count'):
    #good for small data

    if vectorizer_type == 'count':
        vectorizer_desc = CountVectorizer()
    elif vectorizer_type == 'tfidf':
        vectorizer_desc = TfidfVectorizer()
    else:
        raise ValueError("Invalid vectorizer_type. Choose 'count'
            or 'tfidf'.")

    # Making DTM
    dtm_desc = vectorizer_desc.fit_transform(desc)

    id2word_desc = {v: k for k, v in vectorizer_desc.vocabulary_.
        items()}

    corpus_desc = Sparse2Corpus(dtm_desc.T)

    # LDA
    lda_model_desc = LdaModel(corpus = corpus_desc, num_topics =
        num, id2word = id2word_desc, passes = 15)

    # Most likely term in each topic
```

```
    topic_terms_desc = lda_model_desc.print_topics(num_topics =
        num, num_words = num_words)

    return lda_model_desc, topic_terms_desc, corpus_desc,
        id2word_desc

def LDA_gen_2(desc, num, num_words = 6, vectorizer_type = 'count')
    : #good for bigger data

    texts = [text.split() for text in desc]

    gensim_dict = Dictionary(texts)

    corpus_desc = [gensim_dict.doc2bow(text) for text in texts]

    lda_model_desc = LdaModel(corpus=corpus_desc, num_topics = num
        , id2word = gensim_dict, passes = 15)

    topic_terms_desc = lda_model_desc.print_topics(num_topics =
        num, num_words = num_words)

    return lda_model_desc, topic_terms_desc, corpus_desc,
        gensim_dict

def compute_coherence_for_each_topic(model, texts, dictionary,
    coherence = 'c_v'):
    """
    Compute c_v coherence for each topic using the HDP model.
    """
    topic_weights = model.get_topics()
    topic_terms = [sorted(enumerate(topic), key = lambda x: x[1],
        reverse = True) for topic in topic_weights]

    topics = []
    for topic in topic_terms:
        topics.append([(dictionary[id], weight) for id, weight in
            topic])

    coherence_values = []

    for topic in topics:
        topic = [word for word, _ in topic]
        cm = CoherenceModel(topics = [topic], texts = texts,
```

```
            dictionary = dictionary, coherence = coherence)
        coherence_values.append(cm.get_coherence())

    return coherence_values

def format_topics_sentences(ldamodel, corpus, texts):

    sent_topics_df = pd.DataFrame(columns = ['Dominant_Topic', '
        Perc_Contribution', 'Topic_Keywords'])

    # Getting main topic in each document
    for i, row_list in enumerate(ldamodel[corpus]):
        row = row_list[0] if ldamodel.per_word_topics else
            row_list
        row = sorted(row, key = lambda x: x[1], reverse = True)

        # Getting the Dominant topic, Perc Contribution, and
            Keywords for each document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # dominant topic
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in
                    wp])
                sent_topics_df.loc[i] = [int(topic_num), round(
                    prop_topic, 4), topic_keywords]
            else:
                break

    # Addding og text to the end of the output
    contents = pd.Series(texts).reset_index(drop = True)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis =
        1).rename(columns = {0: 'Original_Text'})
    return sent_topics_df

def topics_per_document(model, corpus, start = 0, end = 6):
    corpus_sel = corpus[start:end]
    dominant_topics = []
    topic_percentages = []

    for i, corp in enumerate(corpus_sel):
        topic_percs = model[corp]
        dominant_topic = sorted(topic_percs, key = lambda x: x[1],
            reverse = True)[0][0]
```

```
            dominant_topics.append((i, dominant_topic))
            topic_percentages.append(topic_percs)

    return(dominant_topics, topic_percentages)

def convert_to_gensim_dictionary(id2word_desc):

    word2id_desc = {v: k for k, v in id2word_desc.items()}
    gensim_dict = Dictionary()
    gensim_dict.token2id = word2id_desc

    return gensim_dict

### Critical

crtical = dataframe_by_severity.get_group("CRITICAL")



crit_model_desc, crit_model_terms, crit_corpus_dec,
    crit_id2word_desc = LDA_gen(crtical["clean_description"], num =
     10, vectorizer_type = 'tfidf')

crit_model_terms

log_perplexity = crit_model_desc.log_perplexity(crit_corpus_dec)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')

coherence_model_lda = CoherenceModel(model = crit_model_desc,
    texts = crit_texts, dictionary = gensim_dict_crit, coherence =
    'c_v')
coherence_lda = coherence_model_lda.get_coherence()

coherence_lda

topics = crit_model_desc.show_topics(formatted=False)

coherence_values_per_topic = []
```

```
for i, topic in topics:
    cm = CoherenceModel(topics=[dict(crit_model_desc.show_topic(i,
        topn=20))], corpus=crit_corpus_dec,texts = crit_texts,
        dictionary=gensim_dict_crit, coherence='c_v')
    coherence_values_per_topic.append((i, cm.get_coherence()))

print(f'Coherence Scores per Topic: {coherence_values_per_topic}')

gensim_dict_crit = convert_to_gensim_dictionary(crit_id2word_desc)

crit_hdp = HdpModel(crit_corpus_dec, gensim_dict_crit)

crit_all_topics = crit_hdp.get_topics()
crit_num_of_topics = crit_all_topics.shape[0]
print(f"Total number of topics: {crit_num_of_topics}")


for topic_info in crit_hdp.print_topics(num_topics = 15, num_words
    = 5):
    print(topic_info)

crit_texts = crtical['clean_desc'].apply(lambda x: x.split()).
    tolist()

crit_coherence_model_hdp = CoherenceModel(model = crit_hdp, texts
   = crit_texts, dictionary = gensim_dict_crit, coherence = 'c_v')
crit_coherence_hdp = crit_coherence_model_hdp.get_coherence()

print('Coherence Score:', crit_coherence_hdp)

crit_coherence_values = compute_coherence_for_each_topic(crit_hdp,
    crit_texts, gensim_dict_crit)

crit_top_topics = sorted(range(len(crit_coherence_values)), key =
   lambda i: crit_coherence_values[i], reverse = True)[:10]

print("Top 10 Topics based on Coherence Scores:")
for idx in crit_top_topics:
    print("Topic {}: Coherence Score: {:.4f}".format(idx,
        crit_coherence_values[idx]))

#sorted_coherence_values = sorted(coherence_values, reverse=True)
```

```
topics_range = range(1, len(crit_coherence_values) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range, crit_coherence_values, marker = 'o',
    linestyle = '--')
plt.title('Coherence Scores by Number of Topics for Critical
    severity')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

crit_sorted_coherence_values = sorted(crit_coherence_values,
    reverse=True)

topics_range = range(1, len(crit_sorted_coherence_values) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range, crit_sorted_coherence_values, marker = 'o',
     linestyle = '--')
plt.title('Sorted Coherence Scores by Number of Topics for
    Critical severity')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

df_topic_sents_keywords = format_topics_sentences(ldamodel =
    crit_model_desc, corpus = crit_corpus_dec, texts = crtical["
    clean_desc"].tolist())

df_topic_sents_keywords

doc_lens = [len(d.split()) for d in df_topic_sents_keywords.
    Original_Text]

mean_val = round(np.mean(doc_lens))
median_val = round(np.median(doc_lens))
std_val = round(np.std(doc_lens))
quantile_01_val = round(np.quantile(doc_lens, q = 0.01))
quantile_99_val = round(np.quantile(doc_lens, q = 0.99))

plt.figure(figsize = (16,7), dpi = 160)
```

```
plt.hist(doc_lens, bins = 1000, color = 'navy')

y_pos = max(plt.gca().get_ylim()) - 0.05 * max(plt.gca().get_ylim
    ())
spacing = 0.025 * max(plt.gca().get_ylim())

plt.text(mean_val, y_pos, "Mean: " + str(mean_val))
plt.text(median_val, y_pos - spacing, "Median: " + str(median_val)
    )
plt.text(std_val, y_pos - 2 * spacing, "Stdev: " + str(std_val))
plt.text(quantile_01_val, y_pos - 3 * spacing, "1%ile: " + str(
    quantile_01_val))
plt.text(quantile_99_val, y_pos - 4 * spacing, "99%ile: " + str(
    quantile_99_val))

plt.gca().set(xlim = (0, 300), ylabel = 'Number of Documents',
    xlabel = 'Document Word Count')
plt.tick_params(size = 16)
plt.xticks(np.linspace(0,300,10))
plt.title('Distribution of Document Word Counts for Critical
    severity', fontdict = dict(size = 22))
plt.show()

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
n_topics = 10

plt.figure(figsize=(18, 14), dpi=160)

for i in range(n_topics):
    plt.subplot(5, 2, i + 1)
    df_dominant_topic_sub = df_topic_sents_keywords.loc[
        df_topic_sents_keywords.Dominant_Topic == i, :]
    doc_lens_2 = [len(d.split()) for d in df_dominant_topic_sub.
        Original_Text]
    doc_lens_np = np.array(doc_lens_2)

    # Calculate statistics
    mean_val = np.mean(doc_lens_np)
    median_val = np.median(doc_lens_np)
    std_val = np.std(doc_lens_np)

    plt.hist(doc_lens_np, bins=1000, color=cols[i])
    plt.gca().twinx()
```

```
        density = gaussian_kde(doc_lens_np)
        xs = np.linspace(0, 1000, 1000)
        plt.plot(xs, density(xs), color='black')

        # Plot statistics
        plt.axvline(mean_val, color='red', linestyle='-', label=f'Mean
            : {mean_val}', alpha = 0.3)
        plt.axvline(median_val, color='green', linestyle='-', label=f'
            Median: {median_val}', alpha = 0.3)
        plt.axvline(mean_val + std_val, color='blue', linestyle='-',
            label=f'Std Dev: {std_val}', alpha = 0.3)
        plt.axvline(mean_val - std_val, color='blue', linestyle='-',
            alpha = 0.3)

        plt.xlabel('Document Word Count')
        plt.ylabel('Number of Documents', color=cols[i])
        plt.title(f'Topic: {i}', fontdict=dict(size=16, color=cols[i])
            )
        plt.legend()

plt.tight_layout()
plt.subplots_adjust(top=0.90)
plt.suptitle('Distribution of Document Word Counts by Dominant
    Topic for critical seveirty', fontsize=22)
plt.show()



topics = crit_model_desc.show_topics(formatted = False)
data_flat = [w for desc in crtical["clean_desc"].dropna() for w in
     str(desc).split()]
counter = Counter(data_flat)

out = []
for i, topic in topics:
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])

df = pd.DataFrame(out, columns = ['word', 'topic_id', 'importance'
    , 'word_count'])

df.head()
```

```
fig, axes = plt.subplots(5, 2, figsize = (16,20),dpi=160)
cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]

for i, ax in enumerate(axes.flatten()):
    if i >= len(topics):  #just so it doesnt go beyond
        ax.axis('off')
        continue
    ax.bar(x = 'word', height = "word_count", data = df.loc[df.
        topic_id == i, :], color = cols[i], width = 0.5, alpha =
        0.3, label = 'Word Count')
    ax_twin = ax.twinx()
    ax_twin.bar(x= 'word', height = "importance", data = df.loc[df
        .topic_id == i, :], color = cols[i], width = 0.2, label = '
        Weights')
    ax.set_ylabel('Word Count', color = cols[i])
    ax.set_title('Topic: ' + str(i), color = cols[i], fontsize =
        16)
    ax.tick_params(axis = 'y', left = False)
    ax.set_xticklabels(df.loc[df.topic_id == i, 'word'], rotation
        = 30, horizontalalignment = 'right')
    ax.legend(loc = 'upper left'); ax_twin.legend(loc = 'upper
        right')

fig.tight_layout(w_pad = 2)
fig.suptitle('Word Count and Importance of Topic Keywords for
    Critical severity', fontsize = 22, y = 1.05)
plt.show()

dominant_topics, topic_percentages = topics_per_document(model =
    crit_model_desc, corpus = crit_corpus_dec, end = -1)

df_2 = pd.DataFrame(dominant_topics, columns = ['Document_Id', '
    Dominant_Topic'])
dominant_topic_in_each_doc = df_2.groupby('Dominant_Topic').size()
df_dominant_topic_in_each_doc = dominant_topic_in_each_doc.
    to_frame(name = 'count').reset_index()

topic_weightage_by_doc = pd.DataFrame([dict(t) for t in
    topic_percentages])
df_topic_weightage_by_doc = topic_weightage_by_doc.sum().to_frame(
    name = 'count').reset_index()

topic_top3words = [(i, topic) for i, topics in crit_model_desc.
```

```
    show_topics(formatted = False)
                                    for j, (topic, wt) in enumerate(
                                        topics) if j < 3]

df_top3words_stacked = pd.DataFrame(topic_top3words, columns = ['
    topic_id', 'words'])
df_top3words = df_top3words_stacked.groupby('topic_id').agg(',.
    join)
df_top3words.reset_index(level = 0,inplace = True)

df_top3words

topic_weights = []

for row_list in crit_model_desc[crit_corpus_dec]:
    weights = []

    for item in row_list:
        _, weight = item
        weights.append(weight)

    topic_weights.append(weights)

arr = pd.DataFrame(topic_weights).fillna(0).values

#kepping only things over 0.35
arr = arr[np.amax(arr, axis = 1) > 0.35]

topic_num = np.argmax(arr, axis = 1) #dominant topic for each
    document

# t-SNE Model
tsne_model = TSNE(n_components = 2, verbose = 1, random_state = 6,
    angle = .99, init = 'pca')
tsne_lda = tsne_model.fit_transform(arr)

n_topics = 10
mycolors = get_colormap_colors(n_topics)
mycolors_hex = [rgba_to_hex(color) for color in mycolors]
source = ColumnDataSource(data = dict(
    x = tsne_lda[:, 0],
    y = tsne_lda[:, 1],
    topic = topic_num,
```

```
    color = [mycolors_hex[i] for i in topic_num]
))

hover = HoverTool(tooltips = [("Topic", "@topic")])

mapper = linear_cmap(field_name = 'topic', palette = mycolors_hex,
    low = min(topic_num), high = max(topic_num))

p = figure(width = 900, height = 700, title = "t-SNE Clustering of
    {} LDA Topics for critical severity".format(n_topics))
p.scatter(x = 'x', y = 'y', source = source, size = 8, color = '
    color', legend_field = 'topic', fill_alpha = 0.6)
p.add_tools(hover)

output_notebook()
show(p)

### High

high = dataframe_by_severity.get_group("HIGH")



hig_model_desc, hig_model_terms, hig_corpus_dec, hig_id2word_desc
    = LDA_gen(high["clean_description"], num = 10, vectorizer_type
    = 'tfidf')

hig_model_terms

log_perplexity = hig_model_desc.log_perplexity(hig_corpus_dec)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')

gensim_dict_hig = convert_to_gensim_dictionary(hig_id2word_desc)

hig_hdp = HdpModel(hig_corpus_dec, gensim_dict_hig)

hig_all_topics = hig_hdp.get_topics()
hig_num_of_topics = hig_all_topics.shape[0]
print(f"Total number of topics: {hig_num_of_topics}")
```

```
for topic_info in hig_hdp.print_topics(num_topics = 15, num_words
    = 5):
     print(topic_info)

hig_texts = high['clean_desc'].apply(lambda x: x.split()).tolist()

hig_coherence_model_hdp = CoherenceModel(model = hig_hdp, texts =
    hig_texts, dictionary = gensim_dict_hig, coherence = 'c_v')
hig_coherence_hdp = hig_coherence_model_hdp.get_coherence()

print('Coherence Score:', hig_coherence_hdp)

hig_coherence_values = compute_coherence_for_each_topic(hig_hdp,
    hig_texts, gensim_dict_hig)

hig_top_topics = sorted(range(len(hig_coherence_values)), key =
    lambda i: hig_coherence_values[i], reverse = True)[:10]

print("Top 10 Topics based on Coherence Scores:")
for idx in hig_top_topics:
     print("Topic {}: Coherence Score: {:.4f}".format(idx,
         hig_coherence_values[idx]))

#sorted_coherence_values = sorted(coherence_values, reverse=True)

topics_range = range(1, len(hig_coherence_values) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range, hig_coherence_values, marker = 'o',
    linestyle = '--')
plt.title('Coherence Scores by Number of Topics for High severity'
    )
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

hig_sorted_coherence_values = sorted(hig_coherence_values, reverse
    =True)

topics_range = range(1, len(hig_sorted_coherence_values) + 1)
```

```
plt.figure(figsize = (10,6))
plt.plot(topics_range, hig_sorted_coherence_values, marker = 'o',
    linestyle = '--')
plt.title('Sorted Coherence Scores by Number of Topics for High
    severity')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

df_topic_sents_keywords = format_topics_sentences(ldamodel =
    hig_model_desc, corpus = hig_corpus_dec, texts = high["
    clean_desc"].tolist())

df_topic_sents_keywords

doc_lens = [len(d.split()) for d in df_topic_sents_keywords.
    Original_Text]

mean_val = round(np.mean(doc_lens))
median_val = round(np.median(doc_lens))
std_val = round(np.std(doc_lens))
quantile_01_val = round(np.quantile(doc_lens, q = 0.01))
quantile_99_val = round(np.quantile(doc_lens, q = 0.99))

plt.figure(figsize = (16,7), dpi = 160)
plt.hist(doc_lens, bins = 1000, color = 'navy')

y_pos = max(plt.gca().get_ylim()) - 0.05 * max(plt.gca().get_ylim
    ())
spacing = 0.025 * max(plt.gca().get_ylim())

plt.text(mean_val, y_pos, "Mean: " + str(mean_val))
plt.text(median_val, y_pos - spacing, "Median: " + str(median_val)
    )
plt.text(std_val, y_pos - 2 * spacing, "Stdev: " + str(std_val))
plt.text(quantile_01_val, y_pos - 3 * spacing, "1%ile: " + str(
    quantile_01_val))
plt.text(quantile_99_val, y_pos - 4 * spacing, "99%ile: " + str(
    quantile_99_val))

plt.gca().set(xlim = (0, 300), ylabel = 'Number of Documents',
```

```
    xlabel = 'Document Word Count')
plt.tick_params(size = 16)
plt.xticks(np.linspace(0,300,10))
plt.title('Distribution of Document Word Counts for High severity'
    , fontdict = dict(size = 22))
plt.show()

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
n_topics = 10

plt.figure(figsize=(18, 14), dpi=160)

for i in range(n_topics):
    plt.subplot(5, 2, i + 1)
    df_dominant_topic_sub = df_topic_sents_keywords.loc[
        df_topic_sents_keywords.Dominant_Topic == i, :]
    doc_lens_2 = [len(d.split()) for d in df_dominant_topic_sub.
        Original_Text]
    doc_lens_np = np.array(doc_lens_2)

    # Calculate statistics
    mean_val = np.mean(doc_lens_np)
    median_val = np.median(doc_lens_np)
    std_val = np.std(doc_lens_np)

    plt.hist(doc_lens_np, bins=1000, color=cols[i])
    plt.gca().twinx()
    density = gaussian_kde(doc_lens_np)
    xs = np.linspace(0, 1000, 1000)
    plt.plot(xs, density(xs), color='black')

    # Plot statistics
    plt.axvline(mean_val, color='red', linestyle='-', label=f'Mean
        : {mean_val}', alpha = 0.3)
    plt.axvline(median_val, color='green', linestyle='-', label=f'
        Median: {median_val}', alpha = 0.3)
    plt.axvline(mean_val + std_val, color='blue', linestyle='-',
        label=f'Std Dev: {std_val}', alpha = 0.3)
    plt.axvline(mean_val - std_val, color='blue', linestyle='-',
        alpha = 0.3)

    plt.xlabel('Document Word Count')
    plt.ylabel('Number of Documents', color=cols[i])
```

```
    plt.title(f'Topic: {i}', fontdict=dict(size=16, color=cols[i])
        )
    plt.legend()

plt.tight_layout()
plt.subplots_adjust(top=0.90)
plt.suptitle('Distribution of Document Word Counts by Dominant
    Topic for high severity', fontsize=22)
plt.show()



topics = hig_model_desc.show_topics(formatted = False)
data_flat = [w for desc in high["clean_desc"].dropna() for w in
    str(desc).split()]
counter = Counter(data_flat)

out = []
for i, topic in topics:
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])

df = pd.DataFrame(out, columns = ['word', 'topic_id', 'importance'
    , 'word_count'])

df.head()

fig, axes = plt.subplots(5, 2, figsize = (16,20), sharey=True, dpi
    =160)
cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]

for i, ax in enumerate(axes.flatten()):
    if i >= len(topics):  #just so it doesnt go beyond
        ax.axis('off')
        continue
    ax.bar(x = 'word', height = "word_count", data = df.loc[df.
        topic_id == i, :], color = cols[i], width = 0.5, alpha =
        0.3, label = 'Word Count')
    ax_twin = ax.twinx()
    ax_twin.bar(x= 'word', height = "importance", data = df.loc[df
        .topic_id == i, :], color = cols[i], width = 0.2, label = '
        Weights')
    ax.set_ylabel('Word Count', color = cols[i])
```

```
    ax.set_title('Topic: ' + str(i), color = cols[i], fontsize =
        16)
    ax.tick_params(axis = 'y', left = False)
    ax.set_xticklabels(df.loc[df.topic_id == i, 'word'], rotation
        = 30, horizontalalignment = 'right')
    ax.legend(loc = 'upper left'); ax_twin.legend(loc = 'upper
        right')

fig.tight_layout(w_pad = 2)
fig.suptitle('Word Count and Importance of Topic Keywords for High
    severity', fontsize = 22, y = 1.05)
plt.show()

dominant_topics, topic_percentages = topics_per_document(model =
    hig_model_desc, corpus = hig_corpus_dec, end = -1)

df_2 = pd.DataFrame(dominant_topics, columns = ['Document_Id', '
    Dominant_Topic'])
dominant_topic_in_each_doc = df_2.groupby('Dominant_Topic').size()
df_dominant_topic_in_each_doc = dominant_topic_in_each_doc.
    to_frame(name = 'count').reset_index()

topic_weightage_by_doc = pd.DataFrame([dict(t) for t in
    topic_percentages])
df_topic_weightage_by_doc = topic_weightage_by_doc.sum().to_frame(
    name = 'count').reset_index()

topic_top3words = [(i, topic) for i, topics in hig_model_desc.
    show_topics(formatted = False)
                                for j, (topic, wt) in enumerate(
                                    topics) if j < 3]

df_top3words_stacked = pd.DataFrame(topic_top3words, columns = ['
    topic_id', 'words'])
df_top3words = df_top3words_stacked.groupby('topic_id').agg(','.
    join)
df_top3words.reset_index(level = 0,inplace = True)

df_top3words

topic_weights = []

for row_list in hig_model_desc[hig_corpus_dec]:
```

```
    weights = []

    for item in row_list:
        _, weight = item
        weights.append(weight)

    topic_weights.append(weights)

arr = pd.DataFrame(topic_weights).fillna(0).values

#kepping only things over 0.35
arr = arr[np.amax(arr, axis = 1) > 0.35]

topic_num = np.argmax(arr, axis = 1) #dominant topic for each
    document

# t-SNE Model
tsne_model = TSNE(n_components = 2, verbose = 1, random_state = 6,
    angle = .99, init = 'pca')
tsne_lda = tsne_model.fit_transform(arr)

n_topics = 10
mycolors = get_colormap_colors(n_topics)
mycolors_hex = [rgba_to_hex(color) for color in mycolors]
source = ColumnDataSource(data = dict(
    x = tsne_lda[:, 0],
    y = tsne_lda[:, 1],
    topic = topic_num,
    color = [mycolors_hex[i] for i in topic_num]
))

hover = HoverTool(tooltips = [("Topic", "@topic")])

mapper = linear_cmap(field_name = 'topic', palette = mycolors_hex,
    low = min(topic_num), high = max(topic_num))

p = figure(width = 900, height = 700, title = "t-SNE Clustering of
    {} LDA Topics for high severity".format(n_topics))
p.scatter(x = 'x', y = 'y', source = source, size = 8, color = '
    color', legend_field = 'topic', fill_alpha = 0.6)
p.add_tools(hover)

output_notebook()
```

```
show(p)

### Low

low = dataframe_by_severity.get_group("LOW")

low_model_desc, low_model_terms, low_corpus_dec, low_id2word_desc
    = LDA_gen(low["clean_description"], num = 10, vectorizer_type =
     'tfidf')

low_model_terms

log_perplexity = low_model_desc.log_perplexity(low_corpus_dec)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')

gensim_dict_low = convert_to_gensim_dictionary(low_id2word_desc)

low_hdp = HdpModel(low_corpus_dec, gensim_dict_low)

low_all_topics = low_hdp.get_topics()
low_num_of_topics = low_all_topics.shape[0]
print(f"Total number of topics: {low_num_of_topics}")

for topic_info in low_hdp.print_topics(num_topics = 15, num_words
    = 5):
    print(topic_info)

low_texts = low['clean_desc'].apply(lambda x: x.split()).tolist()

low_coherence_model_hdp = CoherenceModel(model = low_hdp, texts =
    low_texts, dictionary = gensim_dict_low, coherence = 'c_v')
low_coherence_hdp = low_coherence_model_hdp.get_coherence()

print('Coherence Score:', low_coherence_hdp)

low_coherence_values = compute_coherence_for_each_topic(low_hdp,
    low_texts, gensim_dict_low)

low_top_topics = sorted(range(len(low_coherence_values)), key =
```

```
    lambda i: low_coherence_values[i], reverse = True)[:10]

print("Top 10 Topics based on Coherence Scores:")
for idx in low_top_topics:
    print("Topic {}: Coherence Score: {:.4f}".format(idx,
        low_coherence_values[idx]))

#sorted_coherence_values = sorted(coherence_values, reverse=True)

topics_range = range(1, len(low_coherence_values) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range, low_coherence_values, marker = 'o',
    linestyle = '--')
plt.title('Coherence Scores by Number of Topics for Low severity')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

low_sorted_coherence_values = sorted(low_coherence_values, reverse
    =True)

topics_range = range(1, len(low_sorted_coherence_values) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range, low_sorted_coherence_values, marker = 'o',
    linestyle = '--')
plt.title('Sorted Coherence Scores by Number of Topics for Low
    severity')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

df_topic_sents_keywords = format_topics_sentences(ldamodel =
    low_model_desc, corpus = low_corpus_dec, texts = low["
    clean_desc"].tolist())

df_topic_sents_keywords

doc_lens = [len(d.split()) for d in df_topic_sents_keywords.
    Original_Text]
```

```
mean_val = round(np.mean(doc_lens))
median_val = round(np.median(doc_lens))
std_val = round(np.std(doc_lens))
quantile_01_val = round(np.quantile(doc_lens, q = 0.01))
quantile_99_val = round(np.quantile(doc_lens, q = 0.99))

plt.figure(figsize = (16,7), dpi = 160)
plt.hist(doc_lens, bins = 1000, color = 'navy')

y_pos = max(plt.gca().get_ylim()) - 0.05 * max(plt.gca().get_ylim
    ())
spacing = 0.025 * max(plt.gca().get_ylim())

plt.text(mean_val, y_pos, "Mean: " + str(mean_val))
plt.text(median_val, y_pos - spacing, "Median: " + str(median_val)
    )
plt.text(std_val, y_pos - 2 * spacing, "Stdev: " + str(std_val))
plt.text(quantile_01_val, y_pos - 3 * spacing, "1%ile: " + str(
    quantile_01_val))
plt.text(quantile_99_val, y_pos - 4 * spacing, "99%ile: " + str(
    quantile_99_val))

plt.gca().set(xlim = (0, 300), ylabel = 'Number of Documents',
    xlabel = 'Document Word Count')
plt.tick_params(size = 16)
plt.xticks(np.linspace(0,300,10))
plt.title('Distribution of Document Word Counts for Low severity',
     fontdict = dict(size = 22))
plt.show()

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
n_topics = 10

plt.figure(figsize=(18, 14), dpi=160)

for i in range(n_topics):
    plt.subplot(5, 2, i + 1)
    df_dominant_topic_sub = df_topic_sents_keywords.loc[
        df_topic_sents_keywords.Dominant_Topic == i, :]
    doc_lens_2 = [len(d.split()) for d in df_dominant_topic_sub.
        Original_Text]
    doc_lens_np = np.array(doc_lens_2)
```

```
    # Calculate statistics
    mean_val = np.mean(doc_lens_np)
    median_val = np.median(doc_lens_np)
    std_val = np.std(doc_lens_np)

    plt.hist(doc_lens_np, bins=1000, color=cols[i])
    plt.gca().twinx()
    density = gaussian_kde(doc_lens_np)
    xs = np.linspace(0, 1000, 1000)
    plt.plot(xs, density(xs), color='black')

    # Plot statistics
    plt.axvline(mean_val, color='red', linestyle='-', label=f'Mean
        : {mean_val}', alpha = 0.3)
    plt.axvline(median_val, color='green', linestyle='-', label=f'
        Median: {median_val}', alpha = 0.3)
    plt.axvline(mean_val + std_val, color='blue', linestyle='-',
        label=f'Std Dev: {std_val}', alpha = 0.3)
    plt.axvline(mean_val - std_val, color='blue', linestyle='-',
        alpha = 0.3)

    plt.xlabel('Document Word Count')
    plt.ylabel('Number of Documents', color=cols[i])
    plt.title(f'Topic: {i}', fontdict=dict(size=16, color=cols[i])
        )
    plt.legend()

plt.tight_layout()
plt.subplots_adjust(top=0.90)
plt.suptitle('Distribution of Document Word Counts by Dominant
    Topic for Low severity', fontsize=22)
plt.show()



topics = low_model_desc.show_topics(formatted = False)
data_flat = [w for desc in low["clean_desc"].dropna() for w in str
    (desc).split()]
counter = Counter(data_flat)

out = []
for i, topic in topics:
```

```
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])

df = pd.DataFrame(out, columns = ['word', 'topic_id', 'importance'
    , 'word_count'])

df.head()

fig, axes = plt.subplots(5, 2, figsize = (16,20), sharey=True, dpi
    =160)
cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]

for i, ax in enumerate(axes.flatten()):
    if i >= len(topics):  #just so it doesnt go beyond
        ax.axis('off')
        continue
    ax.bar(x = 'word', height = "word_count", data = df.loc[df.
        topic_id == i, :], color = cols[i], width = 0.5, alpha =
        0.3, label = 'Word Count')
    ax_twin = ax.twinx()
    ax_twin.bar(x= 'word', height = "importance", data = df.loc[df
        .topic_id == i, :], color = cols[i], width = 0.2, label = '
        Weights')
    ax.set_ylabel('Word Count', color = cols[i])
    ax.set_title('Topic: ' + str(i), color = cols[i], fontsize =
        16)
    ax.tick_params(axis = 'y', left = False)
    ax.set_xticklabels(df.loc[df.topic_id == i, 'word'], rotation
        = 30, horizontalalignment = 'right')
    ax.legend(loc = 'upper left'); ax_twin.legend(loc = 'upper
        right')

fig.tight_layout(w_pad = 2)
fig.suptitle('Word Count and Importance of Topic Keywords for Low
    severity', fontsize = 22, y = 1.05)
plt.show()

dominant_topics, topic_percentages = topics_per_document(model =
    low_model_desc, corpus = low_corpus_dec, end = -1)

df_2 = pd.DataFrame(dominant_topics, columns = ['Document_Id', '
    Dominant_Topic'])
dominant_topic_in_each_doc = df_2.groupby('Dominant_Topic').size()
```

```
df_dominant_topic_in_each_doc = dominant_topic_in_each_doc.
    to_frame(name = 'count').reset_index()

topic_weightage_by_doc = pd.DataFrame([dict(t) for t in
    topic_percentages])
df_topic_weightage_by_doc = topic_weightage_by_doc.sum().to_frame(
    name = 'count').reset_index()

topic_top3words = [(i, topic) for i, topics in low_model_desc.
    show_topics(formatted = False)
                                 for j, (topic, wt) in enumerate(
                                     topics) if j < 3]

df_top3words_stacked = pd.DataFrame(topic_top3words, columns = ['
    topic_id', 'words'])
df_top3words = df_top3words_stacked.groupby('topic_id').agg(','.
    join)
df_top3words.reset_index(level = 0,inplace = True)

df_top3words

topic_weights = []

for row_list in low_model_desc[low_corpus_dec]:
    weights = []

    for item in row_list:
        _, weight = item
        weights.append(weight)

    topic_weights.append(weights)

arr = pd.DataFrame(topic_weights).fillna(0).values

#kepping only things over 0.35
arr = arr[np.amax(arr, axis = 1) > 0.35]

topic_num = np.argmax(arr, axis = 1) #dominant topic for each
    document

# t-SNE Model
tsne_model = TSNE(n_components = 2, verbose = 1, random_state = 6,
    angle = .99, init = 'pca')
```

```
tsne_lda = tsne_model.fit_transform(arr)

n_topics = 10
mycolors = get_colormap_colors(n_topics)
mycolors_hex = [rgba_to_hex(color) for color in mycolors]
source = ColumnDataSource(data = dict(
    x = tsne_lda[:, 0],
    y = tsne_lda[:, 1],
    topic = topic_num,
    color = [mycolors_hex[i] for i in topic_num]
))

hover = HoverTool(tooltips = [("Topic", "@topic")])

mapper = linear_cmap(field_name = 'topic', palette = mycolors_hex,
    low = min(topic_num), high = max(topic_num))

p = figure(width = 900, height = 700, title = "t-SNE Clustering of
    {} LDA Topics for low severity".format(n_topics))
p.scatter(x = 'x', y = 'y', source = source, size = 8, color = '
    color', legend_field = 'topic', fill_alpha = 0.6)
p.add_tools(hover)

output_notebook()
show(p)

#### Medium

med = dataframe_by_severity.get_group("MEDIUM")

med_model_desc, med_model_terms, med_corpus_dec, med_id2word_desc
    = LDA_gen(med["clean_description"], num = 10, vectorizer_type =
     'tfidf')

med_model_terms

log_perplexity = med_model_desc.log_perplexity(med_corpus_dec)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')
```

```
gensim_dict_med = convert_to_gensim_dictionary(med_id2word_desc)

med_hdp = HdpModel(med_corpus_dec, gensim_dict_med)

med_all_topics = med_hdp.get_topics()
med_num_of_topics = med_all_topics.shape[0]
print(f"Total number of topics: {med_num_of_topics}")

for topic_info in med_hdp.print_topics(num_topics = 15, num_words
    = 5):
     print(topic_info)

med_texts = med['clean_desc'].apply(lambda x: x.split()).tolist()

med_coherence_model_hdp = CoherenceModel(model = med_hdp, texts =
    med_texts, dictionary = gensim_dict_med, coherence = 'c_v')
med_coherence_hdp = med_coherence_model_hdp.get_coherence()

print('Coherence Score:', med_coherence_hdp)

med_coherence_values = compute_coherence_for_each_topic(med_hdp,
    med_texts, gensim_dict_med)

med_top_topics = sorted(range(len(med_coherence_values)), key =
    lambda i: med_coherence_values[i], reverse = True)[:10]

print("Top 10 Topics based on Coherence Scores:")
for idx in med_top_topics:
     print("Topic {}: Coherence Score: {:.4f}".format(idx,
         med_coherence_values[idx]))

#sorted_coherence_values = sorted(coherence_values, reverse=True)

topics_range = range(1, len(med_coherence_values) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range, med_coherence_values, marker = 'o',
    linestyle = '--')
plt.title('Coherence Scores by Number of Topics for Medium
    severity')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
```

```
plt.show()

med_sorted_coherence_values = sorted(med_coherence_values, reverse
    =True)

topics_range = range(1, len(med_sorted_coherence_values) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range, med_sorted_coherence_values, marker = 'o',
    linestyle = '--')
plt.title('Sorted Coherence Scores by Number of Topics for Medium
    severity')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

df_topic_sents_keywords = format_topics_sentences(ldamodel =
    med_model_desc, corpus = med_corpus_dec, texts = med["
    clean_desc"].tolist())

df_topic_sents_keywords

doc_lens = [len(d.split()) for d in df_topic_sents_keywords.
    Original_Text]

mean_val = round(np.mean(doc_lens))
median_val = round(np.median(doc_lens))
std_val = round(np.std(doc_lens))
quantile_01_val = round(np.quantile(doc_lens, q = 0.01))
quantile_99_val = round(np.quantile(doc_lens, q = 0.99))

plt.figure(figsize = (16,7), dpi = 160)
plt.hist(doc_lens, bins = 1000, color = 'navy')

y_pos = max(plt.gca().get_ylim()) - 0.05 * max(plt.gca().get_ylim
    ())
spacing = 0.025 * max(plt.gca().get_ylim())

plt.text(mean_val, y_pos, "Mean: " + str(mean_val))
plt.text(median_val, y_pos - spacing, "Median: " + str(median_val)
    )
plt.text(std_val, y_pos - 2 * spacing, "Stdev: " + str(std_val))
```

```
plt.text(quantile_01_val, y_pos - 3 * spacing, "1%ile: " + str(
    quantile_01_val))
plt.text(quantile_99_val, y_pos - 4 * spacing, "99%ile: " + str(
    quantile_99_val))

plt.gca().set(xlim = (0, 300), ylabel = 'Number of Documents',
    xlabel = 'Document Word Count')
plt.tick_params(size = 16)
plt.xticks(np.linspace(0,300,10))
plt.title('Distribution of Document Word Counts for Medium
    severity', fontdict = dict(size = 22))
plt.show()

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
n_topics = 10

plt.figure(figsize=(18, 14), dpi=160)

for i in range(n_topics):
    plt.subplot(5, 2, i + 1)
    df_dominant_topic_sub = df_topic_sents_keywords.loc[
        df_topic_sents_keywords.Dominant_Topic == i, :]
    doc_lens_2 = [len(d.split()) for d in df_dominant_topic_sub.
        Original_Text]
    doc_lens_np = np.array(doc_lens_2)

    # Calculate statistics
    mean_val = np.mean(doc_lens_np)
    median_val = np.median(doc_lens_np)
    std_val = np.std(doc_lens_np)

    plt.hist(doc_lens_np, bins=1000, color=cols[i])
    plt.gca().twinx()
    density = gaussian_kde(doc_lens_np)
    xs = np.linspace(0, 1000, 1000)
    plt.plot(xs, density(xs), color='black')

    # Plot statistics
    plt.axvline(mean_val, color='red', linestyle='-', label=f'Mean
        : {mean_val}', alpha = 0.3)
    plt.axvline(median_val, color='green', linestyle='-', label=f'
        Median: {median_val}', alpha = 0.3)
    plt.axvline(mean_val + std_val, color='blue', linestyle='-',
```

```
        label=f'Std Dev: {std_val}', alpha = 0.3)
    plt.axvline(mean_val - std_val, color='blue', linestyle='-',
        alpha = 0.3)

    plt.xlabel('Document Word Count')
    plt.ylabel('Number of Documents', color=cols[i])
    plt.title(f'Topic: {i}', fontdict=dict(size=16, color=cols[i])
        )
    plt.legend()

plt.tight_layout()
plt.subplots_adjust(top=0.90)
plt.suptitle('Distribution of Document Word Counts by Dominant
    Topic for Medium severity', fontsize=22)
plt.show()



topics = med_model_desc.show_topics(formatted = False)
data_flat = [w for desc in med["clean_desc"].dropna() for w in str
    (desc).split()]
counter = Counter(data_flat)

out = []
for i, topic in topics:
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])

df = pd.DataFrame(out, columns = ['word', 'topic_id', 'importance'
    , 'word_count'])

df.head()

fig, axes = plt.subplots(5,2, figsize = (16,20), sharey=True, dpi
    =160)
cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]

for i, ax in enumerate(axes.flatten()):
    if i >= len(topics):  #just so it doesnt go beyond
        ax.axis('off')
        continue
    ax.bar(x = 'word', height = "word_count", data = df.loc[df.
        topic_id == i, :], color = cols[i], width = 0.5, alpha =
```

```
        0.3, label = 'Word Count')
    ax_twin = ax.twinx()
    ax_twin.bar(x= 'word', height = "importance", data = df.loc[df
        .topic_id == i, :], color = cols[i], width = 0.2, label = '
        Weights')
    ax.set_ylabel('Word Count', color = cols[i])
    ax.set_title('Topic: ' + str(i), color = cols[i], fontsize =
        16)
    ax.tick_params(axis = 'y', left = False)
    ax.set_xticklabels(df.loc[df.topic_id == i, 'word'], rotation
        = 30, horizontalalignment = 'right')
    ax.legend(loc = 'upper left'); ax_twin.legend(loc = 'upper
        right')

fig.tight_layout(w_pad = 2)
fig.suptitle('Word Count and Importance of Topic Keywords for
    Medium severity', fontsize = 22, y = 1.05)
plt.show()

dominant_topics, topic_percentages = topics_per_document(model =
    med_model_desc, corpus = med_corpus_dec, end = -1)

df_2 = pd.DataFrame(dominant_topics, columns = ['Document_Id', '
    Dominant_Topic'])
dominant_topic_in_each_doc = df_2.groupby('Dominant_Topic').size()
df_dominant_topic_in_each_doc = dominant_topic_in_each_doc.
    to_frame(name = 'count').reset_index()

topic_weightage_by_doc = pd.DataFrame([dict(t) for t in
    topic_percentages])
df_topic_weightage_by_doc = topic_weightage_by_doc.sum().to_frame(
    name = 'count').reset_index()

topic_top3words = [(i, topic) for i, topics in med_model_desc.
    show_topics(formatted = False)
                                for j, (topic, wt) in enumerate(
                                    topics) if j < 3]

df_top3words_stacked = pd.DataFrame(topic_top3words, columns = ['
    topic_id', 'words'])
df_top3words = df_top3words_stacked.groupby('topic_id').agg(','.
    join)
df_top3words.reset_index(level = 0,inplace = True)
```

```
df_top3words

topic_weights = []

for row_list in med_model_desc[med_corpus_dec]:
    weights = []

    for item in row_list:
        _, weight = item
        weights.append(weight)

    topic_weights.append(weights)

arr = pd.DataFrame(topic_weights).fillna(0).values

#kepping only things over 0.35
arr = arr[np.amax(arr, axis = 1) > 0.35]

topic_num = np.argmax(arr, axis = 1) #dominant topic for each
    document

# t-SNE Model
tsne_model = TSNE(n_components = 2, verbose = 1, random_state = 6,
    angle = .99, init = 'pca')
tsne_lda = tsne_model.fit_transform(arr)

n_topics = 10
mycolors = get_colormap_colors(n_topics)
mycolors_hex = [rgba_to_hex(color) for color in mycolors]
source = ColumnDataSource(data = dict(
    x = tsne_lda[:, 0],
    y = tsne_lda[:, 1],
    topic = topic_num,
    color = [mycolors_hex[i] for i in topic_num]
))

hover = HoverTool(tooltips = [("Topic", "@topic")])

mapper = linear_cmap(field_name = 'topic', palette = mycolors_hex,
    low = min(topic_num), high = max(topic_num))

p = figure(width = 900, height = 700, title = "t-SNE Clustering of
```

```
    {} LDA Topics for medium severity".format(n_topics))
p.scatter(x = 'x', y = 'y', source = source, size = 8, color = '
    color', legend_field = 'topic', fill_alpha = 0.6)
p.add_tools(hover)

output_notebook()
show(p)


## All

model_desc, model_terms, corpus_dec, id2word_desc = LDA_gen(
    df_mega_2["clean_description"], num = 10, vectorizer_type = '
    tfidf')

model_terms

lda_model_desc, lda_model_terms, lda_corpus_dec, lda_gensim_dict =
     LDA_gen_2(df_mega_2["clean_description"], num = 10)

lda_model_terms

log_perplexity = model_desc.log_perplexity(corpus_dec)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')

log_perplexity = lda_model_desc.log_perplexity(lda_corpus_dec)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')

coherence_model_lda = CoherenceModel(model = lda_model_desc, texts
    = texts, dictionary = lda_gensim_dict, coherence = 'c_v')
coherence_lda = coherence_model_lda.get_coherence()

print('Coherence Score:', coherence_lda)
```

```
topics = lda_model_desc.show_topics(formatted=False)

coherence_values_per_topic = []

for i, topic in topics:
    cm = CoherenceModel(topics=[dict(lda_model_desc.show_topic(i,
        topn=20))], corpus=lda_corpus_dec,texts = texts, dictionary
        =lda_gensim_dict, coherence='c_v')
    coherence_values_per_topic.append((i, cm.get_coherence()))

print(f'Coherence Scores per Topic: {coherence_values_per_topic}')

hdp = HdpModel(lda_corpus_dec, lda_gensim_dict)

all_topics = hdp.get_topics()
num_of_topics = all_topics.shape[0]
print(f"Total number of topics: {num_of_topics}")

#topics = hdp.print_topics(num_topics = num_of_topics)



for topic_info in hdp.print_topics(num_topics = 15, num_words = 5)
    :
    print(topic_info)

texts = df_mega_2['clean_desc'].apply(lambda x: x.split()).tolist
    ()

coherence_model_hdp = CoherenceModel(model = hdp, texts = texts,
    dictionary = lda_gensim_dict, coherence = 'c_v')
coherence_hdp = coherence_model_hdp.get_coherence()

# coherences = hdp.get_coherence_per_topic()



coherence_values = compute_coherence_for_each_topic(hdp, texts,
    lda_gensim_dict)

top_topics = sorted(range(len(coherence_values)), key = lambda i:
    coherence_values[i], reverse = True)[:10]
```

```
print("Top 10 Topics based on Coherence Scores:")
for idx in top_topics:
    print("Topic {}: Coherence Score: {:.4f}".format(idx,
        coherence_values[idx]))

print('Coherence Score:', coherence_hdp)

#sorted_coherence_values = sorted(coherence_values, reverse=True)

topics_range = range(1, len(coherence_values) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range, coherence_values, marker = 'o', linestyle =
    '--')
plt.title('Coherence Scores by Number of Topics')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

sorted_coherence_values = sorted(coherence_values, reverse=True)

topics_range = range(1, len(sorted_coherence_values) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range, sorted_coherence_values, marker = 'o',
   linestyle = '--')
plt.title('Sorted Coherence Scores by Number of Topics')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

### so from the coherence graph we can assume that 10 number of
   topics is good, basically the



df_topic_sents_keywords = format_topics_sentences(ldamodel =
   lda_model_desc, corpus = lda_corpus_dec, texts = df_mega_2["
   clean_desc"].tolist())
```

```
df_topic_sents_keywords.head()

# np.max(doc_lens)

doc_lens = [len(d.split()) for d in df_topic_sents_keywords.
    Original_Text]

mean_val = round(np.mean(doc_lens))
median_val = round(np.median(doc_lens))
std_val = round(np.std(doc_lens))
quantile_01_val = round(np.quantile(doc_lens, q = 0.01))
quantile_99_val = round(np.quantile(doc_lens, q = 0.99))

plt.figure(figsize = (16,7), dpi = 160)
plt.hist(doc_lens, bins = 1000, color = 'navy')


plt.text(quantile_99_val, 600, "Mean: " + str(mean_val))
plt.text(quantile_99_val, 500, "Median: " + str(median_val))
plt.text(quantile_99_val, 400, "Stdev: " + str(std_val))
plt.text(quantile_99_val, 300, "1%ile: " + str(quantile_01_val))
plt.text(quantile_99_val, 200, "99%ile: " + str(quantile_99_val))

plt.gca().set(xlim = (0, 300), ylabel = 'Number of Documents',
    xlabel = 'Document Word Count')
plt.tick_params(size = 16)
plt.xticks(np.linspace(0,300,10))
plt.title('Distribution of Document Word Counts', fontdict = dict(
    size = 22))
plt.show()

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
n_topics = 10

plt.figure(figsize=(18, 14), dpi=160)

for i in range(n_topics):
    plt.subplot(5, 2, i + 1)
    df_dominant_topic_sub = df_topic_sents_keywords.loc[
        df_topic_sents_keywords.Dominant_Topic == i, :]
    doc_lens_2 = [len(d.split()) for d in df_dominant_topic_sub.
        Original_Text]
    doc_lens_np = np.array(doc_lens_2)
```

```
    # Calculate statistics
    mean_val = np.mean(doc_lens_np)
    median_val = np.median(doc_lens_np)
    std_val = np.std(doc_lens_np)

    plt.hist(doc_lens_np, bins=1000, color=cols[i])
    plt.gca().twinx()
    density = gaussian_kde(doc_lens_np)
    xs = np.linspace(0, 1000, 1000)
    plt.plot(xs, density(xs), color='black')

    # Plot statistics
    plt.axvline(mean_val, color='red', linestyle='-', label=f'Mean
        : {mean_val}', alpha = 0.3)
    plt.axvline(median_val, color='green', linestyle='-', label=f'
        Median: {median_val}', alpha = 0.3)
    plt.axvline(mean_val + std_val, color='blue', linestyle='-',
        label=f'Std Dev: {std_val}', alpha = 0.3)
    plt.axvline(mean_val - std_val, color='blue', linestyle='-',
        alpha = 0.3)

    plt.xlabel('Document Word Count')
    plt.ylabel('Number of Documents', color=cols[i])
    plt.title(f'Topic: {i}', fontdict=dict(size=16, color=cols[i])
        )
    plt.legend()

plt.tight_layout()
plt.subplots_adjust(top=0.90)
plt.suptitle('Distribution of Document Word Counts by Dominant
    Topic', fontsize=22)
plt.show()



topics = lda_model_desc.show_topics(formatted = False)
data_flat = [w for desc in df_mega_2["clean_desc"].dropna() for w
    in str(desc).split()]
counter = Counter(data_flat)

out = []
for i, topic in topics:
```

```
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])

df = pd.DataFrame(out, columns = ['word', 'topic_id', 'importance'
    , 'word_count'])

df.head()

fig, axes = plt.subplots(5, 2, figsize = (20,20), sharey=True, dpi
    =160)
cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]

for i, ax in enumerate(axes.flatten()):
    if i >= len(topics):  #just so it doesnt go beyond
        ax.axis('off')
        continue
    ax.bar(x = 'word', height = "word_count", data = df.loc[df.
        topic_id == i, :], color = cols[i], width = 0.5, alpha =
        0.3, label = 'Word Count')
    ax_twin = ax.twinx()
    ax_twin.bar(x= 'word', height = "importance", data = df.loc[df
        .topic_id == i, :], color = cols[i], width = 0.2, label = '
        Weights')
    ax.set_ylabel('Word Count', color = cols[i])
    ax.set_title('Topic: ' + str(i), color = cols[i], fontsize =
        16)
    ax.tick_params(axis = 'y', left = False)
    ax.set_xticklabels(df.loc[df.topic_id == i, 'word'], rotation
        = 30, horizontalalignment = 'right')
    ax.legend(loc = 'upper left'); ax_twin.legend(loc = 'upper
        right')

fig.tight_layout(w_pad = 2)
fig.suptitle('Word Count and Importance of Topic Keywords',
    fontsize = 22, y = 1.05)
plt.show()




dominant_topics, topic_percentages = topics_per_document(model =
    lda_model_desc, corpus = lda_corpus_dec, end = -1)
```

```
df_2 = pd.DataFrame(dominant_topics, columns = ['Document_Id', '
    Dominant_Topic'])
dominant_topic_in_each_doc = df_2.groupby('Dominant_Topic').size()
df_dominant_topic_in_each_doc = dominant_topic_in_each_doc.
    to_frame(name = 'count').reset_index()

topic_weightage_by_doc = pd.DataFrame([dict(t) for t in
    topic_percentages])
df_topic_weightage_by_doc = topic_weightage_by_doc.sum().to_frame(
    name = 'count').reset_index()

topic_top3words = [(i, topic) for i, topics in lda_model_desc.
    show_topics(formatted = False)
                                 for j, (topic, wt) in enumerate(
                                     topics) if j < 3]

df_top3words_stacked = pd.DataFrame(topic_top3words, columns = ['
    topic_id', 'words'])
df_top3words = df_top3words_stacked.groupby('topic_id').agg(','.
    join)
df_top3words.reset_index(level = 0,inplace = True)

df_top3words

# labels

labels = []

for x in range(len(df_dominant_topic_in_each_doc.Dominant_Topic.
    unique())):

    topic_label = "Topic " + str(x)

    words = df_top3words.loc[df_top3words.topic_id == x, 'words'].
        values[0]
    #words_on_newlines = words.replace(' ', "\n")

    label = topic_label + " \n" + words + " \n" #\n not working do
        something

    labels.append(label)
```

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize = (20, 20), dpi =
    120, sharey = True) #remove sharey


ax1.bar(x = 'Dominant_Topic', height = 'count', data =
    df_dominant_topic_in_each_doc, width = .5, color = 'firebrick')
ax1.set_xticks(range(df_dominant_topic_in_each_doc.Dominant_Topic.
    unique().__len__()))

#labels = ['Topic ' + str(x) + '\n' + df_top3words.loc[
    df_top3words.topic_id==x, 'words'].values[0].replace(' ', '\n')
     for x in range(len(df_dominant_topic_in_each_doc.
    Dominant_Topic.unique()))]

ax1.set_xticklabels(labels, rotation = 0)
ax1.set_title('Number of Documents by Dominant Topic', fontdict =
    dict(size = 10))
ax1.set_ylabel('Number of Documents')

ax2.bar(x = 'index', height = 'count', data =
    df_topic_weightage_by_doc, width = .5, color = 'steelblue')
ax2.set_xticks(range(df_topic_weightage_by_doc.index.unique().
    __len__()))
ax2.set_xticklabels(labels, rotation = 0)
ax2.set_title('Number of Documents by Topic Weightage', fontdict=
    dict(size=10))

plt.show()

topic_weights = []

for row_list in lda_model_desc[lda_corpus_dec]:
    weights = []

    for item in row_list:
        _, weight = item
        weights.append(weight)

    topic_weights.append(weights)

arr = pd.DataFrame(topic_weights).fillna(0).values

#kepping only things over 0.35
```

```
arr = arr[np.amax(arr, axis = 1) > 0.35]

topic_num = np.argmax(arr, axis = 1) #dominant topic for each
    document

topic_num

# t-SNE Model
tsne_model = TSNE(n_components = 2, verbose = 1, random_state = 6,
     angle = .99, init = 'pca')
tsne_lda = tsne_model.fit_transform(arr)

def rgba_to_hex(rgba):
    return "#{:02X}{:02X}{:02X}".format(int(rgba[0]*255), int(rgba
        [1]*255), int(rgba[2]*255))

# mycolors_hex = [rgba_to_hex(color) for color in mycolors]

def get_colormap_colors(n, cmap_name = 'nipy_spectral'):
    colormap = plt.cm.get_cmap(cmap_name, n)
    return [colormap(i) for i in range(colormap.N)]

n_topics = 10
mycolors = get_colormap_colors(n_topics)
mycolors_hex = [rgba_to_hex(color) for color in mycolors]
source = ColumnDataSource(data = dict(
    x = tsne_lda[:, 0],
    y = tsne_lda[:, 1],
    topic = topic_num,
    color = [mycolors_hex[i] for i in topic_num]
))

hover = HoverTool(tooltips = [("Topic", "@topic")])

mapper = linear_cmap(field_name = 'topic', palette = mycolors_hex,
     low = min(topic_num), high = max(topic_num))

p = figure(width = 900, height = 700, title = "t-SNE Clustering of
    {} LDA Topics".format(n_topics))
p.scatter(x = 'x', y = 'y', source = source, size = 8, color = '
    color', legend_field = 'topic', fill_alpha = 0.6)
p.add_tools(hover)
```

```
output_notebook()
show(p)



## 2019

desc_2019 = dataframes_by_year[0] #gen 1 cause smaller

model_desc_2019, model_terms_2019, corpus_dec_2019,
    id2word_desc_2019 = LDA_gen(desc_2019["clean_description"], num
    = 15, vectorizer_type = 'tfidf')

model_terms_2019

log_perplexity = model_desc_2019.log_perplexity(corpus_dec_2019)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')

def convert_to_gensim_dictionary(id2word_desc):

    word2id_desc = {v: k for k, v in id2word_desc.items()}
    gensim_dict = Dictionary()
    gensim_dict.token2id = word2id_desc

    return gensim_dict

gensim_dict_2019 = convert_to_gensim_dictionary(id2word_desc_2019)

hdp_2019 = HdpModel(corpus_dec_2019, gensim_dict_2019)

all_topics_2019 = hdp_2019.get_topics()
num_of_topics_2019 = all_topics_2019.shape[0]
print(f"Total number of topics 2019: {num_of_topics_2019}")

# topics_2019 = hdp_2019.print_topics(num_of_topics_2019 =
    num_of_topics_2019)

texts_2019 = desc_2019['clean_desc'].apply(lambda x: x.split()).
    tolist()
```

```
coherence_model_hdp_2019 = CoherenceModel(model = hdp_2019, texts
    = texts_2019, dictionary = gensim_dict_2019, coherence = 'c_v')
coherence_hdp_2019 = coherence_model_hdp_2019.get_coherence()

coherence_hdp_2019

coherence_values_2019 = compute_coherence_for_each_topic(hdp_2019,
     texts_2019, gensim_dict_2019)

top_topics_2019 = sorted(range(len(coherence_values_2019)), key =
    lambda i: coherence_values_2019[i], reverse = True)[:10]

print("Top 10 Topics based on Coherence Scores 2019:")
for idx in top_topics_2019:
    print("Topic {}: Coherence Score: {:.4f}".format(idx,
        coherence_values_2019[idx]))

topics_range_2019 = range(1, len(coherence_values_2019) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range_2019, coherence_values_2019, marker = 'o',
    linestyle = '--')
plt.title('Coherence Scores by Number of Topics 2019')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

sorted_coherence_values_2019 = sorted(coherence_values_2019,
    reverse = True)

topics_range = range(1, len(sorted_coherence_values_2019) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range_2019, sorted_coherence_values_2019, marker =
     'o', linestyle = '--')
plt.title('Sorted Coherence Scores by Number of Topics 2019')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()
```

```
df_topic_sents_keywords_2019 = format_topics_sentences(ldamodel =
    model_desc_2019, corpus = corpus_dec_2019, texts = desc_2019["
    clean_desc"].tolist())

df_topic_sents_keywords_2019

doc_lens_2019 = [len(d.split()) for d in
    df_topic_sents_keywords_2019.Original_Text]

mean_val_2019 = round(np.mean(doc_lens_2019))
median_val_2019 = round(np.median(doc_lens_2019))
std_val_2019 = round(np.std(doc_lens_2019))
quantile_01_val_2019 = round(np.quantile(doc_lens_2019, q = 0.01))
quantile_99_val_2019 = round(np.quantile(doc_lens_2019, q = 0.99))

plt.figure(figsize = (16,7), dpi = 160)
plt.hist(doc_lens_2019, bins = 1000, color = 'navy')

y_pos = max(plt.gca().get_ylim()) - 0.05 * max(plt.gca().get_ylim
    ())
spacing = 0.025 * max(plt.gca().get_ylim())

plt.text(mean_val_2019, y_pos, "Mean: " + str(mean_val_2019))
plt.text(median_val_2019, y_pos - spacing, "Median: " + str(
    median_val_2019))
plt.text(std_val_2019, y_pos - 2 * spacing, "Stdev: " + str(
    std_val_2019))
plt.text(quantile_01_val_2019, y_pos - 3 * spacing, "1%ile: " +
    str(quantile_01_val_2019))
plt.text(quantile_99_val_2019, y_pos - 4 * spacing, "99%ile: " +
    str(quantile_99_val_2019))

plt.gca().set(xlim = (0, 300), ylabel = 'Number of Documents',
    xlabel = 'Document Word Count')
plt.tick_params(size = 16)
plt.xticks(np.linspace(0,300,10))
plt.title('Distribution of Document Word Counts 2019', fontdict =
    dict(size = 22))
plt.show()

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
n_topics = 10
```

```
plt.figure(figsize=(18, 14), dpi=160)

for i in range(n_topics):
    plt.subplot(5, 2, i + 1)
    df_dominant_topic_sub = df_topic_sents_keywords.loc[
        df_topic_sents_keywords.Dominant_Topic == i, :]
    doc_lens_2 = [len(d.split()) for d in df_dominant_topic_sub.
        Original_Text]
    doc_lens_np = np.array(doc_lens_2)

    # Calculate statistics
    mean_val = np.mean(doc_lens_np)
    median_val = np.median(doc_lens_np)
    std_val = np.std(doc_lens_np)

    plt.hist(doc_lens_np, bins=1000, color=cols[i])
    plt.gca().twinx()
    density = gaussian_kde(doc_lens_np)
    xs = np.linspace(0, 1000, 1000)
    plt.plot(xs, density(xs), color='black')

    # Plot statistics
    plt.axvline(mean_val, color='red', linestyle='-', label=f'Mean
        : {mean_val}', alpha = 0.3)
    plt.axvline(median_val, color='green', linestyle='-', label=f'
        Median: {median_val}', alpha = 0.3)
    plt.axvline(mean_val + std_val, color='blue', linestyle='-',
        label=f'Std Dev: {std_val}', alpha = 0.3)
    plt.axvline(mean_val - std_val, color='blue', linestyle='-',
        alpha = 0.3)

    plt.xlabel('Document Word Count')
    plt.ylabel('Number of Documents', color=cols[i])
    plt.title(f'Topic: {i}', fontdict=dict(size=16, color=cols[i])
        )
    plt.legend()

plt.tight_layout()
plt.subplots_adjust(top=0.90)
plt.suptitle('Distribution of Document Word Counts by Dominant
    Topic for 2019', fontsize=22)
plt.show()
```

```
topics_2019 = model_desc_2019.show_topics(num_topics=15,formatted
    = False)
data_flat_2019 = [w for desc in desc_2019["clean_desc"].dropna()
    for w in str(desc).split()]
counter = Counter(data_flat_2019)

out = []
for i, topic in topics_2019:
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])

df_2019 = pd.DataFrame(out, columns = ['word', 'topic_id', '
    importance', 'word_count'])



fig, axes = plt.subplots(5, 3, figsize=(16,20), sharey = True, dpi
    = 160)
cols = get_colormap_colors(n_topics)
for i, ax in enumerate(axes.flatten()):
    if i >= len(topics_2019):
        ax.axis('off')
        continue
    ax.bar(x = 'word', height = "word_count", data = df_2019.loc[
        df_2019.topic_id == i, :], color = cols[i % len(cols)],
        width = 0.5, alpha = 0.3, label = 'Word Count')
    ax_twin = ax.twinx()
    ax_twin.bar(x = 'word', height = "importance", data = df_2019.
        loc[df_2019.topic_id == i, :], color = cols[i % len(cols)],
         width = 0.2, label = 'Weights')
    ax.set_ylabel('Word Count', color = cols[i % len(cols)])
    ax.set_title('Topic: ' + str(i), color = cols[i % len(cols)],
        fontsize = 16)
    ax.tick_params(axis = 'y', left = False)
    ax.set_xticklabels(df_2019.loc[df_2019.topic_id == i, 'word'],
         rotation = 30, horizontalalignment = 'right')
    ax.legend(loc = 'upper left'); ax_twin.legend(loc = 'upper
        right')

fig.tight_layout(w_pad = 2)
fig.suptitle('Word Count and Importance of Topic Keywords 2019',
```

```
    fontsize = 22, y = 1.05)
plt.show()

dominant_topics_2019, topic_percentages_2019 = topics_per_document
    (model = model_desc_2019, corpus = corpus_dec_2019, end = -1)

df_2_2019 = pd.DataFrame(dominant_topics_2019, columns = ['
    Document_Id', 'Dominant_Topic'])
dominant_topic_in_each_doc_2019 = df_2_2019.groupby('
    Dominant_Topic').size()
df_dominant_topic_in_each_doc_2019 =
    dominant_topic_in_each_doc_2019.to_frame(name = 'count').
    reset_index()

topic_weightage_by_doc_2019 = pd.DataFrame([dict(t) for t in
    topic_percentages_2019])
df_topic_weightage_by_doc_2019 = topic_weightage_by_doc_2019.sum()
    .to_frame(name = 'count').reset_index()

topic_top3words_2019 = [(i, topic) for i, topics in
    model_desc_2019.show_topics(formatted = False)
                                for j, (topic, wt) in enumerate(
                                    topics) if j < 3]

df_top3words_stacked_2019 = pd.DataFrame(topic_top3words_2019,
    columns = ['topic_id', 'words'])
df_top3words_2019 = df_top3words_stacked_2019.groupby('topic_id').
    agg(','.join)
df_top3words_2019.reset_index(level = 0,inplace = True)

df_top3words_2019

topic_weights_2019 = []

for row_list in model_desc_2019[corpus_dec_2019]:
    weights = []

    for item in row_list:
        _, weight = item
        weights.append(weight)

    topic_weights_2019.append(weights)
```

```
arr_2019 = pd.DataFrame(topic_weights_2019).fillna(0).values

arr_2019 = arr_2019[np.amax(arr_2019, axis = 1) > 0.35]

print(arr_2019.dtype)

topic_num_2019 = np.argmax(arr_2019, axis = 1)

# t-SNE Model
tsne_model_2019 = TSNE(n_components = 2, verbose = 1, random_state
    = 6, angle = .99, init = 'pca')
tsne_lda_2019 = tsne_model_2019.fit_transform(arr_2019) #so tsne
    does not work when i go over 15 topics, doesnt matter if i
    change the parameters

n_topics = 15
mycolors = get_colormap_colors(n_topics)
mycolors_hex = [rgba_to_hex(color) for color in mycolors]
source = ColumnDataSource(data = dict(
    x = tsne_lda_2019[:, 0],
    y = tsne_lda_2019[:, 1],
    topic = topic_num_2019,
    color = [mycolors_hex[i] for i in topic_num_2019]
))

hover = HoverTool(tooltips = [("Topic", "@topic")])

mapper = linear_cmap(field_name = 'topic', palette = mycolors_hex,
    low = min(topic_num_2019), high = max(topic_num_2019))

p = figure(width = 900, height = 700, title = "t-SNE Clustering of
    {} LDA Topics 2019".format(n_topics))
p.scatter(x = 'x', y = 'y', source = source, size = 8, color = '
    color', legend_field = 'topic', fill_alpha = 0.6)
p.add_tools(hover)

output_notebook()
show(p)

## 2020
```

```
desc_2020 = dataframes_by_year[1]

model_desc_2020, model_terms_2020, corpus_dec_2020,
    id2word_desc_2020 = LDA_gen(desc_2020["clean_description"], num
    = 10, vectorizer_type = 'tfidf')

model_terms_2020

log_perplexity = model_desc_2020.log_perplexity(corpus_dec_2020)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')

gensim_dict_2020 = convert_to_gensim_dictionary(id2word_desc_2020)

hdp_2020 = HdpModel(corpus_dec_2020, gensim_dict_2020)

all_topics_2020 = hdp_2020.get_topics()
num_of_topics_2020 = all_topics_2020.shape[0]
print(f"Total number of topics 2020: {num_of_topics_2020}")

# topics_2019 = hdp_2019.print_topics(num_of_topics_2019 =
    num_of_topics_2019)

texts_2020 = desc_2020['clean_desc'].apply(lambda x: x.split()).
    tolist()

coherence_model_hdp_2020 = CoherenceModel(model = hdp_2020, texts
    = texts_2020, dictionary = gensim_dict_2020, coherence = 'c_v')
coherence_hdp_2020 = coherence_model_hdp_2020.get_coherence()

coherence_hdp_2020

coherence_values_2020 = compute_coherence_for_each_topic(hdp_2020,
    texts_2020, gensim_dict_2020)

top_topics_2020 = sorted(range(len(coherence_values_2020)), key =
    lambda i: coherence_values_2020[i], reverse = True)[:10]

print("Top 10 Topics based on Coherence Scores 2020:")
for idx in top_topics_2020:
```

```
    print("Topic {}: Coherence Score: {:.4f}".format(idx,
        coherence_values_2020[idx]))

topics_range_2020 = range(1, len(coherence_values_2020) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range_2020, coherence_values_2020, marker = 'o',
    linestyle = '--')
plt.title('Coherence Scores by Number of Topics 2020')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

sorted_coherence_values_2020 = sorted(coherence_values_2020,
    reverse=True)

topics_range_2020 = range(1, len(sorted_coherence_values_2020) +
    1)

plt.figure(figsize = (10,6))
plt.plot(topics_range_2020, sorted_coherence_values_2020, marker =
     'o', linestyle = '--')
plt.title('Sorted Coherence Scores by Number of Topics 2020')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

df_topic_sents_keywords_2020 = format_topics_sentences(ldamodel =
    model_desc_2020, corpus = corpus_dec_2020, texts = desc_2020["
    clean_desc"].tolist())

df_topic_sents_keywords_2020

doc_lens_2020 = [len(d.split()) for d in
    df_topic_sents_keywords_2020.Original_Text]

mean_val_2020 = round(np.mean(doc_lens_2020))
median_val_2020 = round(np.median(doc_lens_2020))
std_val_2020 = round(np.std(doc_lens_2020))
quantile_01_val_2020 = round(np.quantile(doc_lens_2020, q = 0.01))
quantile_99_val_2020 = round(np.quantile(doc_lens_2020, q = 0.99))
```

```
plt.figure(figsize = (16,7), dpi = 160)
plt.hist(doc_lens_2020, bins = 1000, color = 'navy')

y_pos = max(plt.gca().get_ylim()) - 0.05 * max(plt.gca().get_ylim
    ())
spacing = 0.025 * max(plt.gca().get_ylim())

plt.text(mean_val_2020, y_pos, "Mean: " + str(mean_val_2020))
plt.text(median_val_2020, y_pos - spacing, "Median: " + str(
    median_val_2020))
plt.text(std_val_2020, y_pos - 2 * spacing, "Stdev: " + str(
    std_val_2020))
plt.text(quantile_01_val_2020, y_pos - 3 * spacing, "1%ile: " +
    str(quantile_01_val_2020))
plt.text(quantile_99_val_2020, y_pos - 4 * spacing, "99%ile: " +
    str(quantile_99_val_2020))

plt.gca().set(xlim = (0, 300), ylabel = 'Number of Documents',
    xlabel = 'Document Word Count')
plt.tick_params(size = 16)
plt.xticks(np.linspace(0,300,10))
plt.title('Distribution of Document Word Counts 2020', fontdict =
    dict(size = 22))
plt.show()


cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
n_topics = 10

plt.figure(figsize=(18, 14), dpi=160)

for i in range(n_topics):
    plt.subplot(5, 2, i + 1)
    df_dominant_topic_sub = df_topic_sents_keywords.loc[
        df_topic_sents_keywords.Dominant_Topic == i, :]
    doc_lens_2 = [len(d.split()) for d in df_dominant_topic_sub.
        Original_Text]
    doc_lens_np = np.array(doc_lens_2)

    # Calculate statistics
    mean_val = np.mean(doc_lens_np)
    median_val = np.median(doc_lens_np)
```

```
    std_val = np.std(doc_lens_np)

    plt.hist(doc_lens_np, bins=1000, color=cols[i])
    plt.gca().twinx()
    density = gaussian_kde(doc_lens_np)
    xs = np.linspace(0, 1000, 1000)
    plt.plot(xs, density(xs), color='black')

    # Plot statistics
    plt.axvline(mean_val, color='red', linestyle='-', label=f'Mean
        : {mean_val}', alpha = 0.3)
    plt.axvline(median_val, color='green', linestyle='-', label=f'
        Median: {median_val}', alpha = 0.3)
    plt.axvline(mean_val + std_val, color='blue', linestyle='-',
        label=f'Std Dev: {std_val}', alpha = 0.3)
    plt.axvline(mean_val - std_val, color='blue', linestyle='-',
        alpha = 0.3)

    plt.xlabel('Document Word Count')
    plt.ylabel('Number of Documents', color=cols[i])
    plt.title(f'Topic: {i}', fontdict=dict(size=16, color=cols[i])
        )
    plt.legend()

plt.tight_layout()
plt.subplots_adjust(top=0.90)
plt.suptitle('Distribution of Document Word Counts by Dominant
    Topic for 2020', fontsize=22)
plt.show()



topics_2020 = model_desc_2020.show_topics(num_topics=10,formatted
    = False)
data_flat_2020 = [w for desc in desc_2020["clean_desc"].dropna()
    for w in str(desc).split()]
counter = Counter(data_flat_2020)

out = []
for i, topic in topics_2020:
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])
```

```
df_2020 = pd.DataFrame(out, columns = ['word', 'topic_id', '
    importance', 'word_count'])

fig, axes = plt.subplots(5, 2, figsize = (16,20), sharey = True,
    dpi = 160)
cols = get_colormap_colors(n_topics)
for i, ax in enumerate(axes.flatten()):
    if i >= len(topics_2020):
        ax.axis('off')
        continue
    ax.bar(x = 'word', height = "word_count", data = df_2020.loc[
        df_2020.topic_id == i, :], color = cols[i % len(cols)],
        width = 0.5, alpha = 0.3, label = 'Word Count')
    ax_twin = ax.twinx()
    ax_twin.bar(x = 'word', height = "importance", data = df_2020.
        loc[df_2020.topic_id == i, :], color = cols[i % len(cols)],
         width = 0.2, label = 'Weights')
    ax.set_ylabel('Word Count', color = cols[i % len(cols)])
    ax.set_title('Topic: ' + str(i), color = cols[i % len(cols)],
        fontsize = 16)
    ax.tick_params(axis = 'y', left = False)
    ax.set_xticklabels(df_2020.loc[df_2020.topic_id == i, 'word'],
         rotation = 30, horizontalalignment = 'right')
    ax.legend(loc = 'upper left'); ax_twin.legend(loc = 'upper
        right')

fig.tight_layout(w_pad = 2)
fig.suptitle('Word Count and Importance of Topic Keywords 2020',
    fontsize = 22, y = 1.05)
plt.show()

dominant_topics_2020, topic_percentages_2020 = topics_per_document
    (model = model_desc_2020, corpus = corpus_dec_2020, end = -1)

df_2_2020 = pd.DataFrame(dominant_topics_2020, columns = ['
    Document_Id', 'Dominant_Topic'])
dominant_topic_in_each_doc_2020 = df_2_2020.groupby('
    Dominant_Topic').size()
df_dominant_topic_in_each_doc_2020 =
    dominant_topic_in_each_doc_2020.to_frame(name = 'count').
    reset_index()

topic_weightage_by_doc_2020 = pd.DataFrame([dict(t) for t in
```

```
    topic_percentages_2020])
df_topic_weightage_by_doc_2020 = topic_weightage_by_doc_2020.sum()
    .to_frame(name = 'count').reset_index()

topic_top3words_2020 = [(i, topic) for i, topics in
    model_desc_2020.show_topics(formatted = False)
                                    for j, (topic, wt) in enumerate(
                                        topics) if j < 3]

df_top3words_stacked_2020 = pd.DataFrame(topic_top3words_2020,
    columns = ['topic_id', 'words'])
df_top3words_2020 = df_top3words_stacked_2020.groupby('topic_id').
    agg(','.join)
df_top3words_2020.reset_index(level = 0,inplace = True)

df_top3words_2020

topic_weights_2020 = []

for row_list in model_desc_2020[corpus_dec_2020]:
    weights = []

    for item in row_list:
        _, weight = item
        weights.append(weight)

    topic_weights_2020.append(weights)

arr_2020 = pd.DataFrame(topic_weights_2020).fillna(0).values

arr_2020 = arr_2020[np.amax(arr_2020, axis = 1) > 0.35]

topic_num_2020 = np.argmax(arr_2020, axis = 1)

topic_num_2020

# t-SNE Model
tsne_model_2020 = TSNE(n_components = 2, verbose = 1, random_state
    = 6, angle = .99, init = 'pca')
tsne_lda_2020 = tsne_model_2020.fit_transform(arr_2020)
```

```
n_topics = 10
mycolors = get_colormap_colors(n_topics)
mycolors_hex = [rgba_to_hex(color) for color in mycolors]

source = ColumnDataSource(data = dict(
    x = tsne_lda_2020[:, 0],
    y = tsne_lda_2020[:, 1],
    topic = topic_num_2020,
    color = [mycolors_hex[i] for i in topic_num_2020]
))

hover = HoverTool(tooltips = [("Topic", "@topic")])

mapper = linear_cmap(field_name = 'topic', palette = mycolors_hex,
    low = min(topic_num_2020), high = max(topic_num_2020))

p = figure(width = 900, height = 700, title = "t-SNE Clustering of
    {} LDA Topics 2020".format(n_topics))
p.scatter(x = 'x', y = 'y', source = source, size = 8, color = '
    color', legend_field = 'topic', fill_alpha = 0.6)
p.add_tools(hover)

output_notebook()
show(p)

## 2021

desc_2021 = dataframes_by_year[2]

model_desc_2021, model_terms_2021, corpus_dec_2021,
    id2word_desc_2021 = LDA_gen(desc_2021["clean_description"], num
    = 10, vectorizer_type = 'tfidf')

model_terms_2021

log_perplexity = model_desc_2021.log_perplexity(corpus_dec_2021)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')

gensim_dict_2021 = convert_to_gensim_dictionary(id2word_desc_2021)
```

```
hdp_2021 = HdpModel(corpus_dec_2021, gensim_dict_2021)

all_topics_2021 = hdp_2021.get_topics()
num_of_topics_2021 = all_topics_2021.shape[0]
print(f"Total number of topics 2021: {num_of_topics_2021}")

# topics_2019 = hdp_2019.print_topics(num_of_topics_2019 =
    num_of_topics_2019)

texts_2021 = desc_2021['clean_desc'].apply(lambda x: x.split()).
    tolist()

coherence_model_hdp_2021 = CoherenceModel(model = hdp_2021, texts
   = texts_2021, dictionary = gensim_dict_2021, coherence = 'c_v')
coherence_hdp_2021 = coherence_model_hdp_2021.get_coherence()

coherence_hdp_2021

coherence_values_2021 = compute_coherence_for_each_topic(hdp_2021,
     texts_2021, gensim_dict_2021)

top_topics_2021 = sorted(range(len(coherence_values_2021)), key =
    lambda i: coherence_values_2021[i], reverse = True)[:10]

print("Top 10 Topics based on Coherence Scores 2021:")
for idx in top_topics_2021:
    print("Topic {}: Coherence Score: {:.4f}".format(idx,
        coherence_values_2021[idx]))

topics_range_2021 = range(1, len(coherence_values_2021) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range_2021, coherence_values_2021, marker = 'o',
    linestyle = '--')
plt.title('Coherence Scores by Number of Topics 2021')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

sorted_coherence_values_2021 = sorted(coherence_values_2021,
    reverse=True)
```

```
topics_range_2021 = range(1, len(sorted_coherence_values_2021) +
    1)

plt.figure(figsize = (10,6))
plt.plot(topics_range_2021, sorted_coherence_values_2021, marker =
    'o', linestyle = '--')
plt.title('Sorted Coherence Scores by Number of Topics 2021')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

df_topic_sents_keywords_2021 = format_topics_sentences(ldamodel =
    model_desc_2021, corpus = corpus_dec_2021, texts = desc_2021["
    clean_desc"].tolist())

df_topic_sents_keywords_2021

doc_lens_2021 = [len(d.split()) for d in
    df_topic_sents_keywords_2021.Original_Text]

mean_val_2021 = round(np.mean(doc_lens_2021))
median_val_2021 = round(np.median(doc_lens_2021))
std_val_2021 = round(np.std(doc_lens_2021))
quantile_01_val_2021 = round(np.quantile(doc_lens_2021, q = 0.01))
quantile_99_val_2021 = round(np.quantile(doc_lens_2021, q = 0.99))

plt.figure(figsize = (16,7), dpi = 160)
plt.hist(doc_lens_2020, bins = 1000, color = 'navy')

y_pos = max(plt.gca().get_ylim()) - 0.05 * max(plt.gca().get_ylim
    ())
spacing = 0.025 * max(plt.gca().get_ylim())

plt.text(mean_val_2021, y_pos, "Mean: " + str(mean_val_2021))
plt.text(median_val_2021, y_pos - spacing, "Median: " + str(
    median_val_2021))
plt.text(std_val_2021, y_pos - 2 * spacing, "Stdev: " + str(
    std_val_2021))
plt.text(quantile_01_val_2021, y_pos - 3 * spacing, "1%ile: " +
    str(quantile_01_val_2021))
plt.text(quantile_99_val_2021, y_pos - 4 * spacing, "99%ile: " +
```

```
    str(quantile_99_val_2021))

plt.gca().set(xlim = (0, 300), ylabel = 'Number of Documents',
    xlabel = 'Document Word Count')
plt.tick_params(size = 16)
plt.xticks(np.linspace(0,300,10))
plt.title('Distribution of Document Word Counts 2021', fontdict =
    dict(size = 22))
plt.show()

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
n_topics = 10

plt.figure(figsize=(18, 14), dpi=160)

for i in range(n_topics):
    plt.subplot(5, 2, i + 1)
    df_dominant_topic_sub = df_topic_sents_keywords.loc[
        df_topic_sents_keywords.Dominant_Topic == i, :]
    doc_lens_2 = [len(d.split()) for d in df_dominant_topic_sub.
        Original_Text]
    doc_lens_np = np.array(doc_lens_2)

    # Calculate statistics
    mean_val = np.mean(doc_lens_np)
    median_val = np.median(doc_lens_np)
    std_val = np.std(doc_lens_np)

    plt.hist(doc_lens_np, bins=1000, color=cols[i])
    plt.gca().twinx()
    density = gaussian_kde(doc_lens_np)
    xs = np.linspace(0, 1000, 1000)
    plt.plot(xs, density(xs), color='black')

    # Plot statistics
    plt.axvline(mean_val, color='red', linestyle='-', label=f'Mean
        : {mean_val}', alpha = 0.3)
    plt.axvline(median_val, color='green', linestyle='-', label=f'
        Median: {median_val}', alpha = 0.3)
    plt.axvline(mean_val + std_val, color='blue', linestyle='-',
        label=f'Std Dev: {std_val}', alpha = 0.3)
    plt.axvline(mean_val - std_val, color='blue', linestyle='-',
        alpha = 0.3)
```

```
        plt.xlabel('Document Word Count')
        plt.ylabel('Number of Documents', color=cols[i])
        plt.title(f'Topic: {i}', fontdict=dict(size=16, color=cols[i])
            )
        plt.legend()

plt.tight_layout()
plt.subplots_adjust(top=0.90)
plt.suptitle('Distribution of Document Word Counts by Dominant
    Topic for 2021', fontsize=22)
plt.show()



topics_2021 = model_desc_2021.show_topics(num_topics=10,formatted
    = False)
data_flat_2021 = [w for desc in desc_2021["clean_desc"].dropna()
    for w in str(desc).split()]
counter = Counter(data_flat_2021)

out = []
for i, topic in topics_2021:
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])

df_2021 = pd.DataFrame(out, columns = ['word', 'topic_id', '
    importance', 'word_count'])

fig, axes = plt.subplots(5, 2, figsize = (16,20), sharey = True,
    dpi = 160)
cols = get_colormap_colors(n_topics)
for i, ax in enumerate(axes.flatten()):
    if i >= len(topics_2021):
        ax.axis('off')
        continue
    ax.bar(x = 'word', height = "word_count", data = df_2021.loc[
        df_2021.topic_id == i, :], color = cols[i % len(cols)],
        width = 0.5, alpha = 0.3, label = 'Word Count')
    ax_twin = ax.twinx()
    ax_twin.bar(x = 'word', height = "importance", data = df_2021.
        loc[df_2021.topic_id == i, :], color = cols[i % len(cols)],
        width = 0.2, label = 'Weights')
```

```
    ax.set_ylabel('Word Count', color = cols[i % len(cols)])
    ax.set_title('Topic: ' + str(i), color = cols[i % len(cols)],
        fontsize = 16)
    ax.tick_params(axis = 'y', left = False)
    ax.set_xticklabels(df_2021.loc[df_2021.topic_id == i, 'word'],
         rotation = 30, horizontalalignment = 'right')
    ax.legend(loc = 'upper left'); ax_twin.legend(loc = 'upper
        right')

fig.tight_layout(w_pad = 2)
fig.suptitle('Word Count and Importance of Topic Keywords 2021',
    fontsize = 22, y = 1.05)
plt.show()

dominant_topics_2021, topic_percentages_2021 = topics_per_document
    (model = model_desc_2021, corpus = corpus_dec_2021, end = -1)

df_2_2021 = pd.DataFrame(dominant_topics_2021, columns = ['
    Document_Id', 'Dominant_Topic'])
dominant_topic_in_each_doc_2021 = df_2_2021.groupby('
    Dominant_Topic').size()
df_dominant_topic_in_each_doc_2021 =
    dominant_topic_in_each_doc_2021.to_frame(name = 'count').
    reset_index()

topic_weightage_by_doc_2021 = pd.DataFrame([dict(t) for t in
    topic_percentages_2021])
df_topic_weightage_by_doc_2021 = topic_weightage_by_doc_2021.sum()
    .to_frame(name = 'count').reset_index()

topic_top3words_2021 = [(i, topic) for i, topics in
    model_desc_2021.show_topics(formatted = False)
                                 for j, (topic, wt) in enumerate(
                                     topics) if j < 3]

df_top3words_stacked_2021 = pd.DataFrame(topic_top3words_2021,
    columns = ['topic_id', 'words'])
df_top3words_2021 = df_top3words_stacked_2021.groupby('topic_id').
    agg(','.join)
df_top3words_2021.reset_index(level = 0,inplace = True)

df_top3words_2021
```

```
topic_weights_2021 = []

for row_list in model_desc_2021[corpus_dec_2021]:
    weights = []

    for item in row_list:
        _, weight = item
        weights.append(weight)

    topic_weights_2021.append(weights)

arr_2021 = pd.DataFrame(topic_weights_2021).fillna(0).values

arr_2021 = arr_2021[np.amax(arr_2021, axis = 1) > 0.35]

topic_num_2021 = np.argmax(arr_2021, axis = 1)

# t-SNE Model
tsne_model_2021 = TSNE(n_components = 2, verbose = 1, random_state
    = 6, angle = .99, init = 'pca')
tsne_lda_2021 = tsne_model_2021.fit_transform(arr_2021)

n_topics = 10
mycolors = get_colormap_colors(n_topics)
mycolors_hex = [rgba_to_hex(color) for color in mycolors]

source = ColumnDataSource(data = dict(
    x = tsne_lda_2021[:, 0],
    y = tsne_lda_2021[:, 1],
    topic = topic_num_2021,
    color = [mycolors_hex[i] for i in topic_num_2021]
))

hover = HoverTool(tooltips = [("Topic", "@topic")])

mapper = linear_cmap(field_name = 'topic', palette =  mycolors_hex
    , low = min(topic_num_2021), high = max(topic_num_2021))

p = figure(width = 900, height = 700, title = "t-SNE Clustering of
    {} LDA Topics 2021".format(n_topics))
p.scatter(x = 'x', y = 'y', source = source, size = 8, color = '
    color', legend_field = 'topic', fill_alpha = 0.6)
p.add_tools(hover)
```

```
output_notebook()
show(p)

## 2022

desc_2022 = dataframes_by_year[3]

model_desc_2022, model_terms_2022, corpus_dec_2022,
    id2word_desc_2022 = LDA_gen(desc_2022["clean_description"], num
    = 10, vectorizer_type = 'tfidf')

model_terms_2022

log_perplexity = model_desc_2022.log_perplexity(corpus_dec_2022)

perplexity = np.exp2(-log_perplexity)

print(f'Log Perplexity: {log_perplexity}')
print(f'Perplexity: {perplexity}')

gensim_dict_2022 = convert_to_gensim_dictionary(id2word_desc_2022)

hdp_2022 = HdpModel(corpus_dec_2022, gensim_dict_2022)

all_topics_2022 = hdp_2022.get_topics()
num_of_topics_2022 = all_topics_2022.shape[0]
print(f"Total number of topics 2022: {num_of_topics_2022}")

# topics_2019 = hdp_2019.print_topics(num_of_topics_2019 =
    num_of_topics_2019)

texts_2022 = desc_2022['clean_desc'].apply(lambda x: x.split()).
    tolist()

coherence_model_hdp_2022 = CoherenceModel(model = hdp_2022, texts
    = texts_2022, dictionary = gensim_dict_2022, coherence = 'c_v')
coherence_hdp_2022 = coherence_model_hdp_2022.get_coherence()

coherence_hdp_2022

coherence_values_2022 = compute_coherence_for_each_topic(hdp_2022,
    texts_2022, gensim_dict_2022)
```

```
top_topics_2022 = sorted(range(len(coherence_values_2022)), key =
    lambda i: coherence_values_2022[i], reverse = True)[:10]

print("Top 10 Topics based on Coherence Scores 2022:")
for idx in top_topics_2022:
    print("Topic {}: Coherence Score: {:.4f}".format(idx,
        coherence_values_2022[idx]))

topics_range_2022 = range(1, len(coherence_values_2022) + 1)

plt.figure(figsize = (10,6))
plt.plot(topics_range_2022, coherence_values_2022, marker = 'o',
    linestyle = '--')
plt.title('Coherence Scores by Number of Topics 2022')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

sorted_coherence_values_2022 = sorted(coherence_values_2022,
    reverse=True)

topics_range_2022 = range(1, len(sorted_coherence_values_2022) +
    1)

plt.figure(figsize = (10,6))
plt.plot(topics_range_2022, sorted_coherence_values_2022, marker =
    'o', linestyle = '--')
plt.title('Sorted Coherence Scores by Number of Topics 2022')
plt.xlabel('Number of Topics')
plt.ylabel('Coherence Score')
plt.grid(True)
plt.show()

df_topic_sents_keywords_2022 = format_topics_sentences(ldamodel =
    model_desc_2022, corpus = corpus_dec_2022, texts = desc_2022["
    clean_desc"].tolist())

df_topic_sents_keywords_2022

doc_lens_2022 = [len(d.split()) for d in
    df_topic_sents_keywords_2022.Original_Text]
```

```
mean_val_2022 = round(np.mean(doc_lens_2022))
median_val_2022 = round(np.median(doc_lens_2022))
std_val_2022 = round(np.std(doc_lens_2022))
quantile_01_val_2022 = round(np.quantile(doc_lens_2022, q = 0.01))
quantile_99_val_2022 = round(np.quantile(doc_lens_2022, q = 0.99))

plt.figure(figsize = (16,7), dpi = 160)
plt.hist(doc_lens_2020, bins = 1000, color = 'navy')

y_pos = max(plt.gca().get_ylim()) - 0.05 * max(plt.gca().get_ylim
    ())
spacing = 0.025 * max(plt.gca().get_ylim())

plt.text(mean_val_2022, y_pos, "Mean: " + str(mean_val_2022))
plt.text(median_val_2022, y_pos - spacing, "Median: " + str(
    median_val_2022))
plt.text(std_val_2022, y_pos - 2 * spacing, "Stdev: " + str(
    std_val_2022))
plt.text(quantile_01_val_2022, y_pos - 3 * spacing, "1%ile: " +
    str(quantile_01_val_2022))
plt.text(quantile_99_val_2022, y_pos - 4 * spacing, "99%ile: " +
    str(quantile_99_val_2022))

plt.gca().set(xlim = (0, 300), ylabel = 'Number of Documents',
    xlabel = 'Document Word Count')
plt.tick_params(size = 16)
plt.xticks(np.linspace(0,300,10))
plt.title('Distribution of Document Word Counts 2022', fontdict =
    dict(size = 22))
plt.show()

cols = [color for name, color in mcolors.TABLEAU_COLORS.items()]
n_topics = 10

plt.figure(figsize=(18, 14), dpi=160)

for i in range(n_topics):
    plt.subplot(5, 2, i + 1)
    df_dominant_topic_sub = df_topic_sents_keywords.loc[
        df_topic_sents_keywords.Dominant_Topic == i, :]
    doc_lens_2 = [len(d.split()) for d in df_dominant_topic_sub.
        Original_Text]
```

```
    doc_lens_np = np.array(doc_lens_2)

    # Calculate statistics
    mean_val = np.mean(doc_lens_np)
    median_val = np.median(doc_lens_np)
    std_val = np.std(doc_lens_np)

    plt.hist(doc_lens_np, bins=1000, color=cols[i])
    plt.gca().twinx()
    density = gaussian_kde(doc_lens_np)
    xs = np.linspace(0, 1000, 1000)
    plt.plot(xs, density(xs), color='black')

    # Plot statistics
    plt.axvline(mean_val, color='red', linestyle='-', label=f'Mean
        : {mean_val}', alpha = 0.3)
    plt.axvline(median_val, color='green', linestyle='-', label=f'
        Median: {median_val}', alpha = 0.3)
    plt.axvline(mean_val + std_val, color='blue', linestyle='-',
        label=f'Std Dev: {std_val}', alpha = 0.3)
    plt.axvline(mean_val - std_val, color='blue', linestyle='-',
        alpha = 0.3)

    plt.xlabel('Document Word Count')
    plt.ylabel('Number of Documents', color=cols[i])
    plt.title(f'Topic: {i}', fontdict=dict(size=16, color=cols[i])
        )
    plt.legend()

plt.tight_layout()
plt.subplots_adjust(top=0.90)
plt.suptitle('Distribution of Document Word Counts by Dominant
    Topic for 2022', fontsize=22)
plt.show()



topics_2022 = model_desc_2022.show_topics(num_topics=10,formatted
    = False)
data_flat_2022 = [w for desc in desc_2022["clean_desc"].dropna()
    for w in str(desc).split()]
counter = Counter(data_flat_2022)
```

```
out = []
for i, topic in topics_2022:
    for word, weight in topic:
        out.append([word, i , weight, counter[word]])

df_2022 = pd.DataFrame(out, columns = ['word', 'topic_id', '
    importance', 'word_count'])

fig, axes = plt.subplots(5, 2, figsize = (16,20), sharey = True,
    dpi = 160)
cols = get_colormap_colors(n_topics)
for i, ax in enumerate(axes.flatten()):
    if i >= len(topics_2022):
        ax.axis('off')
        continue
    ax.bar(x = 'word', height = "word_count", data = df_2022.loc[
        df_2022.topic_id == i, :], color = cols[i % len(cols)],
        width = 0.5, alpha = 0.3, label = 'Word Count')
    ax_twin = ax.twinx()
    ax_twin.bar(x = 'word', height = "importance", data = df_2022.
        loc[df_2022.topic_id == i, :], color = cols[i % len(cols)],
         width = 0.2, label = 'Weights')
    ax.set_ylabel('Word Count', color = cols[i % len(cols)])
    ax.set_title('Topic: ' + str(i), color = cols[i % len(cols)],
        fontsize = 16)
    ax.tick_params(axis = 'y', left = False)
    ax.set_xticklabels(df_2022.loc[df_2022.topic_id == i, 'word'],
         rotation = 30, horizontalalignment = 'right')
    ax.legend(loc = 'upper left'); ax_twin.legend(loc = 'upper
        right')

fig.tight_layout(w_pad = 2)
fig.suptitle('Word Count and Importance of Topic Keywords 2022',
    fontsize = 22, y = 1.05)
plt.show()

dominant_topics_2022, topic_percentages_2022 = topics_per_document
    (model = model_desc_2022, corpus = corpus_dec_2022, end = -1)

df_2_2022 = pd.DataFrame(dominant_topics_2022, columns = ['
    Document_Id', 'Dominant_Topic'])
dominant_topic_in_each_doc_2022 = df_2_2022.groupby('
    Dominant_Topic').size()
```

```
df_dominant_topic_in_each_doc_2022 =
    dominant_topic_in_each_doc_2022.to_frame(name = 'count').
    reset_index()

topic_weightage_by_doc_2022 = pd.DataFrame([dict(t) for t in
    topic_percentages_2022])
df_topic_weightage_by_doc_2022 = topic_weightage_by_doc_2022.sum()
    .to_frame(name = 'count').reset_index()

topic_top3words_2022 = [(i, topic) for i, topics in
    model_desc_2022.show_topics(formatted = False)
                                 for j, (topic, wt) in enumerate(
                                     topics) if j < 3]

df_top3words_stacked_2022 = pd.DataFrame(topic_top3words_2022,
    columns = ['topic_id', 'words'])
df_top3words_2022 = df_top3words_stacked_2022.groupby('topic_id').
    agg(','.join)
df_top3words_2022.reset_index(level = 0,inplace = True)

df_top3words_2022

topic_weights_2022 = []

for row_list in model_desc_2022[corpus_dec_2022]:
    weights = []

    for item in row_list:
        _, weight = item
        weights.append(weight)

    topic_weights_2022.append(weights)

arr_2022 = pd.DataFrame(topic_weights_2022).fillna(0).values

arr_2022 = arr_2022[np.amax(arr_2022, axis = 1) > 0.35]

topic_num_2022 = np.argmax(arr_2022, axis = 1)

# t-SNE Model
tsne_model_2022 = TSNE(n_components = 2, verbose = 1, random_state
    = 6, angle = .99, init = 'pca')
tsne_lda_2022 = tsne_model_2022.fit_transform(arr_2022)
```

```
n_topics = 10
mycolors = get_colormap_colors(n_topics)
mycolors_hex = [rgba_to_hex(color) for color in mycolors]
source = ColumnDataSource(data = dict(
    x = tsne_lda_2022[:, 0],
    y = tsne_lda_2022[:, 1],
    topic = topic_num_2022,
    color = [mycolors_hex[i] for i in topic_num_2022]
))

hover = HoverTool(tooltips=[("Topic", "@topic")])

mapper = linear_cmap(field_name = 'topic', palette = mycolors_hex,
    low = min(topic_num_2022), high = max(topic_num_2022))

p = figure(width = 900, height = 700, title = "t-SNE Clustering of
    {} LDA Topics 2022".format(n_topics))
p.scatter(x = 'x', y = 'y', source = source, size = 8, color = '
    color', legend_field = 'topic', fill_alpha = 0.6)
p.add_tools(hover)

output_notebook()
show(p)
```

```
X_train, X_test, y_train, y_test = train_test_split(df_mega_2["
    clean_description"], df_mega_2["label"], test_size = 0.3,
    random_state = 6, stratify = df_mega_2["label"])

print(X_train.shape)

def get_topic_distribution_for_descriptions(lda_model,
    descriptions):

    vectorizer_desc = CountVectorizer()

    dtm_desc = vectorizer_desc.fit_transform(descriptions)

    corpus_desc = Sparse2Corpus(dtm_desc.T)
```

```
    topic_distributions = [lda_model[d] for d in corpus_desc]

    dense_topic_distributions = np.zeros((len(descriptions),
        lda_model.num_topics))

    for i, doc_topics in enumerate(topic_distributions):
        for topic, prob in doc_topics:
            dense_topic_distributions[i, topic] = prob

    return dense_topic_distributions

class LDATransformer(BaseEstimator, TransformerMixin):
    def __init__(self, lda_model = None, num_topics = 10):
        self.num_topics = num_topics
        self.lda_model = lda_model
        self.vectorizer = CountVectorizer()

    def fit(self, X, y = None):
        if not self.lda_model:

            dtm_desc = self.vectorizer.fit_transform(X)

            id2word_desc = {v: k for k, v in self.vectorizer.
                vocabulary_.items()}

            corpus_desc = Sparse2Corpus(dtm_desc.T)

            self.lda_model = LdaModel(corpus = corpus_desc,
                num_topics = self.num_topics, id2word =
                id2word_desc, passes = 15)
        return self

    def transform(self, X):
        if self.lda_model is None:
            raise ValueError("LDA model not trained!")
        return get_topic_distribution_for_descriptions(self.
            lda_model, X)

X = df_mega_2["clean_description"]
y = df_mega_2["label"]

### Dummy Classifier
```

```
dummy = DummyClassifier(strategy = "most_frequent")

dummy.fit(X_train, y_train)
y_pred_dum = dummy.predict(X_test)

print("Accuracy :", accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred_dum))

### Logistic Regression

combined_features = FeatureUnion([
    ('tfidf', TfidfVectorizer()),
    ('lda', LDATransformer(lda_model = lda_model_desc))
])

pipeline_lr = Pipeline([
    ('features', combined_features),
    ('clf', LogisticRegression(max_iter = 1000, multi_class = '
        multinomial', solver = 'lbfgs'))
])

## tried k fold sampling

skf = StratifiedKFold(n_splits = 5, shuffle = True, random_state =
    6)

for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    pipeline_lr.fit(X_train, y_train)

    y_pred = pipeline_lr.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    pres = precision_score(y_test, y_pred, average = 'weighted')
    recall = recall_score(y_test, y_pred, average = 'weighted')
    f1_score = f1_metric(y_test, y_pred, average = 'weighted')
    print("Accuracy Score for this fold:", accuracy)
    print("Prescision Score for this fold:", pres)
    print("Recall Score for this fold:", recall)
    print("F1 Score for this fold:", f1_score)
```

```
    print("Next fold")



pipeline_lr.fit(X_train, y_train)
y_pred = pipeline_lr.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
pres = precision_score(y_test, y_pred, average = 'weighted')
recall = recall_score(y_test, y_pred, average = 'weighted')
f1_score = f1_metric(y_test, y_pred, average = 'weighted')

print("Accuracy :", accuracy)
print("Prescision :", pres)
print("Recall :", recall)
print("F1_Score :", f1_score)

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (10, 7))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

f1_scores = [f1_metric(y_test, y_pred, labels = [i], average='
    weighted') for i in range(4)]

plt.bar(range(4), f1_scores)
plt.xticks(range(4))
plt.xlabel('Classes')
plt.ylabel('F1 Score')
plt.title('F1 Score per Class')
plt.show()

print(classification_report(y_test, y_pred))

### Logistic Regression L1

pipeline_lr1 = Pipeline([
    ('features', combined_features),
    ('clf', LogisticRegression(max_iter = 1000, multi_class = '
```

```
        multinomial', penalty = 'l1', solver = 'saga', random_state
        = 6))
])


pipeline_lr1.fit(X_train, y_train)
y_pred_lr1 = pipeline_lr1.predict(X_test)

for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    pipeline_lr1.fit(X_train, y_train)

    y_pred = pipeline_lr1.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    pres = precision_score(y_test, y_pred, average = 'weighted')
    recall = recall_score(y_test, y_pred, average = 'weighted')
    f1_score = f1_metric(y_test, y_pred, average = 'weighted')
    print("Accuracy Score for this fold:", accuracy)
    print("Prescision Score for this fold:", pres)
    print("Recall Score for this fold:", recall)
    print("F1 Score for this fold:", f1_score)
    print("Next fold")

accuracy_lr1 = accuracy_score(y_test, y_pred_lr1)
pres_lr1 = precision_score(y_test, y_pred_lr1, average = 'weighted
    ')
recall_lr1 = recall_score(y_test, y_pred_lr1, average = 'weighted'
    )
f1_score_lr1 = f1_metric(y_test, y_pred_lr1, average = 'weighted')

print("Accuracy :", accuracy_lr1)
print("Prescision :", pres_lr1)
print("Recall :", recall_lr1)
print("F1_Score :", f1_score_lr1)

cm_l1 = confusion_matrix(y_test, y_pred_lr1)
plt.figure(figsize = (10, 7))
sns.heatmap(cm_l1, annot = True, cmap = 'Blues', fmt = 'g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
f1_scores_l1 = [f1_metric(y_test, y_pred_lr1, labels = [i],
    average='weighted') for i in range(4)]

plt.bar(range(4), f1_scores_l1)
plt.xticks(range(4))
plt.xlabel('Classes')
plt.ylabel('F1 Score')
plt.title('F1 Score per Class')
plt.show()

print(classification_report(y_test, y_pred_lr1))

### Logistic Regression L2

pipeline_lr2 = Pipeline([
    ('features', combined_features),
    ('clf', LogisticRegression(max_iter = 1000, multi_class = '
        multinomial', penalty = 'l2', solver = 'lbfgs',
        random_state = 6))
])

for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    pipeline_lr2.fit(X_train, y_train)

    y_pred = pipeline_lr2.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    pres = precision_score(y_test, y_pred, average = 'weighted')
    recall = recall_score(y_test, y_pred, average = 'weighted')
    f1_score = f1_metric(y_test, y_pred, average = 'weighted')
    print("Accuracy Score for this fold:", accuracy)
    print("Prescision Score for this fold:", pres)
    print("Recall Score for this fold:", recall)
    print("F1 Score for this fold:", f1_score)
    print("Next fold")

pipeline_lr2.fit(X_train, y_train)
y_pred_lr2 = pipeline_lr2.predict(X_test)

accuracy_lr2 = accuracy_score(y_test, y_pred_lr2)
```

```
pres_lr2 = precision_score(y_test, y_pred_lr2, average = 'weighted
    ')
recall_lr2 = recall_score(y_test, y_pred_lr2, average = 'weighted'
    )
f1_score_lr2 = f1_metric(y_test, y_pred_lr2, average = 'weighted')

print("Accuracy :", accuracy_lr2)
print("Prescision :", pres_lr2)
print("Recall :", recall_lr2)
print("F1 Score :", f1_score_lr2)

cm_l2 = confusion_matrix(y_test, y_pred_lr2)
plt.figure(figsize = (10, 7))
sns.heatmap(cm_l2, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

f1_scores_l2 = [f1_metric(y_test, y_pred_lr2, labels = [i],
    average='weighted') for i in range(4)]

plt.bar(range(4), f1_scores_l2)
plt.xticks(range(4))
plt.xlabel('Classes')
plt.ylabel('F1 Score')
plt.title('F1 Score per Class')
plt.show()

print(classification_report(y_test, y_pred_lr2))

### Logistic Regression L1-2

pipeline_lr12 = Pipeline([
    ('features', combined_features),
    ('clf', LogisticRegression(max_iter = 1000, penalty = '
        elasticnet', solver = 'saga', l1_ratio = 0.5, random_state
        = 6))
])

pipeline_lr12.fit(X_train, y_train)
y_pred_lr12 = pipeline_lr12.predict(X_test)

accuracy_lr12 = accuracy_score(y_test, y_pred_lr12)
```

```
pres_lr12 = precision_score(y_test, y_pred_lr12, average = '
    weighted')
recall_lr12 = recall_score(y_test, y_pred_lr12, average = '
    weighted')
f1_score_lr12 = f1_metric(y_test, y_pred_lr12, average = 'weighted
    ')

print("Accuracy :", accuracy_lr12)
print("Prescision :", pres_lr12)
print("Recall :", recall_lr12)
print("F1_Score :", f1_score_lr12)

cm_l12 = confusion_matrix(y_test, y_pred_lr12)
plt.figure(figsize = (10, 7))
sns.heatmap(cm_l12, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

f1_scores_l12 = [f1_metric(y_test, y_pred_lr12, labels = [i],
    average='weighted') for i in range(4)]

plt.bar(range(4), f1_scores_l12)
plt.xticks(range(4))
plt.xlabel('Classes')
plt.ylabel('F1 Score')
plt.title('F1 Score per Class')
plt.show()

print(classification_report(y_test, y_pred_lr2))

### Descision Tree

pipeline_dt = Pipeline([
    ('features', combined_features),
    ('clf', DecisionTreeClassifier(random_state = 6))
])

pipeline_dt.fit(X_train, y_train)
y_pred_dt = pipeline_dt.predict(X_test)

accuracy_dt = accuracy_score(y_test, y_pred_dt)
pres_dt = precision_score(y_test, y_pred_dt, average = 'weighted')
```

```
recall_dt = recall_score(y_test, y_pred_dt, average = 'weighted')
f1_score_dt = f1_metric(y_test, y_pred_dt, average = 'weighted')

print("Accuracy :", accuracy_dt)
print("Prescision :", pres_dt)
print("Recall :", recall_dt)
print("F1_Score :", f1_score_dt)

cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize = (10, 7))
sns.heatmap(cm_dt, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

f1_scores_dt = [f1_metric(y_test, y_pred_dt, labels = [i], average
   ='weighted') for i in range(4)]

plt.bar(range(4), f1_scores_dt)
plt.xticks(range(4))
plt.xlabel('Classes')
plt.ylabel('F1 Score')
plt.title('F1 Score per Class')
plt.show()

print(classification_report(y_test, y_pred_dt))

### Random Forest

pipeline_rf = Pipeline([
    ('features', combined_features),
    ('clf', RandomForestClassifier(n_estimators = 100,
        random_state = 6))
])

pipeline_rf.fit(X_train, y_train)
y_pred_rf = pipeline_rf.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)
pres_rf = precision_score(y_test, y_pred_rf, average = 'weighted')
recall_rf = recall_score(y_test, y_pred_rf, average = 'weighted')
f1_score_rf = f1_metric(y_test, y_pred_rf, average = 'weighted')
```

```
print("Accuracy :", accuracy_rf)
print("Prescision :", pres_rf)
print("Recall :", recall_rf)
print("F1_Score :", f1_score_rf)

cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize = (10, 7))
sns.heatmap(cm_rf, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

f1_scores_rf = [f1_metric(y_test, y_pred_rf, labels = [i], average
    ='weighted') for i in range(4)]

plt.bar(range(4), f1_scores_rf)
plt.xticks(range(4))
plt.xlabel('Classes')
plt.ylabel('F1 Score')
plt.title('F1 Score per Class')
plt.show()

print(classification_report(y_test, y_pred_rf))
```

```
### Support Vector Machines

pipeline_svm = Pipeline([
    ('features', combined_features),
    ('clf', SVC(kernel = 'linear', C = 1, probability = True,
        random_state = 6))
])

pipeline_svm.fit(X_train, y_train)
y_pred_svm = pipeline_svm.predict(X_test)

accuracy_svm = accuracy_score(y_test, y_pred_svm)
pres_svm = precision_score(y_test, y_pred_svm, average = 'weighted
    ')
recall_svm = recall_score(y_test, y_pred_svm, average = 'weighted'
    )
f1_score_svm = f1_metric(y_test, y_pred_svm, average = 'weighted')

print("Accuracy :", accuracy_svm)
print("Prescision :", pres_svm)
print("Recall :", recall_svm)
print("F1_Score :", f1_score_svm)

cm_svm = confusion_matrix(y_test, y_pred_svm)
plt.figure(figsize = (10, 7))
sns.heatmap(cm_svm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

f1_scores_svm = [f1_metric(y_test, y_pred_svm, labels = [i],
    average='weighted') for i in range(4)]

plt.bar(range(4), f1_scores_svm)
plt.xticks(range(4))
plt.xlabel('Classes')
plt.ylabel('F1 Score')
plt.title('F1 Score per Class')
plt.show()

print(classification_report(y_test, y_pred_svm))
```

```
### LSTM LDA

document_topic_dists = [lda_model_desc.get_document_topics(doc_bow
    ) for doc_bow in lda_corpus_dec]

document_topic_matrix = np.zeros((len(document_topic_dists), 10))

for idx, doc_topics in enumerate(document_topic_dists):
    for topic_id, proportion in doc_topics:
        document_topic_matrix[idx, topic_id] = proportion

print(document_topic_matrix.shape)

document_topic_dists_tensor = torch.tensor(document_topic_matrix,
    dtype=torch.float32)

def tokens_to_ids(tokens):
    token_ids = []

    for token in tokens:

        if token in vocab:
            token_ids.append(vocab[token])
        else:
            token_ids.append(vocab["<UNK>"])

    return token_ids

word_freq = Counter()

for z in df_mega_2["tokens"]:
    for y in z:
        word_freq[y] += 1

vocab = {}
for idx, (word, freq) in enumerate(word_freq.items()):
    if freq >= 3:
```

```
            vocab[word] = idx

vocab["<PAD>"] = 0
vocab["<UNK>"] = len(vocab)

len(vocab)

df_mega_2["token_ids"] = df_mega_2["tokens"].apply(tokens_to_ids)

MAX_SEQ_LEN = 100

def pad_sequence(seq, maxlen = MAX_SEQ_LEN):
    return seq[:maxlen] + [vocab["<PAD>"]] * (maxlen - len(seq))

df_mega_2["token_ids_padded"] = df_mega_2["token_ids"].apply(
    pad_sequence)

max_token_id = df_mega_2["token_ids_padded"].apply(max).max()
min_token_id = df_mega_2["token_ids_padded"].apply(min).min()
print(f'Max token ID: {max_token_id}')
print(f'Min token ID: {min_token_id}')



class TextDataset(Dataset):
    def __init__(self, token_ids, labels, lda_topic_dists):
        self.token_ids = token_ids
        self.labels = labels
        self.lda_topic_dists = lda_topic_dists

    def __len__(self):
        return len(self.token_ids)

    def __getitem__(self, idx):
        token_tensor = torch.tensor(self.token_ids[idx], dtype=
            torch.long)
        label_tensor = torch.tensor(self.labels[idx], dtype=torch.
            long)
        lda_tensor = torch.tensor(self.lda_topic_dists[idx], dtype
            =torch.float)
        return token_tensor, label_tensor, lda_tensor

sss = StratifiedShuffleSplit(n_splits = 1, test_size = 0.2,
```

```
    random_state = 6)

for temp_train_idx, test_idx in sss.split(df_mega_2["
    token_ids_padded"].tolist(), df_mega_2["label"].tolist()):

    temp_train_texts = [df_mega_2["token_ids_padded"].tolist()[i]
        for i in temp_train_idx]

    temp_train_labels = [df_mega_2["label"].tolist()[i] for i in
        temp_train_idx]

    temp_train_lda_dists = [document_topic_dists_tensor.numpy()[i]
         for i in temp_train_idx]

    test_texts = [df_mega_2["token_ids_padded"].tolist()[i] for i
        in test_idx]
    test_labels = [df_mega_2["label"].tolist()[i] for i in
        test_idx]
    test_lda_dists = [document_topic_dists_tensor.numpy()[i] for i
         in test_idx]

sss1 = StratifiedShuffleSplit(n_splits = 1, test_size = 0.25,
    random_state = 6)

for train_idx, val_idx in sss1.split(temp_train_texts,
    temp_train_labels):
    train_texts = [temp_train_texts[i] for i in train_idx]
    train_labels = [temp_train_labels[i] for i in train_idx]
    train_lda_dists = [temp_train_lda_dists[i] for i in train_idx]

    val_texts = [temp_train_texts[i] for i in val_idx]
    val_labels = [temp_train_labels[i] for i in val_idx]
    val_lda_dists = [temp_train_lda_dists[i] for i in val_idx]

train_dataset = TextDataset(train_texts, train_labels,
    train_lda_dists)
val_dataset = TextDataset(val_texts, val_labels, val_lda_dists)
test_dataset = TextDataset(test_texts, test_labels, test_lda_dists
    )

train_dataloader = DataLoader(train_dataset, batch_size = 64,
    shuffle = True)
val_dataloader = DataLoader(val_dataset, batch_size = 64, shuffle
```

```
    = False)
test_dataloader = DataLoader(test_dataset, batch_size = 64,
    shuffle = False)

class LDA_LSTM_Model(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab_size,
        num_topics, output_dim_keywords):
        super(LDA_LSTM_Model, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim)

        # Compute combined_size
        lstm_output_size = hidden_dim
        lda_topic_dist_size = num_topics
        combined_size = lstm_output_size + lda_topic_dist_size

        self.fc1 = nn.Linear(combined_size, 4)
        self.fc2 = nn.Linear(combined_size, output_dim_keywords)

    def forward(self, text, lda_topic_dist):
        embedded = self.embedding(text)
        lstm_out, _ = self.lstm(embedded)
        lstm_out = lstm_out[:, -1, :]
        #print("LSTM Output Size:", lstm_out.shape)
        #print("lda_topic_dist shape",lda_topic_dist.shape)
        combined_out = torch.cat((lstm_out, lda_topic_dist), dim
            =1)
        #print("combined_out shape:", combined_out.shape)
        severity = self.fc1(combined_out)
        keywords = F.softmax(self.fc2(combined_out), dim = 1)

        return severity, keywords

num_embeddings = max_token_id + 1

lstm_model = LDA_LSTM_Model(embedding_dim = num_embeddings,
    hidden_dim = 120, vocab_size = 5000, num_topics = 10,
    output_dim_keywords = 1000)

lstm_criterion = nn.CrossEntropyLoss()
lstm_optimizer = torch.optim.Adam(lstm_model.parameters(), lr =
    0.001625, betas=(0.9, 0.999))
```

```
ROOT = r"C:/Users/Dell/Documents/Jupyter/Dissertation/"
results_path_lstm = ROOT + "resultl"

def train_model(load, model, optimizer, criterion):
    model.train()
    total_loss, correct, total = 0, 0, 0

    for data in load:
        inputs, labels, lda_dists = data

        optimizer.zero_grad()

        severity_output, keywords_output = model(inputs, lda_dists
            )

        loss = criterion(severity_output, labels)

        loss.backward()

        optimizer.step()

        pred = severity_output.data.max(1)[1]

        correct += pred.eq(labels.data).sum()

        total += severity_output.size(0)

        total_loss += loss.data.item()

    avg_loss = total_loss / len(load)

    acc = correct / total

    return avg_loss, acc

def validate_model(load, model, criterion):
    model.eval()
    val_loss, correct, total = 0, 0, 0

    with torch.no_grad():
        for inputs, labels, lda_dists in load:
```

```
            severity_output, keywords_output = model(inputs,
                lda_dists)

            loss = criterion(severity_output, labels)

            pred = severity_output.data.max(1)[1]

            correct += pred.eq(labels.data).sum()

            total += labels.size(0)

            val_loss += loss.data.item()

    avg_val_loss = val_loss / len(load)

    val_acc = correct / total

    return avg_val_loss, val_acc

train_loss = []
train_acc = []
val_loss = []
val_acc = []
epochs = 10
for epoch in range(epochs):
    tloss, tacc = train_model(train_dataloader, lstm_model,
        lstm_optimizer, lstm_criterion)
    vloss, vacc = validate_model(val_dataloader, lstm_model,
        lstm_criterion)

    print(f"Epoch {epoch} Training loss: {tloss} Training accuracy
        : {tacc:.1%} Validation loss: {vloss} Validation accuracy:
        {vacc:.1%}")

    train_loss.append(tloss)
    train_acc.append(tacc)
    val_loss.append(vloss)
    val_acc.append(vacc)

torch.save(lstm_model.state_dict(), results_path_lstm)


plt.figure(figsize = (12, 6))
```

```
plt.subplot(1, 2, 1)
plt.plot(train_loss, label = 'Training Loss')
plt.plot(val_loss, label = 'Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss Curve')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_acc, label = 'Training Accuracy')
plt.plot(val_acc, label = 'Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy Curve')
plt.legend()

plt.show()



model_load = LDA_LSTM_Model(embedding_dim = num_embeddings,
    hidden_dim = 120, vocab_size = 5000, num_topics = 10,
    output_dim_keywords = 1000)
model_load.load_state_dict(torch.load(results_path_lstm))
model_load.eval()

all_predictions = []
all_labels = []
correct = 0
total = 0

with torch.no_grad():
    for inputs, labels, lda_dists in test_dataloader:
        severity_output, keywords_output = model_load(inputs,
            lda_dists)

        # Get the predicted class (For the severity output)
        _, predicted = torch.max(severity_output, 1)
        all_predictions.extend(predicted.tolist())

        # If you have labels and you want to compute accuracy
        all_labels.extend(labels.tolist())
        correct += predicted.eq(labels).sum().item()
```

```
        total += labels.size(0)


accuracy = correct / total
print(f"Test Accuracy: {accuracy:.2%}")

df_result = pd.DataFrame({'True_Labels': all_labels, 'Predictions'
    : all_predictions})

print(confusion_matrix(all_labels, all_predictions))

matrix = confusion_matrix(all_labels, all_predictions)

plt.figure(figsize = (10,7))
sns.heatmap(matrix, annot = True, cmap = 'Blues', fmt = 'g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print(classification_report(all_labels, all_predictions))



n_classes = 4

all_labels_bin = label_binarize(all_labels, classes = range(
    n_classes))
all_predictions_bin = label_binarize(all_predictions, classes =
    range(n_classes))

plt.figure()
for i in range(n_classes):
    fpr, tpr, _ = roc_curve(all_labels_bin[:, i],
        all_predictions_bin[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw = 2, label = f'Class {i} (area = {
        roc_auc:.2f})')

fpr, tpr, _ = roc_curve(all_labels_bin.ravel(),
    all_predictions_bin.ravel())
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color = 'darkorange', lw = 2, label = f'Micro-
    average (area = {roc_auc:.2f})')
```

```
plt.plot([0, 1], [0, 1], color='navy', lw = 2, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Multi-Class')
plt.legend(loc = "lower right")
plt.show()

all_keyword_predictions = []

with torch.no_grad():
    for inputs, labels, lda_dists in test_dataloader:
        severity_output, keywords_output = model_load(inputs,
            lda_dists)


        _, predicted_severity = torch.max(severity_output, 1)
        all_predictions.extend(predicted_severity.tolist())


        _, predicted_keywords = torch.max(keywords_output, 1)
        all_keyword_predictions.extend(predicted_keywords.tolist()
            )


        all_labels.extend(labels.tolist())
        correct += predicted_severity.eq(labels).sum().item()
        total += labels.size(0)

accuracy = correct / total
print(f"Test Accuracy: {accuracy:.2%}")

id_to_token = {v: k for k, v in vocab.items()}

actual_tokens = []

for token_id in all_keyword_predictions:
    token = id_to_token.get(token_id, "<UNK>")
    actual_tokens.append(token)

unique_values_list = []
```

```
unique = set()

for item in actual_tokens:
    if item not in unique:
        unique_values_list.append(item)
        unique.add(item)

print("Unique Values in Order:", unique_values_list)

print("Predicted Tokens:", actual_tokens)

predictions_dict = {}
actual_tokens = []

with torch.no_grad():
    for inputs, labels, lda_dists in test_dataloader:
        severity_output, keywords_output = model_load(inputs,
            lda_dists)

        _, predicted_severity = torch.max(severity_output, 1)

        _, predicted_keywords = torch.max(keywords_output, 1)

        for i in range(len(predicted_severity)):
            sample_id = i

            keyword_token = id_to_token.get(predicted_keywords[i].
                item(), "<UNK>")
            actual_tokens.append(keyword_token)

            predictions_dict[sample_id] = {
                'severity': predicted_severity[i].item(),
                'keywords': keyword_token
            }

predictions_dict

predictions_dict = {}

with torch.no_grad():
    for inputs, labels, lda_dists in test_dataloader:
        severity_output, keywords_output = model_load(inputs,
            lda_dists)
```

```
        _, predicted_severity = torch.max(severity_output, 1)


        topk_values, topk_indices = torch.topk(keywords_output, k
            =5, dim=1)

        for i in range(len(predicted_severity)):
            sample_id = i
            unique_keywords = list(set(topk_indices[i].tolist()))
            predictions_dict[sample_id] = {
                'severity': predicted_severity[i].item(),
                'keywords': unique_keywords
            }



# predictions_df = pd.DataFrame(columns=['Sample_ID', '
    Predicted_Severity', 'Predicted_Keywords'])

# with torch.no_grad():
#     for inputs, labels, lda_dists in test_dataloader:
#         severity_output, keywords_output = model_load(inputs,
    lda_dists)

#
#         _, predicted_severity = torch.max(severity_output, 1)

#
#         _, predicted_keywords = torch.max(keywords_output, 1)

#         for i in range(len(predicted_severity)):
#             sample_id = i  # Replace with actual sample ID if
    available
#             new_row = {'Sample_ID': sample_id, '
    Predicted_Severity': predicted_severity[i].item(), '
    Predicted_Keywords': predicted_keywords[i].item()}
#             predictions_df = predictions_df.append(new_row,
    ignore_index=True)
```