“COVID-19 Vaccine Types Sentiment Analysis with Data Augmentation and CNN-LSTM”

by

Mathias Yeremia Aryadi (吴智恩)

A Thesis

Submitted to the Faculty of Nanjing Xiaozhuang University

in Partial Fulfillment of the Requirements for the degree of

Bachelor of Software Engineering

School of Information Engineering

Fangshan, Nanjing

May 2022

To:

Dean Xiangjun Zhao

School of Information Engineering

This thesis, written by Mathias Yeremia Aryadi (吴智恩), and entitled “COVID-19 Vaccine Type Sentiment Analysis with Data Augmentation and CNN-LSTM”, having been approved in respect to style and intellectual content, is referred to you for judgment.

I have read this thesis and recommend that it be approved.

Committee A

Committee B

Committee C

Date Defense: May 24, 2022

The thesis of Mathias Yeremia Aryadi (吴智恩) is approved

Dean Xiangjun Zhao

School of Information Engineering

Xxx

Chief Foreign Affairs Office

Nanjing Xiaozhuang University, 2022

DECLARATION OF ORGINALITY

I, the undersigned below:

Student Name	:	MATHIAS YEREMIA ARYADI (吴智恩)
Student ID	:	L20253011 / 2017103126
Study Program	:	SOFTWARE ENGINEERING
Degree	:	BACHELOR'S DEGREE
Faculty	:	SCHOOL OF INFORMATION TECHNOLOGY

Hereby declared that the thesis I wrote with the title:

"**COVID-19 Vaccine Type Sentiment Analysis with Data Augmentation and CNN-LSTM."**

1. It is truly research written and conducted purely by myself, not copying from other published researchers, and not a result of plagiarism.
2. I will allow Nanjing Xiaozhuang University and Kalbis Insitute to manage and keep the copy of this thesis to be used as they deem necessary.

I made this statement of a declaration with full responsibility, and I am willing to accept any consequences according to the rules and regulations should the statement above proved to be wrong in any way.

Jakarta, 03 May 2022

Mathias Yeremia Aryadi

L20253011 / 2017103126

COVID-19 VACCINE TYPE SENTIMENT ANALYSIS

WITH DATA AUGMENTATION AND CNN-LSTM

ABSTRACT

Abstract: *Twitter, one of the most used and popular social media has produced a huge information on the internet. With several COVID-19 vaccine types discovered in this era of pandemic, this becomes a huge trend where people discuss it in social media. Now, people can share their various sentiments about the COVID-19 vaccine types. Sentiment analysis can help us to extract emotions behind the people's sentiment, whether it is positive or negative sentiment. Therefore, the aims of this research are to build and compare the sentiment analysis with hybrid CNN-LSTM model, single CNN model, and single LSTM with data augmentation on the four dataset topics about COVID-19 vaccine type; and the sentiment analysis web application. The results are the single CNN model has the highest evaluation metrics average 76% recall, 76% precision, and 76% f1-score on the Sinovac dataset topic; the single CNN model has the highest evaluation metrics average with 80% recall, 81% precision, and 81% f1-score on the AstraZeneca dataset topic; the hybrid LSTM-CNN model has the highest evaluation metrics average with 74% recall, 75% precision, and 74% f1-score on the Pfizer dataset topic; and the hybrid LSTM-CNN model has the highest evaluation metrics average with 74% recall, 75% precision, and 74% f1-score on the Moderna dataset. Finally, the sentiment analysis has produced the sentiment classification with new dataset using the trained sentiment analysis models on the four dataset topics.*

Keywords: *Twitter, Sentiment Analysis, Data Augmentation, Hybrid CNN-LSTM, CNN, LSTM*

ACKNOWLEDMENT

Praise and gratitude from the researcher give to the Almighty God because of His blessing and grace, the researcher can complete a thesis entitled *"**COVID-19 Vaccine Type Sentiment Analysis with Data Augmentation and CNN-LSTM**"* well and on time. The purpose of the preparation and writing for this thesis is to fulfil one of the requirements to obtain a bachelor's degree in Software Engineering specialization study at Nanjing Xiaozhuang University and bachelor's degree in Computer Science at Kalbis Institute.

With all limitations during preparation and finishing of this thesis, the researcher has received many guidance, supports, loves, and assistance from several persons. Therefore, the researcher would like to express their gratitude and appreciation to:

1. The researcher's family who has always provided loves, supports, encouragements, and prayers so the researcher can finish this thesis very well and on time.
2. The researcher's thesis supervisors both from Nanjing Xiaozhuang University and Kalbis Institute, Mr. Andy Song and Mr. Yulius Denny Prabowo S.T., M.T.I., who has provided a good direction, guidance, and inputs during the process of writing this thesis until completed.
3. Special thanks and gratitude to Mr. Yulius Denny Prabowo S.T., M.T.I. as head of Computer Science Major, Faculty of Computer Science and Design, Kalbis Institute, who has provided the experiment resources such as GPU server for the researcher so the experiment ran smoothly.
4. Mr. Li Qing as head of School of Information Engineering for international student in Nanjing Xiaozhuang University, who has provided a good guidance and knowledge to the researcher.
5. Ms. Sophie Mou and Ms. Wang Jing who has taken care my study as international student in Nanjing Xiaozhuang University for two years.

6. All lecturers both from Nanjing Xiaozhuang University and Kalbis Institute who have shared their knowledges to the researcher during the study.
7. Classmates and colleagues from IT'2017 Kalbis Institute and IT'2018 Nanjing Xiaozhuang University who have supported, encouraged, accompanied, and struggled together to complete the researcher's thesis and bachelor study.

The researcher realizes that this thesis still has limitations, therefore the researcher is open to suggestions and constructive criticism for more improvements. Finally, the researcher hope that this thesis can provide benefits and knowledge to all reader.

Jakarta, 03 May 2022

Mathias Yeremia Aryadi

L20253011 / 2017103126

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Background

The first case of COVID-19 was reported on Wuhan, People's Republic of China at the end of 2019 and then the disease has been found widely almost around the world, becoming a global wide pandemic. Fortunately, the first COVID-19 vaccine was created and distributed around the world at the end of 2020 after the struggle against the COVID-19 pandemic. The several nations have started to use the COVID-19 vaccine and encourage people to take the shoot. The COVID-19 vaccine topics become a hot discussion around the world, especially in digital platform such as social network; considering there are different types of COVID-19 vaccine: Sinovac, AstraZeneca, Pfizer, and Moderna.

The explosion information in digital platforms, there are plenty of people sharing their opinions or thoughts about COVID-19 vaccine types on social network from 2021 until now. People have been shared opinions about: their decision to take a shoot of certain COVID-19 vaccine type, because some of them resulting a serious side effect; their feelings after taking a shoot one of the COVID-19 vaccine type; and their doubts about COVID-19 vaccine type. One of the most used and most popular social networks is Twitter. Based on Statista Research Department statistics, from the top 20 most popular social networks as of October 2021 by number of active users, Twitter is on the rank 16 with 463 million of active users [1]. Twitter was launched in 2006 by Jack Dorsey. In 2021, Twitter already has 2 million active users [2]. This implies everyday Twitter produces huge information that contain people opinions or thoughts and feelings to express their emotions about something. The opinions and feelings may express positive sentiment, negative sentiment, or neutral sentiment.

Sentiment analysis, also known as opinion mining, is the field of study that analyses: people's opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards entities such as products, services, organizations, issues, events, topics, and their attributes. Sentiment analysis has three level of analysis: document

level; sentence level; and entity and aspect level. Document level analyses and classifies the whole opinion in the document whether is positive or negative; sentence level analyses and classifies each opinion through sentence determines whether the opinion sentence is positive, negative, or neutral; and entity and aspect level overcome the limitation of document level and sentence level where analyses and classifies the sentiments on entities or/and their aspects. Sentiment analysis and Twitter datasets has been used along over the time, because Twitter provides an Application Programming Interface (API) access for us to retrieve all the data based on our query. Sentiment analysis allow us to extract the expressed emotion from people opinions or thoughts such as positive, negative, and neutral [3]. Sentiment analysis allow us to summarize the sentiments of opinions from a huge information stream on the social network. Sentiment analysis has been already used together with artificial intelligence such as machine learning and deep learning.

There are several works on sentiment analysis with machine learning, mostly they have used Naïve Bayes. Sentiment analysis on Twitter about anti-LGBT campaign in Indonesia [4], YouTube movie trailer comments [5], and Twitter about COVID-19 vaccine in Indonesia [6]. These works proofed that Naïve Bayes as a machine learning model resulted an accuracy from 81% to 83% and performed better in sentiment analysis compared to another model [4].

However, the machine learning model has several limitations [7]: require a large amount of annotated data to increase the performance, domain-dependent (need to retrain the classifier model for different domain), and set-selection feature limitation (cannot capture phenomena like negation detection or representative feature). Therefore, deep learning model has the capability to learn bigger dataset, capture, and address the representative feature of the data. Several works that related to them are sentiment analysis using both CNN and LSTM on IMDB dataset [8], Word2Vec and LSTM on Indonesian hotel reviews [9], and LSTM on IMDB and Amazon dataset [10]. The deep learning implementations on sentiment analysis resulted an accuracy from 85% to 88% that is higher than Naïve Bayes as machine learning model.

Additionally, it is also possible to combine multiple deep learning model to achieve a better result and performance, for allowing the model to capture the feature data more profound instead of a single deep learning model. Sentiment analysis using Word2Vec-CNN-LSTM on movie reviews [11], multiple word embedding CNN-LSTM on Arabic sentiment analysis [12], Word2Vec-CNN-BiLSTM on Quora question reviews [13], and CNN-LSTM on airline quality reviews. The hybrid model on sentiment analysis resulted a high accuracy from 90% to 91% higher than single learning model.

I also found data augmentation and sentiment analysis can be implemented together based on several works. A document level multi-topic sentiment analysis on email data with Bi-LSTM and random word replacement data augmentation [15]; and sentiment analysis on Vietnamese language with word replacements, word deletion, word swapping, and word insertion data augmentation using classifier models [16].

From the several works mentioned above, the hybrid model performance has the highest accuracy among the baseline model (single machine learning model and deep learning model). However, the data augmentation has not yet implemented together for the sentiment analysis models. Therefore, I interested to build a sentiment analysis of each COVID-19 vaccine types by using data augmentation and CNN-LSTM, to extract what people's sentiment about it.

I built the sentiment analysis models that consisted of the hybrid CNN-LSTM, the single CNN model, and the single LSTM. In contrast, I implemented the data augmentation together with the sentiment analysis models to increase the dataset size. I used tweets about COVID-19 vaccine types that related to Sinovac vaccine, AstraZeneca vaccine, Pfizer vaccine, and Moderna vaccine for experiment dataset and implementation dataset.

I also did a comparative study the performance between the hybrid CNN-LSTM model, single CNN model, and single LSTM model with same parameters. The result of this work provides sentiment analysis model performance

comparisons between the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model; and sentiment analysis web application about COVID-19 vaccine types that are Sinovac, AstraZeneca, Pfizer, and Moderna.

1.2 Problems Definition

From the background above, I identified several problems that occurred in this work, as follows:

1. How to implement the data augmentation on the sentiment analysis models?
2. How is the performance difference between the hybrid CNN-LSTM model, single CNN model, and single LSTM model for sentiment analysis with data augmentation?
3. How to build a sentiment analysis web application about COVID-19 vaccine types?

1.3 Problem Limitations

This work has several limitations, described as follows:

1. This work built a sentiment analysis on sentence level
2. This work built a sentiment analysis for positive sentiment and negative sentiment
3. This work used supervised hybrid deep learning CNN-LSTM model, single CNN model, and single LSTM model for sentiment analysis as binary classification problem
4. This work used tweets with four topics about COVID-19 vaccine type such as Sinovac, AstraZeneca. Pfizer, and Moderna
5. This work used tweets of four topics about the COVID-19 vaccine type from 1 March 2022 until 21 March 2022 for the experiment data
6. This work used tweets of four topics about the COVID-19 vaccine type from 22 March 2022 until 31 March 2022 for the implementation data
7. This work used split 90% training data and 10% testing data on the four topics experiment data

8. This work used an automated tweet sentiments labeling with lexicon based for unlabeled crawled Twitter data
9. This work built the user interface for result of the sentiment analysis in web-based application
10. This work used incremental model to define the process of building the sentiment analysis model and building the sentiment analysis in web application
11. This work produced a comparison between the hybrid CNN-LSTM model performance, single CNN model, and single LSTM model with same model parameters on four topics sentiment analysis on the first increment
12. This work produced sentiment analysis web application on the second increment

1.4 Research Aims

Based on the defined problem, the aims of this research are described as follows:

1. Implement data augmentation on the sentiment analysis models.
2. Build sentiment analysis models such as the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model with data augmentation and compare the results for COVID-19 vaccine type topics that are Sinovac dataset topic, AstraZeneca dataset topic, Pfizer dataset topic, and Moderna dataset topic.
3. Build sentiment analysis web application for COVID-19 vaccine type topics that are Sinovac dataset topic, AstraZeneca dataset topic, Pfizer dataset topic, and Moderna dataset topic.

1.5 Benefits

The benefits of this work are expected to be useful for both theoretically and practically:

1.5.1 Theoretically

4. Science development contribution in artificial intelligence especially in natural language processing and software engineering field
5. Know and discover about how to build a COVID-19 vaccine type sentiment analysis with data augmentation and supervised hybrid CNN-LSTM model, single CNN model, and single LSTM model
6. Know and discover between the difference supervised model performance for sentiment analysis on different topics

1.5.2 Practically

For Researcher:

1. Improving practical skills in artificial intelligence and software engineering field, especially on building supervised CNN-LSTM model, single CNN model, and single LSTM model with data augmentation for sentiment analysis
2. Understand about how to implement data augmentation and supervised model for sentiment analysis
3. Understand about how to use natural language processing approach to pre-process the text data

For University:

Research material contribution for future work or research.

For Public:

Provide the sentiment of COVID-19 types summary from vary sentiment on the Twitter in web-based application.

1.6 Schedule

The schedule will be implemented on this work is as follows:

Table 1.1 Thesis Schedule of 9 weeks

Activities	**Weeks**								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Introduction writing									

Literature reviews writing		■							
Research methodology writing			■	■					
Implementation writing					■	■	■		
System Development					■	■	■		
Summary writing								■	■

1.6 Organizational Structure of the Paper

This work is consisted of structural systematic writing, divided into five chapters as follows:

CHAPTER 1: INTRODUCTION

The chapter describes about the background of the work, the problems definition, the problem limitations, the benefits, the schedule, and the organizational structure of the paper.

CHAPTER 2: THEORITICAL BACKGROUND

In this chapter describes about the related works and the related theory such as Twitter data, sentiment analysis, data augmentation, CNN (Convolutional Neural Network), LSTM (Long Short Term-Memory), Incremental model, black box testing. Python, and Flask.

CHAPTER 3: RESEARCH METHODOLOGY

In this chapter describes about the method that used in this work such as conceptual design, research flow and design, incremental I, and incremental II. The increment I focus on building the sentiment analysis model, whereas the incremental II focus on building the sentiment analysis user interface in web application.

CHAPTER 4: RESULT AND DISCUSSION

In this chapter elaborates the result from the first increment and the second increment. The first increment describes the dataset result, the built sentiment

analysis model results, the sentiment analysis model training results, and the sentiment analysis model testing and evaluation results. The second increment describes the sentiment analysis web application user interface results and the sentiment analysis web application black-box testing result.

CHAPTER 5: SUMMARY

In this chapter summarize the summary of this work and the suggestion for further work to improve the current state of this work.

CHAPTER 2
THEORITICAL BACKGROUND

2.1 Twitter Data

Generally, the Twitter data contains tweet, user or username, mention, retweet, and hashtag. The tweet is a single message posted on Twitter and contains maximum 140 characters from personal information or opinion on products or events to others such as links, news, photos, or videos. The user or username is a registered user on the platform and gained access to post tweets. The mention is a message in tweet to mention another user on a post and uses "@" symbol, followed by the specific username they refer to; for example, "@username". The retweet is the tweet that are shared and uses its abbreviation "RT", followed by the author's username; for example, "RT @username". The hashtag labels the relevance of a tweet to a certain topic and uses "#" symbol, followed by the topic name on post; for example, "#topic"; then the users use the hashtags to search and get all the tweets with search tag [14].

2.2 Twitter API

Twitter API has provided developers to have an access to their data programmatically. Twitter API has three developer accesses such as: essential, elevated, and academic research [15].

Table 2.1 Twitter API Product Access

Essential	***Elevated***	***Academic Research***
A free access	*A free access with additional endpoints access and App environment*	*A free access with more endpoints access and App environment*
Retrieve 500,000 tweets per month	*Retrieve 2,000,000 tweets per month*	*Retrieve 10,000,000 tweets per month*
1 project per account	*1 project per account*	*Access to full-archive search and full-archive Tweet counts*
1 App environment per project	*3 App environments per project*	*Access to advanced search operators*

No access to standard v1.1, premium v1.1, or enterprise	*Access to standard v1.1, premium v1.1, or enterprise*

In order to access and retrieve the Twitter data, the Twitter API has provided a search query. The search query matches the Twitter API endpoints with GET request and return a set of historical tweets. I used the elevated access that only carries 512 characters on the search query. The Table 2.2 describe several Twitter API search query operators.

Table 2.2 Twitter API Search Query Operators

Query	***Description***
Keyword term	*Every keyword on the search query will be tokenized.* *For example: "I like coca cola", will be split into following tokens: "I", "like", "coca", "cola"* *The keyword term is evaluated in a case-insensitive manner.* *For example: a search query term cat will match tweets that contain terms cat, Cat, or Cat.*
Hashtag	*A conditional search query operator to match any tweet that containing a recognized hashtag.* *For example: a search query #newyear will match any tweet that contain #newyear but not the #happynewyear*
AND logic	*A conditional search query operator to match if both conditions are true.* *For example: a search query @andy AND #happynewyear will match any tweet that contain username andy and happynewyear hashtag*

OR logic	*A conditional search query operator to match any tweet if one of the conditions are true.* *For example: a search query @andy OR #happynewyear will match any tweet that at least contain username andy or happynewyear hashtag*
NOT logic	*An operator to negate the search query by using "- " symbol.* *For example: a search query happy -#birthday will match any tweet that contain term happy but without the birthday hashtag*
Grouping	*A search query operator to group operators together with "()" parentheses symbol.* *For example: a search query (happy day) OR (new stuff) will match any tweet that contain either term happy and day or either term new and stuff*

2.3 Sentiment Analysis

Sentiment analysis or usually called as opinion mining, is the field of study of NLP (Natural Language processing) that analyse and extract people's opinion sentiments, evaluations, appraisals, attitudes, and emotions towards entities such as products, services, organizations, individuals, issues, events, topics, and their attributes [3]. The Figure 2.1 shows the sentiment analysis has three level applications that are document level, sentence level, and aspect level [7].

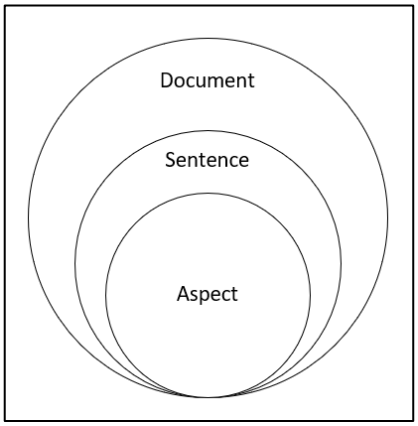


Figure 2.1 The Sentiment Analysis Level

2.3.1 Document Level Sentiment Analysis

The aim of the document level sentiment analysis is to summarize and classify whether a whole document expresses negative or positive sentiment. To do the document level sentiment analysis, I need to sum up the scores of each sentence based on weighting rules [7].

Table 2.3 The Document Level Sentiment Analysis Example

Document	***Sentiment***
The movie has great story and cinematic. But some of the casts who are not suitable in playing the characters. Fortunately, the CGI and the lighting of the movie is fantastic.	*Positive*
The movie has bad and confusing story. The cinematic element is very lacking. Only the casts that has good role playing most of the character in the movie. Unfortunately, the CGI and the lighting of the movie also bad.	*Negative*

2.3.2 Sentence Level Sentiment Analysis

The aim of the sentence level sentiment analysis is to summarize and classify whether a sentence expresses negative or positive sentiment. The difference between the document level and the sentence level is the document level focus on the whole document, whereas the sentence level focus on each sentence [7].

Table 2.4 The Sentence Level Sentiment Analysis Example

Sentence	***Sentiment***
The movie has good story and cinematic.	*Positive*
Fortunately, the CGI and the lighting of the movie is fantastic.	*Negative*
Unfortunately, the CGI and the lighting of the movie also bad.	*Negative*
The cinematic element is very lacking.	*Negative*
The movie has bad and confusing story.	*Negative*

2.3.3 Aspect Level Sentiment Analysis

In contrast of document level and sentence level sentiment analysis, the aspect level sentiment analysis has more precise sentiments or opinions of each entity that existed topic [7].

Table 2.5 The Aspect Level Sentiment Analysis Example

Topic	***Aspect***	***Sentiment***
The movie has great story and cinematic. But some of the casts who are not suitable in playing the characters. Fortunately, the CGI and the lighting of the movie is fantastic.	*Story*	*Positive*
	Cinematic	*Negative*
	Cast	*Negative*
	CGI	*Negative*
	Lighting	*Negative*

2.4 Data Augmentation

There are several regularization techniques such as dropout, batch normalization, transfer learning, pre-training, one-shot, and zero-shot learning. These regularization techniques mostly implemented on deep learning model with small dataset [16]. In contrast, data augmentation is also one of the regularization techniques to reduce overfitting when it comes deep learning training by fixing the dataset problem (i.e., small training sample). Data augmentation is a set of algorithms that build synthetic data from an original dataset. Data augmentation tweaks or changes the original dataset with different variant and generated them as additional dataset without changing the original one. Data augmentation can help deep learning model to avoid learning spurious data correlations and memorizing high-frequency patterns. Data augmentation has been implemented on image tasks

with deep learning model. In NLP (Natural Language Processing) tasks, data augmentation shuffles the form of language. There are two data augmentation techniques on NLP, as follows [17]:

2.4.1 Symbolic Data Augmentation

Symbolic data augmentation uses rule-based, graph-structured decomposition, Mix-up Augmentation, and feature space augmentation to generate new examples of dataset. The rule-based consist of: Easy Data Augmentation (i.e., random swap, random insertion, random deletion, random synonym replacement), regular expression augmentation, and syntactic heuristic (e.g., inverse the subject and the object of the sentence, change active form to passive form and vice versa). The graph-structured decomposition consists of: knowledge graph, WordNet, and syntactic parsing. Mix-up generate new example by mashing up the existing examples by taking a half of one text sequence and concatenate with another sequence. Feature space augmentation manipulates the features at deep neural network space by isolating them and applying noise to generate new examples.

2.4.2 Neural Data Augmentation

Neural data augmentation uses auxiliary neural network to generate new training examples. The neural data augmentation consists of several methods such as back-translation augmentation, style augmentation, and generative augmentation. The back-translation augmentation uses machine translation model to translates text from one language to another and then back from the translation to the original language, that this may lead to the difference structure of the back-translated text. The style augmentation changes the author writing style on the examples by extracting the semantic similarities between them. The generative augmentation uses transfer learning and pre-trained model to generate new training examples such as C-BERT (Conditional-BERT), GPT-3, RAG (Retrieval Augmented Generation), and REALM (Retrieval Augmented Language Model).

2.5 CNN (Convolutional Neural Network)

CNN is one of the deep learning algorithms that learn small regions of the scene rather than the whole scene (i.e., extract the local correlated feature that

available in the input). This implies, the CNN can simplify and speed up the training process of the network. The CNN has three key benefits makes difference from FC (Fully Connected Network): equivalent representations, sparse interactions, and parameter sharing. I can see the Figure 2.2 shows the CNN model architecture several layers that are input layer, convolutional layer (followed by the activation layer), pooling layer, and FC (Fully Connected) layer [18].

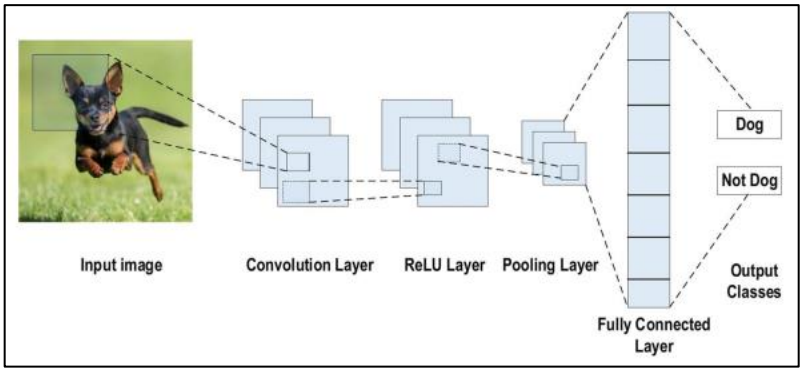


Figure 2.2 The CNN Model Architecture Example [18]

2.5.1 Input Layer

Mainly, the input layer of the CNN organizes the input data x with three dimensions: heigh, width, and depth, I can denote them as $w \times h \times r$, where the width (w) is equal to the height (h) and the r is the depth of the data such as in RGB (Red Green Blue) image has three depths (r) [18].

2.5.2 Convolutional Layer

The convolutional layer is the core of the CNN architecture as its name. The convolutional layer uses convolution filter to convolve through the input data. The convolution filter contains weights that initialized before the training process. The output of this layer is a feature map. The convolutional layer has several hyperparameter such as [18]:

2.5.2.1 Convolution Filter

The filter defines the convolution filter with size of the filter as the depth and size of the kernel as the two dimensional. I can denote the convolution filter size as $k \times k \times f$, where the kernel width (k) is equal to the kernel height (k), however the k value is smaller than the width (w) and the height (h) of the input

data, while the filter size of f is either equal or smaller than the depth (r). The kernel size and the filter size determine the total of the convolution filter weight that will be initialized, for example, the convolution filter size of 3x3x1 means there are 3x3 kernel size, 1 filter size, and total 9 weight values on the convolutional filter. Finally, the output feature map is calculated with one of the activation functions such as sigmoid, tanh, ReLU (Reactified Linear Unit), Leaky ReLU, noisy ReLU, and parametric linear units.

2.5.2.2 Stride

The stride defines how many the vertical or the horizontal steps needed on the convolution process, denoted as s. The bigger stride value (s), the size of the output feature map will have lower dimension.

2.5.2.3 Padding

The padding defines the border size of the input data. I can denote the padding size as $p \times p$, where the padding width (p) is equal to the padding height (p). The bigger padding size (p), the size of the input data and the output feature map will increase.

2.5.3 Pooling Layer

The pooling layer is the next step after I convolve the input data with the filter. The pooling layer perform a sub-sampling by shrinking the output feature data size from the convolutional layer to create a smaller feature map. Several pooling methods such as: tree pooling, gated pooling, average pooling, min pooling, max pooling, GAP (Global Average Pooling), and GMP (Global Max Pooling). I can denote the pooling size as $P \times P$, where the pooling width (P) is equal to the pooling height (P). The bigger pooling size, the sensitivity of capturing certain feature will higher.

2.5.4 Fully Connected (FC) Layer

The FC layer contains neurons that connected each other from the previous and the next layer (i.e., deep neural networks). I can define the convolution and the pooling layer as the feature extraction and the FC layer as the classifier. Then, the

output layer of the FC layer to predict the output from the CNN model by performing deep neural networks process: feed-forward that is multiply the flattened feature map with the neuron weights, followed by activation function; and backpropagation, calculate the error (the distance with the predicted output and the expected output) and minimize the loss function with optimization algorithm.

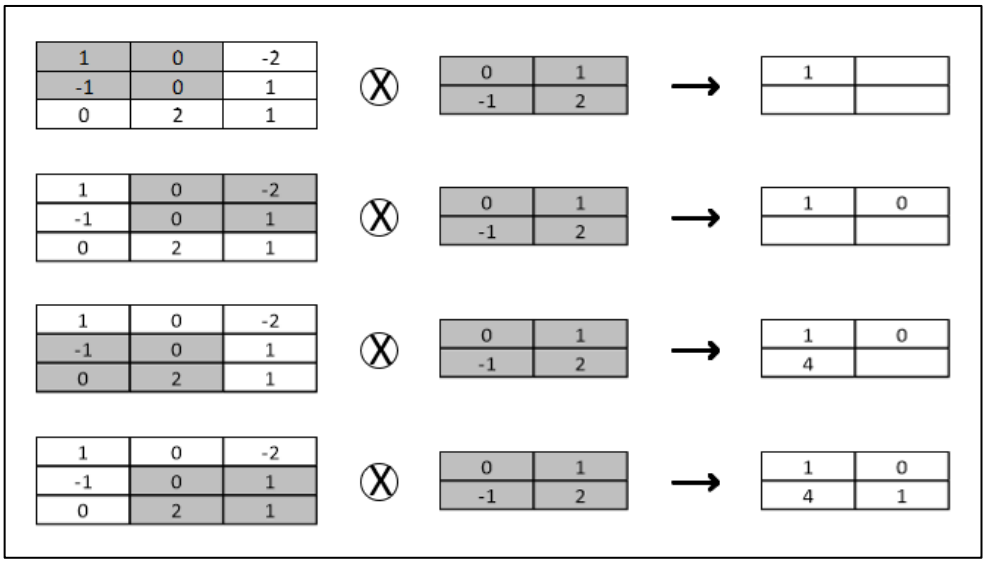


Figure 2.3 Convolution Filter of The Convolutional Layer Calculation Example

The Figure 2.3 shows how the filter process in the convolutional layer on CNN model on 3x3 input data convolved with 2x2 kernel size, filter size of 1 (the convolution filter size will be 2x2x1 with total 4 initialized weights), one stride, and zero padding. The convolution filter slides over the input data to capture the spatial feature as the output feature map by performing a dot product process. The equation 2.1 define the dot product calculation in the filter of convolutional layer.

$$O_{ij} = \sum_{i=1}^{n} X_i F_i \qquad (2.1)$$

The formula above, the dot product is the sum of multiplication between one matrix with another matrix on same rows. The X_i is the input data matrix with row i, the F_i is the convolution filter matrix with row i, and the O_{ij} is the output feature map matrix with row i and column j. The Table 2.5 shows the dot product

of convolutional filter calculation based on the Figure 2.3 with each filter slides calculation.

Table 2.6 The Dot Product of Convolution Filter Calculation Example

Number of Filter Slides	***Dot Product Calculation*** $(\sum_{i=1}^{n} X_i F_i)$	***Output*** (O_{ij})
1	$(1 \times 0) + (0 \times 1) + (-1 \times -1) + (0 \times 2)$ $0 + 0 + 1 + 0$	*1*
2	$(0 \times 0) + (-2 \times 1) + (0 \times -1) + (1 \times 2)$ $0 - 2 + 0 + 2$	*0*
3	$(-1 \times 0) + (0 \times 1) + (0 \times -1) + (2 \times 2)$ $0 + 0 + 0 + 4$	*4*
4	$(0 \times 0) + (1 \times 1) + (2 \times -1) + (1 \times 2)$ $0 + 1 + -2 + 2)$	*1*

The convolved 3x3 input data with 2x2x1 convolution filter resulting in a 2x2x1 (one-dimensional kernel) output feature map (two-dimensional feature map), where the width is 2, the height is 2, and the filter is 1. The equation 2.2 and 2.3 define the convolution filter dimension calculation to achieve the size values [19]:

$$w' = \frac{w - f + s}{s} \tag{2.2}$$

$$h' = \frac{h - f + s}{s} \tag{2.3}$$

The formula above, is how the output feature map size is determined after the convolution process with zero padding. The w' is the output feature map width, the h' is the output feature map height, the w is the input data width, the h is the input data height, the f is the filter size of the convolution filter, and the s is the stride value. The equation 2.4 and 2.5 shows the additional notation p for the padding value in the output feature map size calculation.

$$w' = \frac{w - f + s + p}{s} \tag{2.4}$$

$$h' = \frac{h - f + s + p}{s} \tag{2.5}$$

If I follow the formula above, I can calculate the output feature map size by using the 3x3 input data, the 2x2 convolution filter, one stride, and zero padding. The calculation is performed as follows.

$$w' = \frac{3 - 2 + 1 + 0}{1} \tag{2.6}$$

$$w' = \frac{1 + 1 + 0}{1} \tag{2.7}$$

$$w' = \frac{2}{1} \tag{2.8}$$

$$w' = 2 \tag{2.9}$$

As well as the height for the output feature map,

$$h' = \frac{3 - 2 + 1 + 0}{1} \tag{2.10}$$

$$h' = \frac{1 + 1 + 0}{1} \tag{2.11}$$

$$h' = \frac{2}{1} \tag{2.12}$$

$$h' = 2 \tag{2.13}$$

Therefore, I got the size of the output feature map is 2x2x1.

Then, the pooling layer reduces the convolved output feature map size with the available pooling method that mentioned before. The pooling layer has similar process with the convolution layer, where it slides through the convolved output feature map horizontally and vertically with pooling method and the stride equal to the pooling size. The Figure 2.4 shows a brief example of 2x2 maximum pooling, 2x2 average pooling, GMP (Global Maximum Pooling), and GAP (Global Average Pooling). The maximum pooling finds the maximum value on each slide, the average pooling finds the average value on each slide, while the GMP finds the whole maximum value and the GAP finds the whole average value without sliding. However, both the GMP and the GAP perform the pooling without sliding through the convolved output feature map.

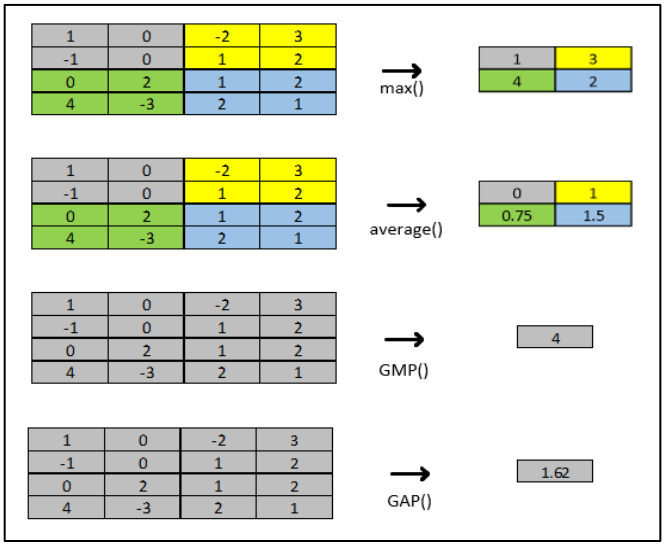


Figure 2.4 The Pooling Layer Process for Maximum Pooling, Average Pooling, GMP (Global Maximum Pooling), GVP (Global Average Pooling)

2.6 LSTM (Long Short-Term Memory)

LSTM (Long Short-Term Memory) or a sequential network is an improved of RNN (Recurrent Neural Network) handles the vanishing gradient problem that suffered from the RNN because the long-term dependency. In contrast, the LSTM has two main components on each neuron in the hidden layer that are memory cell and gates. Additionally, each transition from the previous neuron to the next neuron

is considered as timestamp. The memory cell has a cell state that tracks the information from previous timestamp of hidden layer neurons and updates whether the previous information is need to be remembered or to be forgotten for the next timestamp of memory cell. The gates in each LSTM neuron have several gates that are forget gate, input gate, and output gate [20].

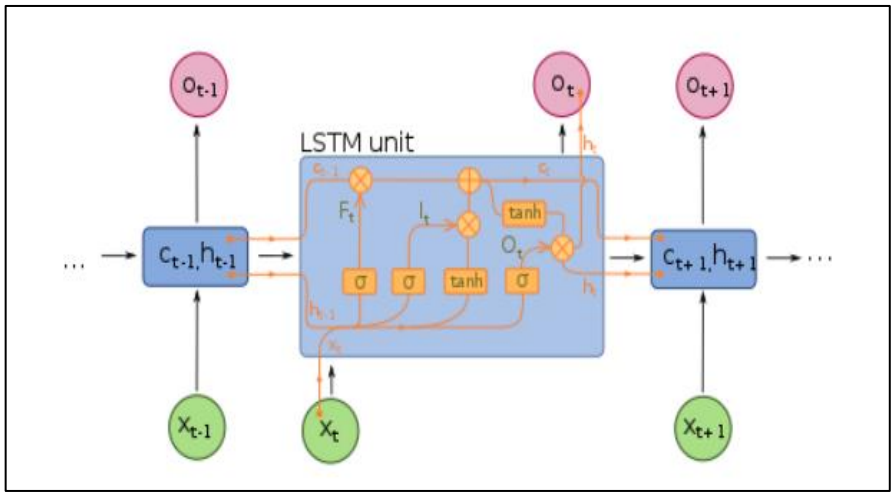


Figure 2.5 The LSTM Architecture [20]

The Figure 2.5 depicts the detail of the LSTM architecture with the notated memory cell and the gates. The basic notation of the Figure 2.5 consisted of t, $t-1$, C, and h, where the t is the current step of the timestamp, the $t-1$ is the previous step of the timestamp, the C is the cell state of the memory cell, and the h is the hidden state at the output gated

2.6.1 Forget Gate

The forget gate decides whether the neuron should keep or forget the information from the previous timestamp output. From the Figure 2.5, I can define the forget gate as follows:

$$F_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.14}$$

The equation above shows the calculation behind the forget gate, where the F_t is the forget gate output at the current timestamp t, the σ is the activation function that is sigmoid applied the after calculation inside is completed, the W_f is

the forget gate weight, the h_{t-1} is the previous hidden state output from previous timestamp, the x_t is the input data at the current timestamp t, and the b_f is the forget gate bias.

The whole process of the forget gate based on the equation, is begun with the multiplication between the forget gate weight W_f with the previous timestamp hidden state output h_{t-1} and the current timestamp input data x_t, then add them with the forget gate bias b_f. The sigmoid activation function applies the output of those calculation and resulting two numbers either 0 or 1 as the current timestamp forget output. Later, the current timestamp cell state will consider the current timestamp forget gate output whether the information should be kept or ignored, by multiplying the forget gate output with the previous timestamp cell state. I can define the calculation as follows:

$$C_t = F_t * C_{t-1} + \cdots \quad (2.15)$$

The equation above shows the calculation for updating the current timestamp cell state value and later another calculation will be added with input gate section. I can describe that the current timestamp cell state will forget the previous information if the value of the previous timestamp cell state C_{t-1} is 0, otherwise, the cell state will keep the previous information if the value of the previous timestamp cell state C_{t-1} is 1.

2.6.2 Input Gate

The input gate quantifies the importance of the new information carried by the input. From the Figure 2.5, I can define the input gate as follows:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2.16)$$

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C \quad (2.17)$$

The equation above shows the calculation behind the input gate to calculate the current timestamp input and the new candidate cell state. The cell state uses them to update the previous timestamp cell state. First, the current timestamp input equation defines the i_t is the input at the current timestamp t, the σ is the activation function that is sigmoid applied after the calculation inside is completed, the W_i is the input gate weight for the input quantification, the h_{t-1} is the previous hidden state output from previous timestamp, the x_t is the input data at the current timestamp t, and the b_i is the input gate bias for the input quantification. Second, the new candidate cell state equation defines the $\tilde{C}_t$ is the new candidate cell state at the current timestamp t, the $tanh$ is the activation function that is tanh that applies the after the calculation inside is completed, the W_C is the input gate weight for the new candidate cell state, the h_{t-1} is the previous hidden state output from previous timestamp, the x_t is the input data at the current timestamp t, and the b_C is the input gate bias for the new candidate cell state.

The whole process of the input gate based on the equation, is begun with the multiplication between the input gate weight (associated with either W_i and W_C) with the previous timestamp hidden state output h_{t-1} and the current timestamp input data x_t, then add them with the input gate bias (associated with either b_i and b_C). In contrast, the input gate calculation uses sigmoid activation function and the new candidate cell state calculation uses tanh activation function. Later, the current timestamp cell state will update the previous timestamp cell state value, by multiplying the output of the current timestamp input gate with the current timestamp new candidate cell state. Finally, I complete the current timestamp update equation as follows

$$C_t = F_t * C_{t-1} + i_t * \tilde{C}_t \qquad (2.18)$$

The equation above shows the complete calculation for updating the current timestamp cell state value by adding the multiplication in the forget gate output and the multiplication in the input gate output. The C_t is the new cell state output at the

current timestamp t. The new value of the C_t will be proceed to the next timestamp (the next neuron) and to the output gate.

2.6.3 Output Gate

The output gate is the last gate of the LSTM neurons to make the final output as the decision or prediction. From the Figure 2.5, the output gate has similar equation from the input gate as follows:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (2.19)$$

$$h_t = tanh(C_t) * o_t \quad (2.20)$$

The equation above shows the calculation behind the output gates to calculate the output and the new current timestamp hidden state. First, the output equation defines the o_t is the output at the current timestamp t, the σ is the activation function that is sigmoid applied after the calculation inside is completed, the W_o is the output gate weight for the output calculation, the h_{t-1} is the previous hidden state output from previous timestamp, the x_t is the input data at the current timestamp t, and the b_o is the output gate bias for the output calculation. Second, the new current timestamp hidden state equation defines the h_t is the new hidden state at the current timestamp t, the $tanh$ is the activation function that is tanh that applies the after the calculation inside is completed, the C_t is the new or updated current timestamp cell state, and the o_t is the calculated output at the current timestamp t.

The whole process of the output gate based on the equation, is begun by calculating the output o_t that similar to the input o_t at the input gate, then calculating the new current timestamp hidden state h_t multiplies the output o_t with applied activated function tanh of the updated current timestamp cell state C_t. In the end of the current timestamp, the new hidden state h_t will be also proceed to the next timestamp (the next neuron).

2.7 Related Works

Table 2.7 List of Related Works with Paper Title, Author, Method, and Result

No.	Paper	Author	Method	Result
1	Sentiment analysis of social media Twitter with case of Anti-LGBT campaign in Indonesia using Naïve Bayes, decision tree, and random forest algorithm [4]	Veny Amilia Fitri, Rachmadita Andreswari, Muhammad Hasibuan Azani	Naïve Bayes, Decision Tree, Random Forest	The Naïve Bayes model gained the accuracy of 83.43% higher than Decision Tree model and Random Forest model
2	Sentiment Analysis on Twitter Data by Using Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) [8]	Usha Devi Gandhi, Priyan Malarvizhi Kumar, Gokulnath Chandra Babu, Gayathri Karthick	CNN, LSTM	The CNN model gained 87.72% accuracy and the LSTM model gained 88.02% accuracy
3	Sentiment Analysis Using Word2vec and Long Short-Term Memory (LSTM) for Indonesian Hotel Reviews [9]	Putra Fissabil Muhammad, Retno Kusumaningrum, Adi Wibowo	LSTM	The LSTM model gained the best accuracy of 85.96%
4	A Hybrid CNN-LSTM Model for Improving Accuracy of Movie Reviews Sentiment Analysis [11]	Anwar Ur Rehman, Ahmad Kamran Malik, Basit Raza, Waqar Ali	CNN-LSTM	The CNN-LSTM model gained 91% higher than traditional machine learning and deep learning models
5	Sentiment analysis using word2vec-cnn-bilstm classification [13]	Wang Yue, Lei Li	Word2Vec, CNN-Bi-LSTM	The CNN-BiLSTM model gained high accuracy of 91.48% from other deep learning models (CNN, LSTM, Bi-LSTM, CNN-LSTM)
6	A Hybrid CNN-LSTM: A Deep Learning Approach for Consumer Sentiment Analysis Using Qualitative User-Generated Contents [21]	Praphula Kumar Jain, Vijayalakshmi Saravanan, Rajendra Pamula	CNN-LSTM	The CNN-LSTM model gained accuracy from 90% and 91% from both datasets, higher other than single machine learning models

7	*Document-level multi-topic sentiment classification of email data with Bi-LSTM and data augmentation [22]*	*Sisi Liu, Kyungmi Lee, Ickjai Lee*	*Data Augmentation, Bi-LSTM*	*The MultiTopic(MT)-Bi-LSTM model gained high accuracy from 78.8% to 91.8% (three datasets) and an improvement of accuracy from 1.5% to 10% with data augmentation*
8	*A data augmentation technique based on text for Vietnamese sentiment analysis [23]*	*Thien Ho Huong, Vinh Truong Hoang*	*Data Augmentation, Naïve Bayes. Random Forest, SVM*	*The Naïve Bayes, Random Forest, and Support Vector Machine models gained average accuracy of 84%, 86%, and 87% from the three datasets, then the average of accuracy improved by 10% on Random Forest model*

Fitri, Veny Amilia *et al.* [4], used several classifier models such as Naïve Bayes, Decision Tree, and Random Forest for sentiment analysis on Anti-LGBT campaign in Indonesia. They used RapidMiner to crawl 3744 comments on Twitter and pre-process with remove irrelevant tweets, case folding, tokenization, stop words removal, and stemming. The result is the Naïve Bayes model gained the accuracy of 83.43% higher than Decision Tree model and Random Forest model.

Gandhi, Usha Devi *et al.* [8], used CNN (Convolutional Neural Network) and LSTM (Long Short-Term Memory) for sentiment analysis on IMDB dataset. They used Python to crawl and process the 50,000 movie reviews; pre-process with data cleaning (e.g., removing unwanted URL, tags, links, spaces, brackets), case folding, tokenization, stop words removal; and extract the future with Word2Vec. The CNN model has input layer, Word2Vec (embedding layer), 1D convolution layer (128 features, 5x6 kernel size, ReLU activation function), Global Max Pooling 1D layer with ReLU activation, Dropout layer with 0.2 rate, and output

layer with Softmax activation. Besides that, the LSTM model has input layer, embedding layer, LSTM layer with 128 features size, and output layer with sigmoid activation function. The results are the CNN model gained 87.72% accuracy and the LSTM model gained 88.02% accuracy.

Muhammad, Putra Fissabil et al. [9], used LSTM (Long Short-Term Memory) on Indonesia hotel reviews. They used Selenium and Scrappy Python's libraries to crawl and process 5,000 sentiment texts; pre-process with tokenization, stop words removal, stemming, padding comprises; and extract the future with Word2Vec. The result is the LSTM model gained the best accuracy of 85.96% with the best Word2Vec parameters: Skip-Gram, Hierarchical Softmax evaluation, and vector dimension of 300; and the best LSTM model parameters: 0.2 dropout value, 0.001 learning rate, and average pooling.

Rehman, Anwar Ur *et al.* [11], used CNN-LSTM model for sentiment analysis on IMDB and Amazon movie reviews. They used Python to crawl and process the 40,000 movie reviews on IMDB dataset and 2000 movie reviews on Amazon dataset; pre-process with sentence segmentation, space removal, tokenization, stop words removal, duplicate words removal, stemming; and extract the feature with Word2Vec. The CNN-LSTM model has several layers: Word2Vec (embedding layer with 300 vector size), three convolutional layers (with 256 filter units, ReLU activation function), three global max pooling layers (ReLU activation function), one dense layer (ReLU activation function, 0.2 dropout rate), LSTM layer (256 filter units, 0.2 dropout rate), and output layer (sigmoid activation function). The result is the CNN-LSTM model gained 91% higher than traditional machine learning and deep learning models with 0,001 learning rate, 20 steps size, SGD (Stochastic Gradient Descent) optimizer, and 64 batch size.

Yue, Wang and Li, Lei [13], used CNN-BiLSTM model for sentiment analysis on Quora dataset. They used 2006 data on Quora, pre-processed stop words removal, low-frequency words removal, and extracted the feature with Word2Vec. The CNN-BiLSTM model has several layers: Word2Vec (embedding layer), convolutional layer (with 2x2 kernel size, 3x3 kernel size, 4x4 kernel size, 5x5

kernel size, ReLU activation function), max pooling layer (2x2 pooling size), Bi-LSTM layer, dropout layer, dense layer, and output layer with softmax activation function. The result is the CNN-BiLSTM model gained high accuracy of 91.48% from other deep learning models (CNN, LSTM, BiLSTM, CNN-LSTM) with 20 epochs, 64 batch size, and ADAM optimization.

Jain, Praphula Kumar *et al.* [21], used CNN-LSTM model for sentiment analysis on airline quality reviews. They used Python to crawl and process the 82,842 reviews on airline quality sentiment data and the 14,640 tweets on Twitter airline data; pre-process with remove unwanted words, symbols, and unknown words, tokenization, padding; and extract the feature with Keras Embedding layer. The CNN-LSTM model has several layers: embedding layer, one convolution layer (4x4 kernel size, 0.3 dropout rate, and ReLU activation function), one 1D max pooling layer (ReLU activation function), dropout layer (0.3 rate), LSTM layer (0.3 dropout rate, ReLU activation function, batch normalization), one dense layer (10 neurons, ReLU activation), and output layer (1 neurons, sigmoid activation function). The result is the CNN-LSTM model gained accuracy from 90% and 91% from both datasets, higher other than single machine learning models (Linear Regression, Support Vector Machine, Naïve Bayes, CNN, and LSTM).

Liu, Sisi *et al.* [22], used Bi-LSTM model and data augmentation for sentiment analysis on multi-topic email document. They used Python to crawl and process the 1,815 texts from three different topic of documents; pre-process with email cleaning, lowercasing, tokenization, spell-checking, stop words removal, and lemmatization; and extract the feature with word embedding. They also used the random word replacement data augmentation, by generating new text with some words replaced with their synonym or similar words. The result is the MultiTopic(MT)-BiLSTM model gained high accuracy from 78.8% to 91.8% (three datasets) and an improvement of accuracy from 1.5% to 10% with data augmentation.

Ho Huong, Thien et al. [23], used several machine learning models (Naïve Bayes, Support Vector Machine, Random Forest) and data augmentation for

sentiment analysis on Vietnamese language. They used Python to crawl and process 56,073 texts from three Vietnamese datasets; pre-process with tokenization, URLs removal, hashtag removal, email removal, symbols removal, emoticons handling, numbers removal, lowercasing, duplicated letters removal, punctuations removal, stop words removal, negation handling; and extract the future with TF-IDF (Term Frequence-Inverse Document Frequency). The result is the Naïve Bayes, Random Forest, and Support Vector Machine models gained average accuracy of 84%, 86%, and 87% from the three datasets, then the average of accuracy improved by 10% on Random Forest model.

This work, I used the hybrid model CNN-LSTM and compared the single model CNN and the single model LSTM for sentiment analysis like [11], [13], and [21]. In contrast, I compared those models with data augmentation from [22] and [23]; I used EDA (Easy Data Augmentation) for the data augmentation. In addition, I used different dataset for four sentiment analysis topics about COVID-19 vaccine types and I also built the sentiment analysis web application on the four topics about COVID-19 vaccine type to visualize the sentiment analysis summary; for example, tweets about Sinovac vaccine topic will have its own sentiment analysis, tweets about AstraZeneca vaccine topic will also have its own sentiment analysis, and so on. I used the pre-trained Glove Twitter for the word embedding with 200 vector dimensions. For the text pre-processing, I used most methods from all mentioned works above, however, I added another text pre-processing methods such as remove non-ASCII (American Standard Code for Information Interchange) character, contraction expansion, and negation handling.

2.8 Incremental Model

The incremental model is one of SDLC (Software Development Life Cycle) model uses the combination of linear model elements and prototyping iterative. The incremental model builds the iterative prototyping and each iterative or increment produces an actual product. Each iterative or increment also has its own progress phase such as design phase, implementation phase, and testing phase [24].

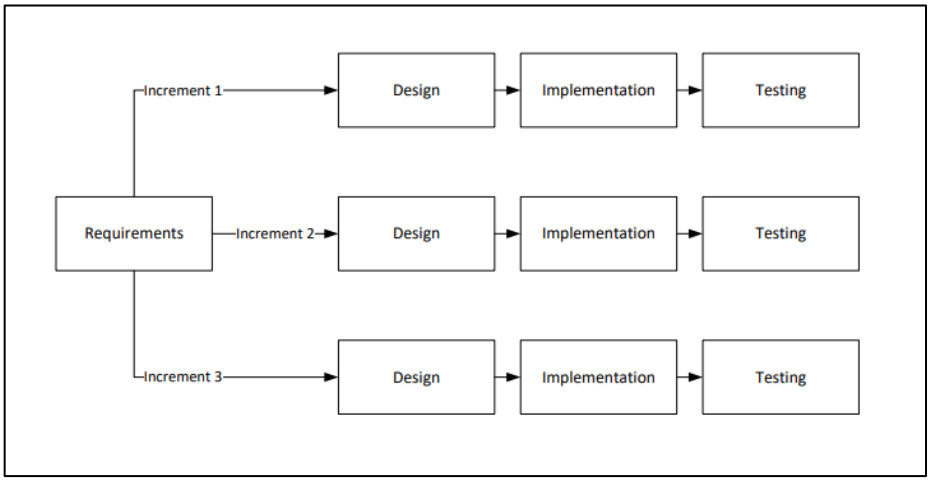


Figure 2.6 The Black Box Process Diagram

2.9 Confusion Matrix

Confusion matrix is a baseline method to measure and summarize the performance of the machine learning on classification problem. The confusion matrix depicts between the predicted values and the actual or true values [25].

Table 2.8 Binary Classification Confusion Matrix Table Example

Predicted	***Actual***	
	Negative	***Positive***
Negative	*TN (True Negative)*	*FP (False Positive)*
Positive	*FN (False Negative)*	*TP (True Positive)*

The Table 2.8 presents the confusion matrix measurement table on binary classification that consisted of TP (True Positive), TN (True Negative), FP (False Negative), and FN (False Positive). The TP means the actual values are positive and the model predicted as positive. The TN means the values are negative and the model predicted as negative. The FP means the actual values are positive and the model predicted as negative, it is known as type-I error. The FN means the actual values are negative and the model predicted as positive, it is known as type-II error [25]. Based on the confusion matrix summary, the model evaluation metrics such as accuracy, precision, recall, and f1-score can be calculated.

2.9.1 Accuracy

Accuracy is the number of correct predictions (both TP and TN) divided by the number of all samples (all values in the confusion matrix summed up). The accuracy calculation can be defined as follows [26]:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.21)$$

2.9.2 Precision

Precision is the number of samples are predicted as positive that are actually positive, it is known as PPV (Positive Predicted Value). The precision metrics is used to limit and concern the number of the false positive values. The precision calculation can be defined as follows [26]:

$$precision = \frac{TP}{TP + FP} \quad (2.22)$$

2.9.3 Recall

Recall is to measure all positive values is predicted correctly, it is known as sensitivity, hit rate, or TPR (True Positive Rate). The recall metrics is used to limit and concern the number of the false negative values. The recall calculation can be defined as follows [26]:

$$recall = \frac{TP}{TP + FN} \quad (2.23)$$

2.9.4 F1-Score

F1-score is the average overall between precision and recall. The f1-score metrics concerns and takes both precision and recall into account. The F1-score also better in measuring imbalanced binary classification datasets. The f1-score calculation can be defined as follows [26]:

$$f1 - score = 2 \times \frac{precision \times recall}{precision + recall} \quad (2.24)$$

2.10 Black Box Testing

Black box testing is to check and verify the software application from the user's perspective. The black box testing focuses on the input given to the software application and verify the produced output and the expected output. The scope of the black box testing is does not cover the details inside the system such as code, server logic, and development method. There are several black box testing techniques to design the test case of the system as follows [27]:

2.10.1 BVA (Boundary Value Analysis)

The BVA (Boundary Value Analysis) uses extreme values of test data on the test cases and identifies the errors that appear because to the limits of input data. For example, test case for age selection that only receive an input data between 1 – 100, the BVA test will try to test the system with extreme value data like -1, 1, 100, and 101 and check the system response using those values.

2.10.2 Equivalence Partitioning

The equivalence partitioning has similar test approach to the BVA, but in contrast the equivalence partitioning divides the input test case into equivalent classes. For example, taking the example that mentioned in BVA, the test case input data of the age selection between 1 – 100 will have three classes:

1. The valid class: input data between 1 – 100
2. The invalid class 1: input data value of -1, to check the lowest of the lowest value
3. The invalid class 2: input data value of 101, to check the highest of the highest value

2.10.3 State Transition Testing

The state transition testing covers the inputs, outputs, and state of the system during the test phase to check the system behaviour on each state, and its transition to another state with same input. For example, a login system with email and password input and a user is allowed to input until three attempts. Every time the user has invalid email or password, the system will redirect the user to the login

page. Finally, the system will redirect the user to the error page after the third attempts. The state transition testing will test each state and its transition on the login system until it passes with the right sequence.

2.10.4 Decision Table Testing

The decision table testing uses various probabilities approach on the test cases. The decision table checked and fulfilled each condition until pass the test with accurate output. For example, a food delivery system with several function and its conditional test case:

1. Case 1: if the user has the card for payment, then the system will not check for cash payment or coupon bonus and will take action to place the order.
2. Case 2: if the user has the coupon bonus for payment, then the system will not check for card or cash and action will be taken directly.
3. Case 3: if the user has the cash for payment, then the action will also be taken directly.
4. Case 4: if the user does not have anything for payment, then there is no action is taken.

2.10.5 Graph-Based Testing

The graph-based testing has similar testing process with the decision table testing, but in contrast it consider the relationship between the links and the input cases.

2.10.6 Error Guessing Testing

The error guessing testing guesses the output from the given input to fix any errors that might arise in the system. The tester compares and validate the results based on the given input cases and the outputs.

Table 2.9 Black Box Testing Table Example

Test Scenario	***Test Case***	***Expected Output***	***Test Result***	***Summary***
Add New Item	*Input all the item data and click the add button*	*The new item has added*	*Success*	*Normal*
Delete Item	*Choose the item and click the delete button*	*The item has deleted*	*Success*	*Normal*
Update Item Quantity	*Change the item quantity values and click the update button*	*The item quantity has updated*	*Success*	*Normal*

2.11 Python

Python is a strong, procedural, object-oriented, functional programming language created in the late 1980s by Guido Van Rossum. I can use Python in various application because each application has its own library. The application is GUI (Graphical User Interface) development, scripting web pages, database programming, prototyping, gaming, scientific development, web development, and component-based programming. Python provides several features that makes many developers use this programming language such as [28]:

2.11.1 Easy

Python is one of the easiest programming languages to understand, because unlike most of the programming language, Python does not have braces, parentheses, and semi-colon. For example, swapping variables in Python can be done in one row such as *(a,b) = (b,a)*.

2.11.2 Type and Run

Python provides an interactive environment for users to type and execute their code without execute the whole code when there are some changes happened.

2.11.3 Syntax

Python has simple, small, and flexible syntax that it makes easier to learn and understand. For example, to display something on the screen I can just type *print("Hello World")*.

2.11.4 Mixing

I can embed Python code into another programming languages, it helps with various programming language that used in a big project.

2.11.5 Dynamic Typing

Python supports dynamic memory management when an object is created and bring back the memory when the life cycle of the object is end.

2.11.6 Built in Object Types

Python has built in object types that helps users to accomplish task easier and faster.

2.11.7 Numerous Libraries and Tools

Python also supported with various libraries and tools on each application. For example, Django or Flask for web development in Python.

2.11.8 Portable

Execution in Python is very flexible, because it is supported on multiple operating system such as Windows, Linux, and Mac.

2.11.9 Free

Python is one of the open-source programming language, therefore users can download and install it with various available choices.

Table 2.10 Python Code Syntax Example [28]

No.	Python Syntax	Example Code
1	Variable	age = 10 average_age = 10.56 is_created = True full_name = “John Doe”
2	Operator	1 + 3 = 4 4 * 5 = 20 40 / 10 = 4 5 % 3 = 2 2 ** 2 = 4

3	Conditional Statement	if a == b:
		  print(“a is same as b”)
		
		if a < b:
		  print(“a smaller than b”)
		if a > b:
		  print(“a is bigger b”)
		
		if (a < b) and (b < c):
		  print(“b is betIen a and c”)

4	Looping	for number in range(5): print(number)
5	Function	def multiply_number(a,b): return a * b
6	Data Structure	student_list = [“A”, “B”, “C”] student_id_tuple = (1, 2, 3, 4) student_dictionary = {“name”: “John Doe”, “age”: 20}

2.12 Flask

Flask is a micro-framework for Python web development developed by Armin Ronacher. Flask has three main dependencies that are WSGI (web Server Gateway Interface) subsystems come from werkzeug, template engine supported by Jinja2, and CLI (Command Line Interface) integration comes from Click. Other

than mentioned above, Flask also supports standard feature like many web frameworks such as routing, debugging, database, web forms generation, REST API (Application Programming Interface), and email [29].

CHAPTER 3
RESEARCH METHODOLOGY

3.1 Research Framework

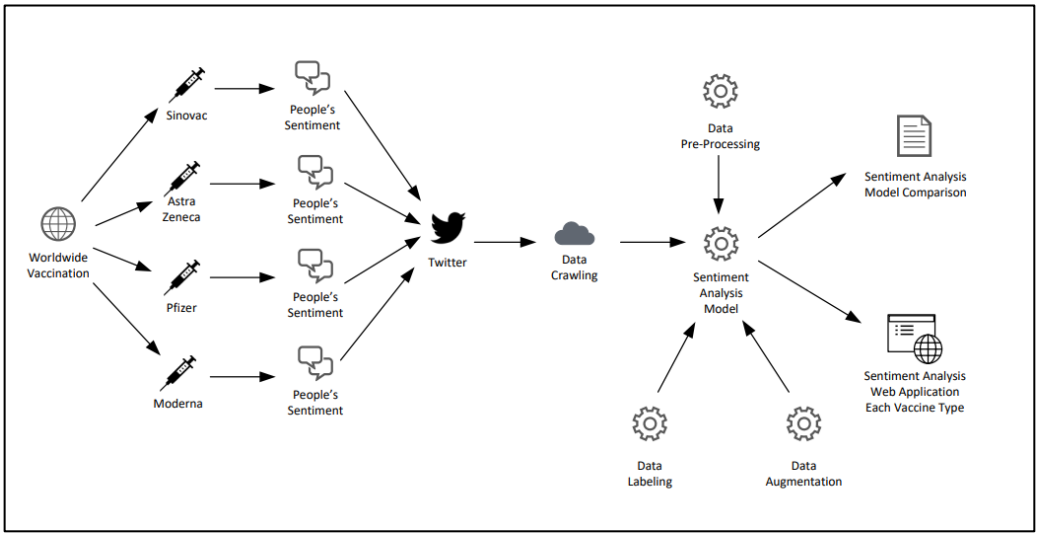


Figure 3.1 Research Framework

The Figure 3.1 shows bigger picture of research framework of the work to build the sentiment analysis for each COVID-19 vaccine type from the Twitter. All the people's sentiment about each COVID-19 vaccine type such as Sinovac, AstraZeneca, Pfizer, and Moderna on the Twitter will be crawled through Twitter API. The crawled each COVID-19 vaccine type tweets will be pre-processed, labeled, and augmented using Python programming language. The pre-processed and augmented data will be fit to the sentiment analysis models: the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model. Then, the output of the work resulted in sentiment analysis model performance comparison between three of them on the four topics and sentiment analysis web application for each COVID-19 vaccine type to present the sentiments results.

3.2 Research Process

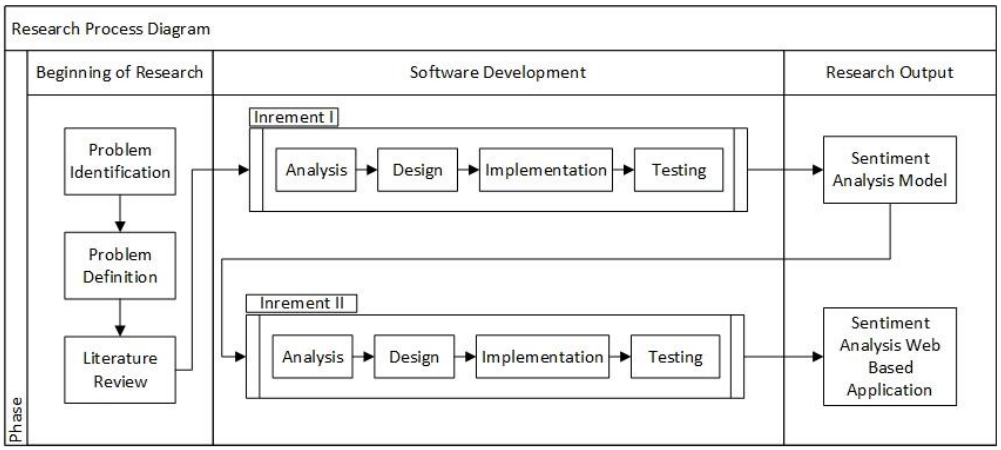


Figure 3.2 Research Process

The Figure 3.2 shows the research process that used to realize the research framework that defined before. The research process is consisted into three phases: the beginning of research phase, software development phase, and research output phase.

The beginning of research phase there are problem identification, problem definition, and literature review. The problem identification identifies where sentiment analysis for each COVID-19 vaccine type on the Twitter has not been used for sentiment analysis yet and sentiment analysis model comparison also has not been implemented with data augmentation. Therefore, the problem definition will be how the performance difference of the sentiment analysis model between the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model with data augmentation; using the COVID-19 vaccine type, the Twitter data. Then for the literature review, I gathered several information and related works from papers, books, and web references.

The software development phase, I used incremental model for the software development methodology. The incremental model of this work involved two increments. The first increment focus on building the sentiment analysis model and model analysis through model comparison. Furthermore, the second increment focus on building the sentiment analysis web application.

3.3 Software Development

The software development phase of this work used one of the SDLC (Software Development Life Cycle) method that is incremental model. The first increment focus on building the sentiment analysis model through data crawling, data merging, data filtering, data pre-processing, data labeling, data augmentation, and model building. The first increment involved the SDLC processes such as requirement analysis, design, implementation, and testing.

On the other hand, the second increment focus on building the sentiment analysis web application as the additional development from the first increment. The second increment started after the first increment is fulfilled. The second increment also involved the SDLC processes such as requirement analysis, design, implementation, and testing.

3.4 Incremental 1

In the first increment process, I defined the requirements needed to build the sentiment analysis model such as the dataset, the libraries, the pre-process method, the data augmentation method, and the model. Then, I designed the activity process design to build the sentiment analysis model and the sentiment analysis model architecture design. After that, I implemented the process according to the design through coding and testing the sentiment analysis model to compare the result. The output of the first increment is the sentiment analysis models and the performance comparison analysis for the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model.

3.4.1 Requirement Analysis

The requirement analysis of the first increment covered several aspects: the dataset, the pre-processing methods, the data augmentation method, the sentiment analysis models, the hardware, and the libraries.

The dataset that will be used is about each COVID-19 vaccine types tweets from Twitter by using Twitter API. Hence, there will be four dataset topics such as Sinovac dataset, AstraZeneca dataset, Pfizer dataset, and Moderna dataset; where

contains tweet texts that are related to them. The period of the dataset is taken from 1 March 2022 until 31 March 2022, because the limitation of the elevated access on Twitter API that only can retrieve tweet texts no more than seven days. Therefore, the dataset crawling is carried manually every week to crawl the four COVID-19 vaccine type tweets. The crawled dataset will be divided into two datasets, one will be used for building the sentiment analysis model and the other will be used for implementing the dataset to the sentiment analysis web application. The dataset that will be used for building the sentiment analysis model is the first three weeks of March 2022, which is, from 1 March 2022 until 21 March 2022 as the experiment dataset. On the other hand, the dataset that will be used for implementing the dataset to the sentiment analysis web application is the last one week of March 2022, which is, from 22 March 2022 until 31 March 2022 as the implementation dataset.

The pre-processing used several NLP (Natural Language Processing) approaches to handle text. The first part focuses on removing HTML (HyperText Markup Language) tags, removing retweets notation "RT", removing URLs, removing mentions that contain username (notated with "@"), removing hashtags (notated with "#") with its term, removing non-ASCII characters, changing the written numbers in words into actual numbers, removing numbers, and case folding. The second part focuses on expanding contractions, replacing negations, removing punctuations, removing stopwords, and lemmatizing. The third part focuses on removing duplicated tweets, removing empty values, and dividing into experiment dataset and implementation dataset.

The labeling process used lexicon-based method as automated labeling that resulting in positive sentiment, negative sentiment, and neutral sentiment; because of the huge datasets, limited time, and resources then it is expensive to label all the datasets manually. However, the positive sentiment and the negative sentiment will be used for the sentiment analysis. Afterwards, the pre-processed and labeled datasets will be split into two samples that are training data and testing data with 90% and 10% respectively.

The data augmentation will be applied only on the training samples for each four dataset topics. The data augmentation method used is EDA (Easy Data Augmentation). The EDA method consisted of synonym replacement, random insertion, random swap, and random deletion. The augmented text data with EDA will be merged to the original datasets, hence will be resulting in bigger training samples.

The sentiment analysis models that will be used is a supervised model that are the hybrid deep learning model CNN-LSTM, the single CNN model, and the single LSTM model. All of them are compared with each other respect to four dataset topics using same architecture and parameters. The augmented training data is fit to the sentiment analysis models. During the training process, the training data is split for the validation data with 10% portion. The pre-trained Glove Twitter word embedding weight will be used to map the pre-trained weight to my own corpus.

Table 3.1 The Hardware Requirement

No.	***Type***	***Name***
1	*Operating System*	*Linux Ubuntu*
2	*Processor*	*Intel i7 6859K 3.60 GHz 12 cores*
3	*Memory*	*62GB*
4	*GPU*	*NVIDIA RTX A6000* *NVIDIA GeForce 1080 Ti*

Table 3.2 The Software Requirement

No.	***Name***	***Type***	***Version***
1	*Jupyter Notebook*	*Integrated Development Environment (IDE)*	*-*
2	*Python*	*Programming Language*	*3.6.9*
3	*tweepy*	*Python Library*	*4.6.0*
4	*pandas*	*Python Library*	*1.1.5*
5	*matplotlib*	*Python Library*	*3.3.4*
6	*regex*	*Python Library*	*2022.1.8*
7	*string*	*Python Library*	*-*

8	*NLTK (Natural Language Tool Kit)*	*Python Library*	*3.6.7*
9	*unicodedata2*	*Python Library*	*-*
10	*BeautifulSoap*	*Python Library*	*4.11.1*
11	*pycontractions*	*Python Library*	*2.0.1*
12	*word2number*	*Python Library*	*1.1*
13	*vaderSetiment*	*Python Library*	*3.3.2*
14	*gensim*	*Python Library*	*4.1.2*
15	*textaugment*	*Python Library*	*1.3.4*
16	*scikit-learn*	*Python Library*	*0.24.2*
17	*tensorflow*	*Python Library*	*2.6.2*
18	*keras*	*Python Library*	*2.6.0*

The Table 3.1 defines all the requirement tools that will be used in this work. I used hosted *Jupyter Notebook* for the Python IDE, a web-based Python notebook, because of the limited resources of my personal computer to build the sentiment analysis model. The *Python* programming language will be used throughout the first increment to produce the sentiment analysis model. The *tweepy* library will be used for the Twitter data crawling. The *pandas* library will be used for reading, filtering, concatenating, and grouping the dataset as dataframe. The *matplotlib* library will be used for visualizing the dataset. The *regex* library will be used for performing regular expression on removing retweets, removing hashtags, removing mentions, and removing URLs. The *string* library will be used for removing punctuations on the tweet cleaning process. The *NLTK* library will be used for tokenizing texts, lemmatizing words, removing stopwords, and replacing negation with antonyms. The *unicodedata2* library will be used for removing non-ASCII characters. The *BeautifulSoap* library will be used for removing HTML tags. The *pycontractions* library will be used for expanding contractions. The *word2number* library will be used for replacing written numbers in words to actual numbers. The *vaderSentiment* library will be used for labeling the dataset based on its lexicon automatically. The *gensim* library will be used for building the pre-trained word embedding Word2Vec model. The *textaugment* library will be used for augmenting the text dataset with

EDA method. The *scikit-learn* library will be used for splitting the datasets and evaluating the models with confusion matrix. The *tensorflow* and *keras* library will be used for building the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model through training, validation, and testing process.

3.4.2 Design

The design of the first increment divided into two designs that are the activity process design and the sentiment analysis model architecture design. The activity process design is designed with flowchart, depicted all the processes in the first increment. The sentiment analysis model architecture design designed the sentiment analysis model with diagram about the layers, the neurons, the hyperparameters, and the training parameters.

3.4.2.1 Activity Process Design

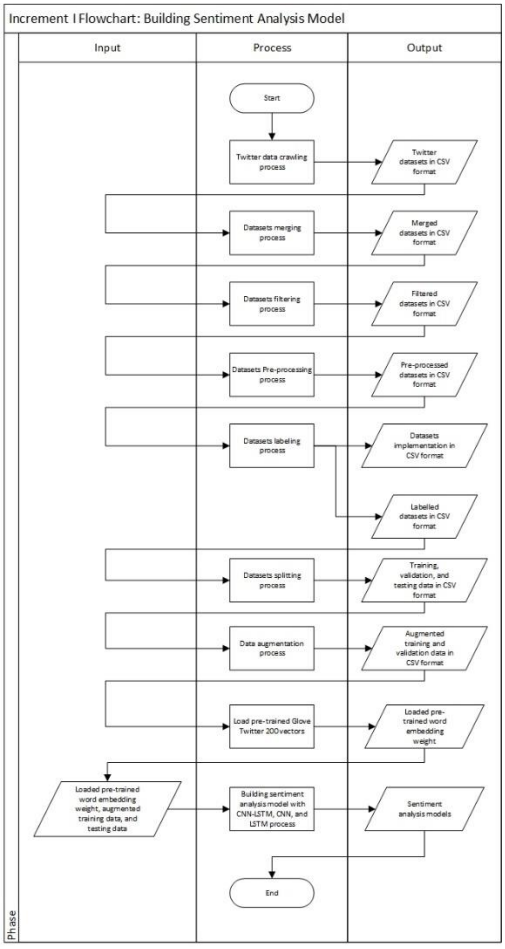


Figure 3.3 Building the Sentiment Analysis Model Flowchart

The Figure 3.3 depicts the flowchart of all the processes in the first increment to produce the sentiment analysis models based on the requirement. The main processes in the first increment are started with crawling the Twitter data into datasets; merging the crawled datasets; filtering the merged datasets; pre-processing the filtered datasets; labeling the pre-processed dataset; splitting the pre-processed dataset into training and testing data; augmenting the labeled training data; loading the pre-trained Glove Twitter with 200 vectors word embedding; and building the sentiment analysis model such as the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model using the loaded pre-trained Glove Twitter with 200 vectors word embedding weight, augmented training data, and testing data.

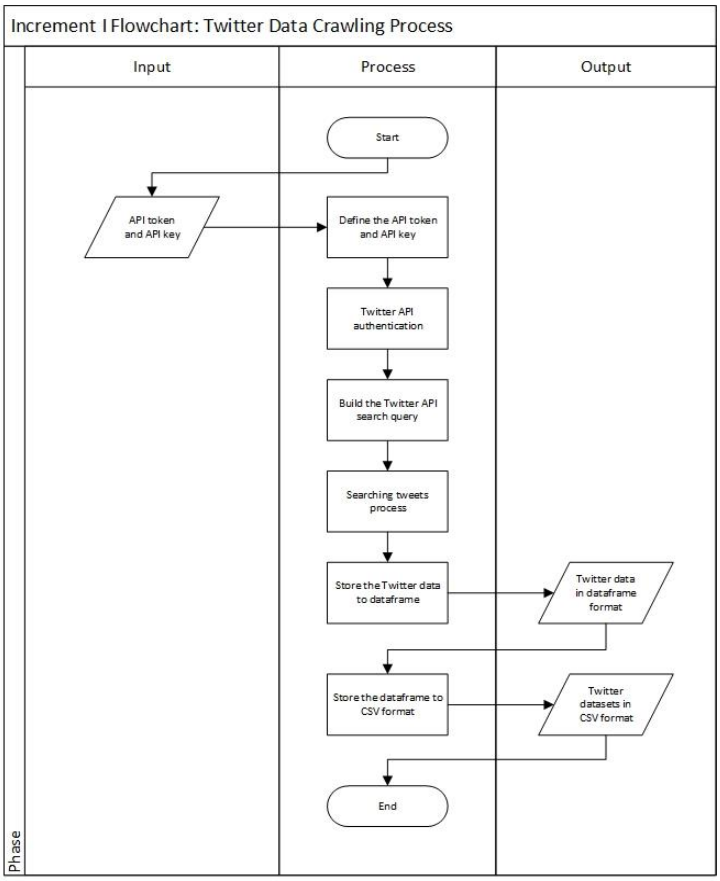


Figure 3.4 Twitter Data Crawling Process Flowchart

The Figure 3.4 depicts the flowchart of the Twitter data crawling process in detail. First, the Twitter data crawling process is begun with retrieving the API token and the API key to access the Twitter API. The API token and the API key are obtained through Twitter API developer access application. Second, the API token and the API key are defined in the Twitter data crawling program using Python. Third, the defined API token and API key are used for the Twitter API authentication using *tweepy* library for Twitter API. Fifth, built the Twitter API search query to search for tweets that are related to each COVID-19 vaccine types: Sinovac, AstraZeneca, Pfizer, and Moderna. The search query used keyword terms, hashtag, and negation operation. Sixth, the built search query is used for the searching process and stored them into dataframe using *pandas* library. The whole processes are done by weekly. Finally, the number of the Twitter data crawling result is 20 raw dataframes from each week that are Sinovac dataframe,

AstraZeneca dataframe, Pfizer dataframe, and Moderna dataframe. The stored 20 raw dataframes is saved into CSV format respectively so it can be re-used for further process.

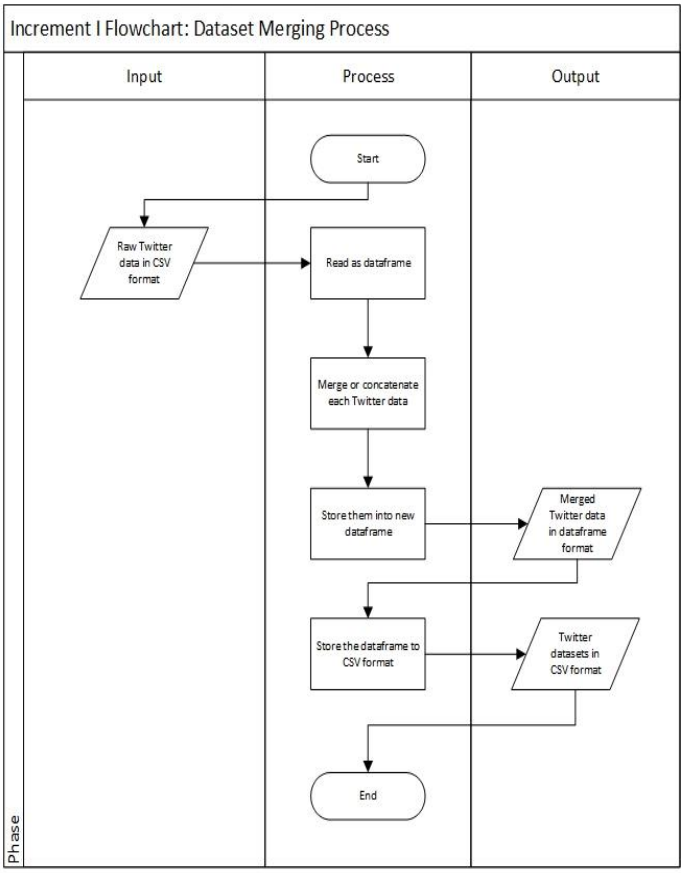


Figure 3.5 Dataset Merging Process Flowchart

The Figure 3.5 depicts the flowchart of the dataset merging process in detail. The dataset merging process merges the crawled Twitter data on each week into four new datasets respected to their topics. For example, the dataset about Sinovac with five datasets that are crawled for five weeks, are merged into one dataset. Therefore, the result of the dataset merging process produced four new dataset topics in CSV format: Sinovac dataset, AstraZeneca dataset, Pfizer dataset, and Moderna dataset; and so, it can be re-used for further process.

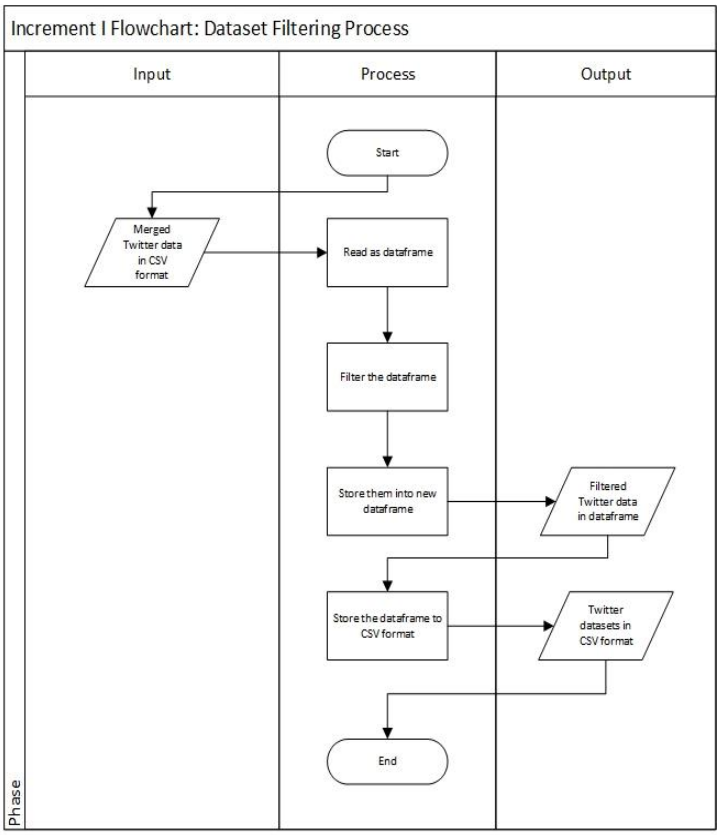


Figure 3.6 Dataset Filtering Process Flowchart

The Figure 3.6 depicts the flowchart of the dataset filtering process in detail. The dataset filtering process filters the four dataset topics by the date, in order to retrieve all datasets only in March 2022 period. Therefore, the result of the dataset filtering process produced four new dataset topics in CSV format with only March 2022 period. The four new datasets in CSV format will be re-used for further process.

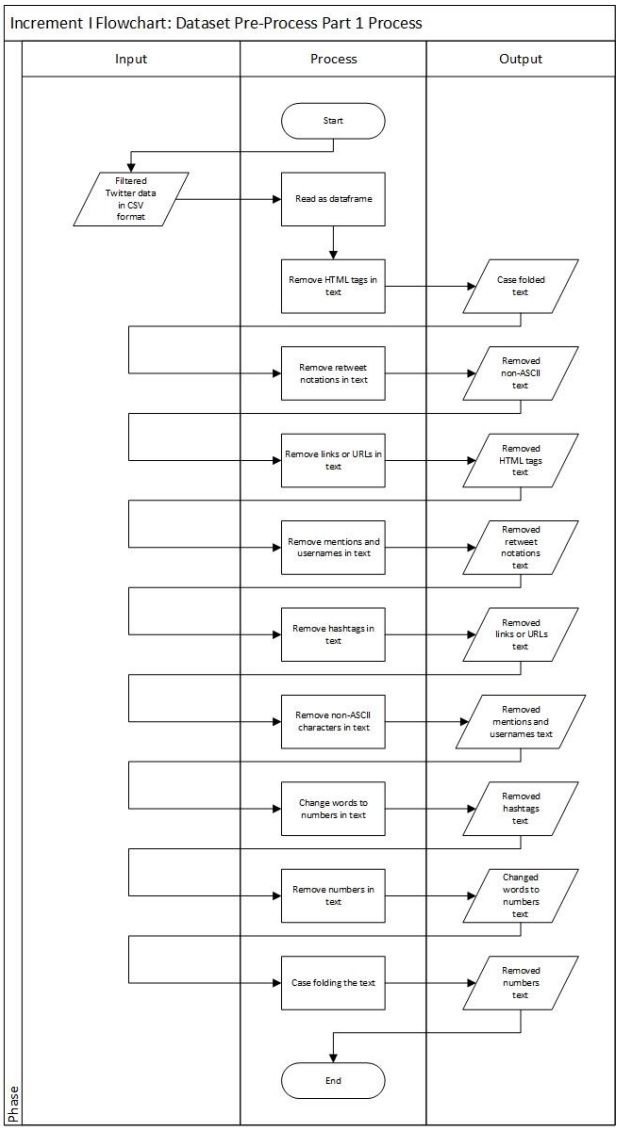


Figure 3.7 Dataset Pre-Processing Part 1 Process Flowchart

The Figure 3.7 depicts the design of the first part dataset pre-processing process in detail. First, the four filtered datasets in CSV format will be read as dataframe. Second, the four dataframes will be applied with removing all HTML tags in text using *BeautifulSoup* library. Third, the removed HTML tags will be applied with removing all retweet notations "RT" using *regex* library. Forth, the removed HTML tags will be applied with removing URLs in text using *regex* library. Fifth, the removed URLs will be applied with removing all mention notations "@"and its usernames in text using *regex* library. Sixth, the removed mentions and usernames will be applied with removing hashtags and its term in text

using *regex* library. Seventh, the removed hashtags and its terms will be applied with removing all non-ASCII characters in text using *unicodedata2* library. Eighth, the removed non-ASCII characters will be applied with changing all numbers that written in words in text into actual numbers using *word2number* library. Ninth, the changed words to numbers will be applied with removing all numbers in text. Finally, the removed numbers will be applied with case folding to lower case all the words in the text. The result of the first part dataset pre-processing is continued to the second part dataset pre-processing.

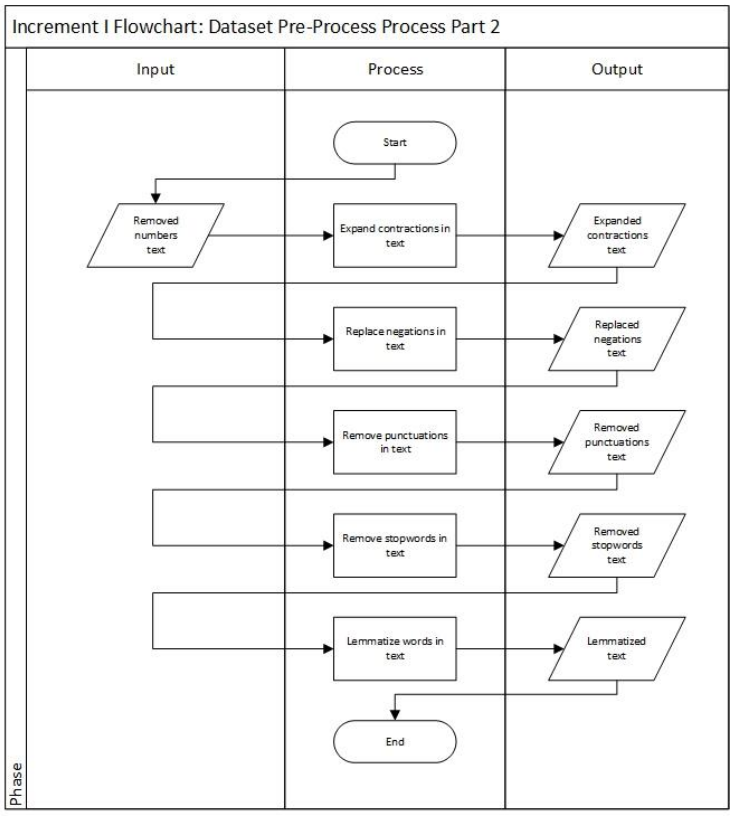


Figure 3.8 Dataset Pre-Processing Part 2 Process Flowchart

The Figure 3.8 depicts the flowchart of the second part dataset pre-processing process in detail that continued from the first part. First, the case-folded four pre-processed dataset topics from the first part will be applied with expanding all contractions in text using *pycontraction* library. Second, the expanded contraction text will be applied with replacing all negations with their antonym using *NLTK wordnet* library. Third, the replaced negation text will be applied with

removing punctuations using *string* library. Forth, the removed punctuations will be applied with removing all stopwords in text using *NLTK stopwords* library. Fifth, the removed stopwords text will be applied with word lemmatization using *NLTK WordNetLemmatizer* and *NLTK POST tag* library. Finally, the lemmatized text is finalized by removing several words that only has 2 characters (residual from the text pre-processing). The lemmatized text as the result of the second part dataset pre-processing is continued to the third part dataset pre-processing.

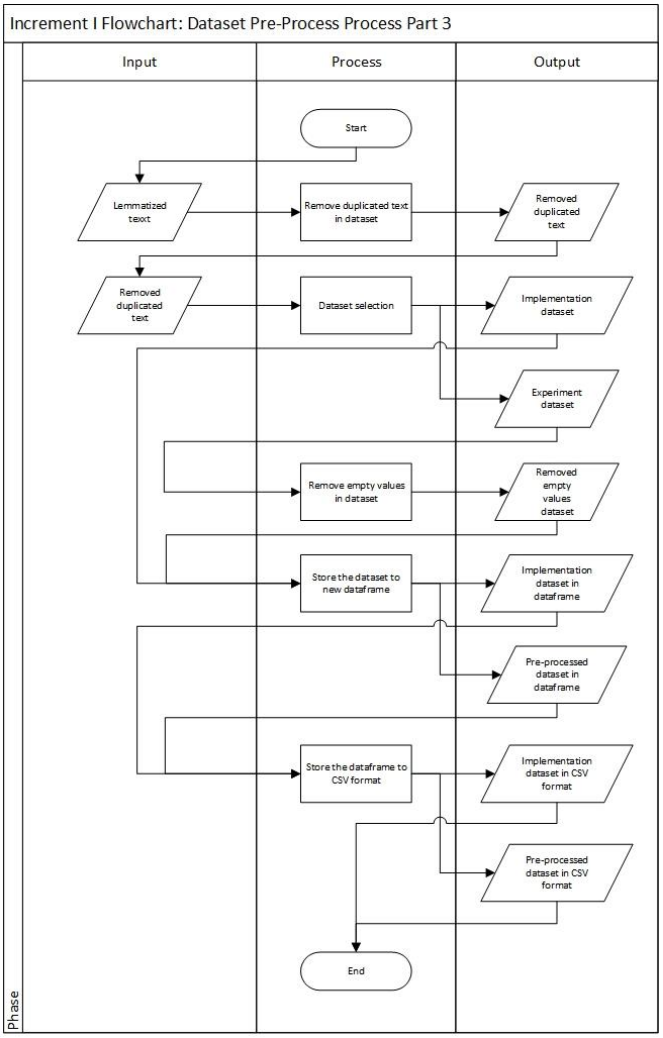


Figure 3.9 Dataset Pre-Processing Part 3 Process Flowchart

The Figure 3.9 depicts the flowchart of the third part dataset pre-processing process in detail. First, the lemmatized four dataset topics from the second will be applied with removing all duplicated text but the first occurrence is kept. Second,

the removed duplicated text will be applied with selecting which dataset will be used for experiment and implementation. The experiment dataset will be used for training and testing the sentiment analysis model. On the other hand, the implementation dataset will be used for the sentiment analysis web application. In contrast, the implementation dataset only used the raw or not pre-processed tweet texts. Second, the both experiment dataset and the implementation dataset will be applied with removing empty values based on the pre-processed text result, because some texts become empty after the pre-processing process. Finally, both the removed empty value experiment dataset and the implementation dataset will be stored in new dataframes and saved them into new datasets in CSV format for further process. The result of the dataset per-processing is four pre-processed experiment dataset topics are used to build the sentiment analysis model and four raw implementation dataset topics are used as in the sentiment analysis web application.

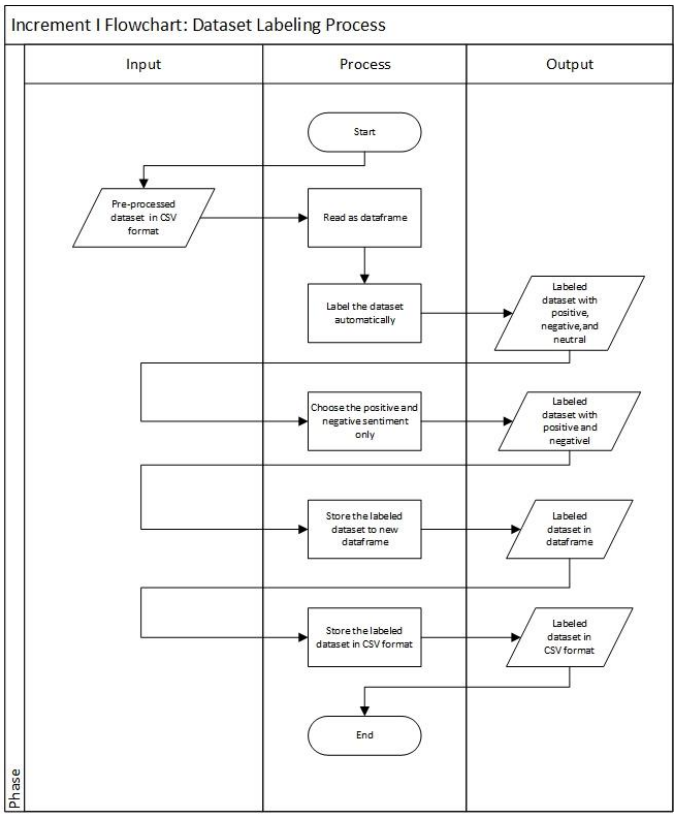


Figure 3.10 Dataset Labeling Process Flowchart

The Figure 3.10 depicts the flowchart of the dataset labeling process in detail. The *vaderSentiment* library will be used because it has performed better than other lexicon-based sentiment analysis libraries like *TextBlob* library on social media text [30] [31] [32]. The dataset labeling process labels each text from the four pre-processed experiment dataset topics into positive sentiment, negative sentiment, and neutral sentiment. However, the positive sentiment and the negative sentiment will be used and saved into four new dataset topics in CSV format. The four labeled dataset topics in CSV format will be re-used for further process.

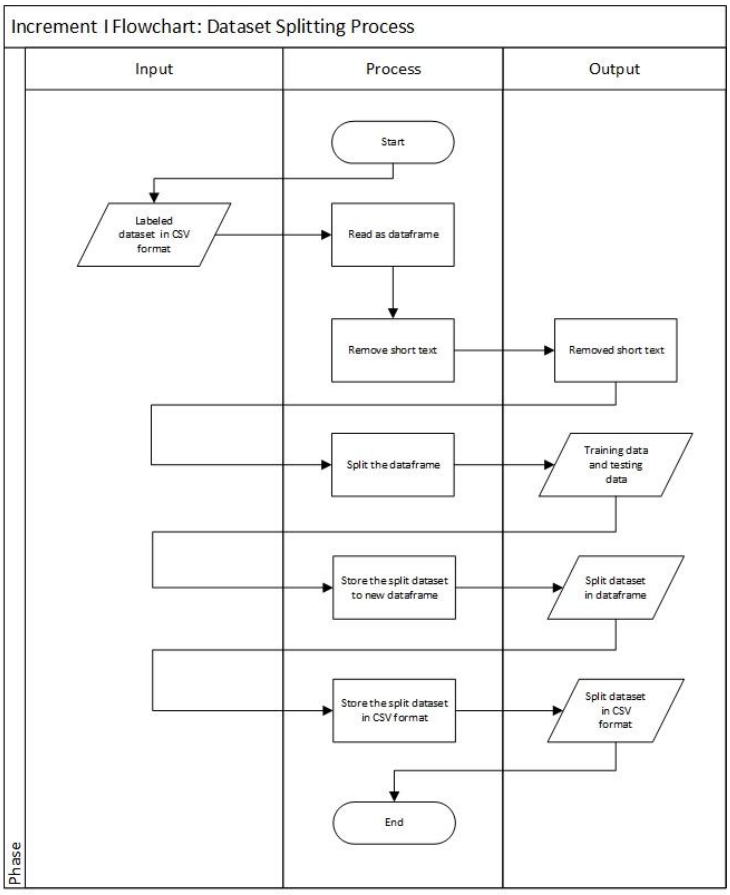


Figure 3.11 Dataset Splitting Process Flowchart

The Figure 3.11 depicts the flowchart of the dataset splitting process in detail. The dataset splitting process splits the four labeled dataset topics into training data and testing data. First, read the four labeled datasets in CSV format to dataframe. Before the dataset split, the number of words in the tweet texts that below five will be removed, because is considered as short text. Then, the four dataframe topics are split into training and testing data with 90% and 10% respectively. Finally, the result of the dataset splitting process produced eight new datasets in CSV format, where each of the four dataset topics has training data and testing data. The eight new datasets in CSV format will be re-used for further process.

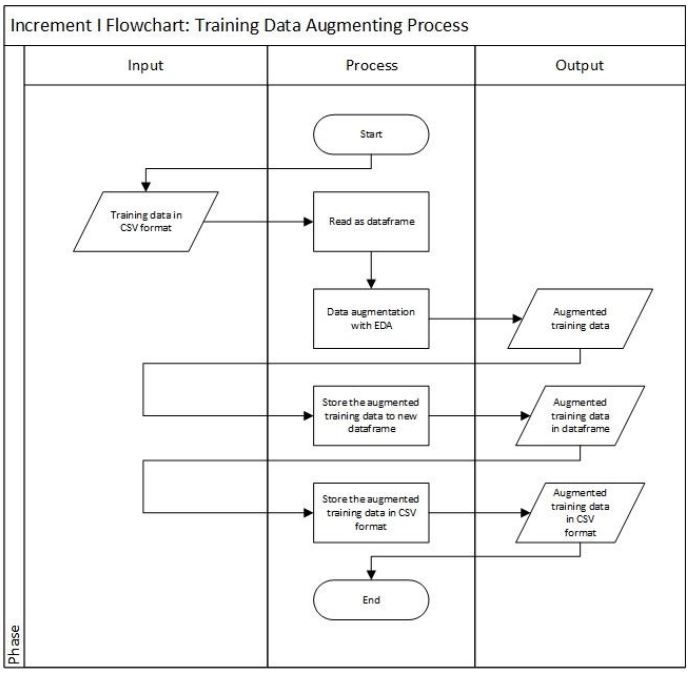


Figure 3.12 Training Data Augmentation Process Flowchart

The Figure 3.12 depicts the flowchart of the dataset augmentation process in detail. The dataset augmentation process is applied on the training data, where the sentiment analysis models learn from them. First, read the four training dataset topics in CSV format to dataframe. Then, they will be applied with EDA using *textaugment* library. The EDA method parameters is set as it is suggested from [33], like below:

1. **Synonym replacement**

   The number of words to be replaced randomly with its synonym is 10% from the total length of the text.

2. **Random Insertion**

   The number of words to be inserted randomly with other words is 10% from the total length of the text.

3. **Random Swap**

   The number of swap times to swap randomly between two words is 10% from the total length of the text.

4. **Random Deletion**

   The probability to delete words randomly is 10% from the total length of the text.

Then, the result of the dataset augmentation process produced four new datasets in CSV format, that the four dataset topics contain augmented training data. The four new datasets in CSV format will be re-used for further process.

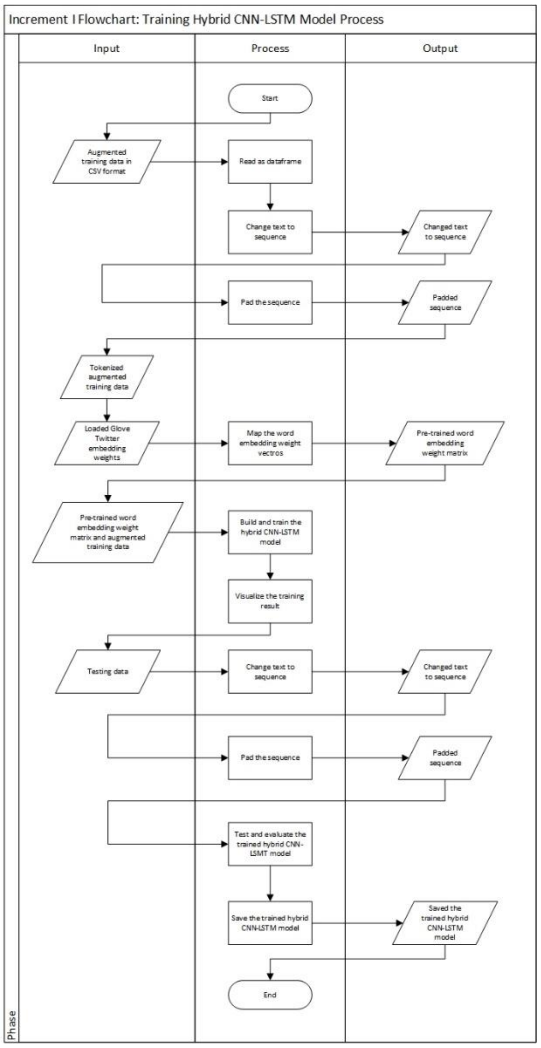


Figure 3.13 Building Sentiment Analysis Models: Training The Hybrid CNN-LSTM Model Process Flowchart

The Figure 3.13 depicts the flowchart of the hybrid CNN-LSTM model training process in detail. The four augmented training and testing dataset topics are used to train, test, and evaluate the hybrid CNN-LSTM model. The *keras* library is

used to build, train, test, and evaluate the hybrid CNN-LSTM model. First, the four augmented training dataset topics is read as dataframe. Second, the four augmented training dataset topics are tokenized and changed into sequence of positive integers respect to its word uniquely and the four training data label topics are changed into one-hot encoding. Third, the augmented training data that changed into sequence of positive integers will be applied with sequence padding to equalize the sequence length, because the input for the sentiment analysis models require same sequence length (the sequence length can be varied). Forth, the pre-trained Glove Twitter 200 vectors is loaded and mapped each word in the pre-trained word embedding weight vectors correspond to each word in the tokenized training data respectively. Hence, the process resulted in mapped four pre-trained word embedding weight matrix with own corpus. Fifth, the hybrid CNN-LSTM model is built by defining the model architecture such as the layers, the neurons, the hyperparameters, and the training parameters for four dataset topics. The mapped pre-trained word embedding matrix and the padded augmented training data will be used in the hybrid CNN-LSTM model training process. The training process will be done separately for the four dataset topics and split the training data into validation data by 10%. Sixth, the training result will be visualized and compared between training accuracy and training loss, the validation accuracy and validation loss. Eighth, the testing data will be used to test and evaluate the trained hybrid CNN-LSTM model, began with tokenization, changing text into sequence of positive integers, one-hot encoding for the test data label, and padding the sequence length. Afterwards, the padded testing sequence will be used for the trained hybrid CNN-LSTM model to test the predicted outcomes and evaluate the performance from the unseen data. Finally, the result of the hybrid CNN-LSTM model produced four models for Sinovac dataset, AstraZeneca dataset, Pfizer dataset, and Moderna dataset and saved them to be used in the sentiment analysis web application.

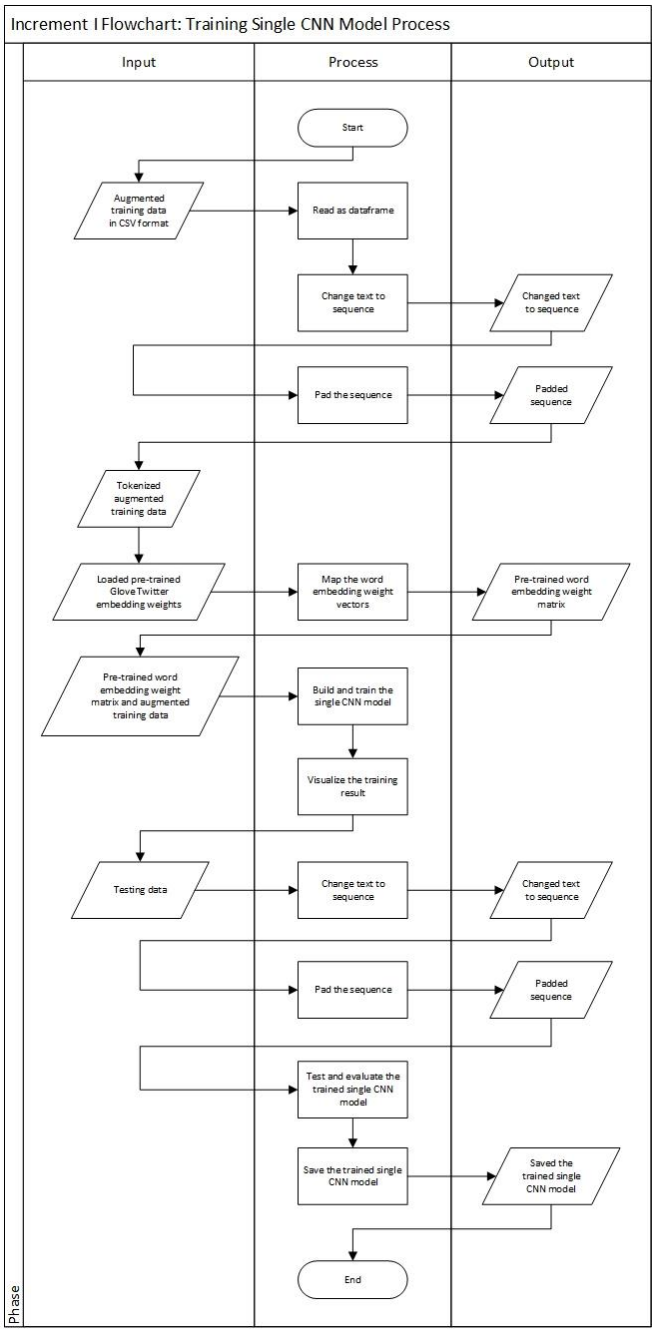


Figure 3.14 Training Single CNN Model Process Flowchart

The Figure 3.14 depicts the flowchart of the single CNN model training process in detail. The process is exactly the same as the training hybrid CNN-LSTM model training process. The single CNN model training process also used the exact same model architecture as the hybrid CNN-LSTM so the model comparison is valid. Finally, the result of the single CNN model will be also four models for

Sinovac dataset, AstraZeneca dataset, Pfizer dataset, and Moderna dataset and saved them to be used in the sentiment analysis web application.

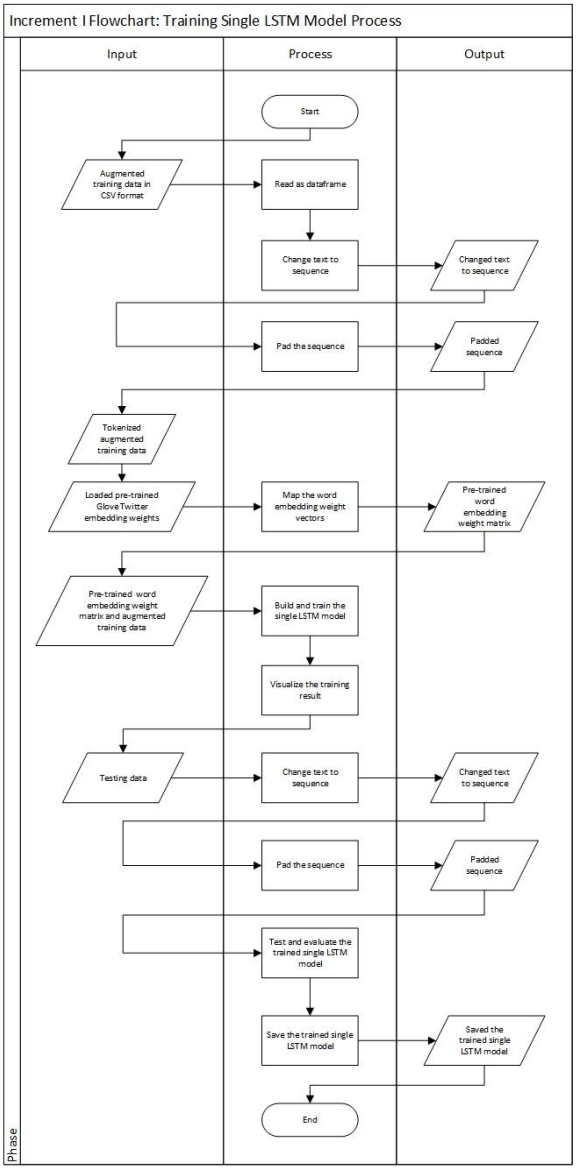


Figure 3.15 Increment I Flowchart: Training Single LSTM Model Process

The Figure 3.15 depicts the flowchart of the single LSTM model training process in detail. The process is exactly the same as the training hybrid CNN-LSTM model training process. The single LSTM model training process also used the exact same model architecture as the hybrid CNN-LSTM so the model comparison is valid. Finally, the result of the single LSTM model will be also four models for

Sinovac dataset, AstraZeneca dataset, Pfizer dataset, and Moderna dataset and saved them to be used in the sentiment analysis web application.

3.4.2.2 Sentiment Analysis Model Architecture Design

This section depicts and defines the model architecture design for the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model. The model architecture design covers the number of layers, the number of neurons, and the model hyperparameters depicted with diagram. The three sentiment analysis models used exactly same training parameters that are 128 batch size, 500 epochs, categorical loss entropy, and ADAM optimizer with 0,001 learning rate,

3.4.2.2.1 Hybrid CNN-LSTM Model Design

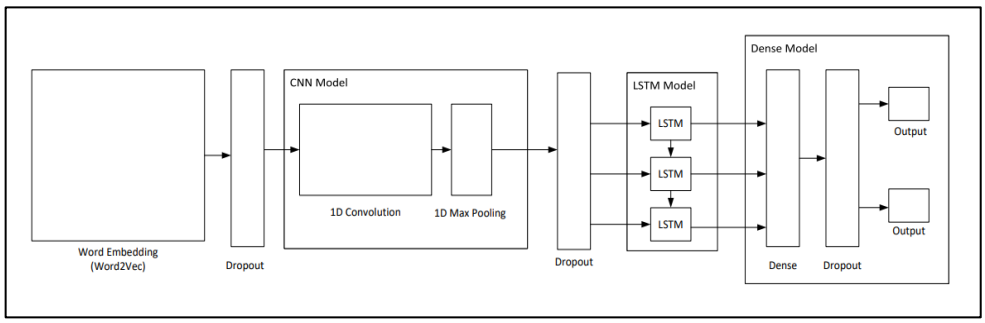


Figure 3.16 The Hybrid CNN-LSTM Model Architecture Design

The Figure 3.16 depicts the hybrid CNN-LSTM model architecture design. First, the word embedding layer weight uses the trained custom Word2Vec model with my own corpus. The mapped pre-trained Glove Twitter word embedding with own corpus has 200-dimension vectors. Then, the dropout layer is applied after the word embedding layer with 50% dropout rate. The hybrid CNN-LSTM model architecture is similar to [21]. The CNN model consisted of one convolution layer and one max pooling layer. The 1D convolution layer and 1D max pooling layer are used. The 1D convolution layer has 64 filter units, 3 kernel size, 1 stride, valid padding, and ReLU activation function. The 1D max pooling layer has 2 pool size and 1 stride. Then, the dropout layer is applied after the CNN model with 50% dropout rate. Further, the CNN model output will be fit to the LSTM model that has one layer LSTM. The LSTM layer has 128 units, 50% dropout rate, and tanh activation function. Finally, the output from the LSTM model

will be fit to the two layers classifier dense (basic neural network) model. The first dense layer has 128 units and ReLU activation function, followed by dropout layer with 50% dropout rate. The second layer has 2 units and softmax activation function as the output layer.

3.4.2.2.2 Single CNN Model Design

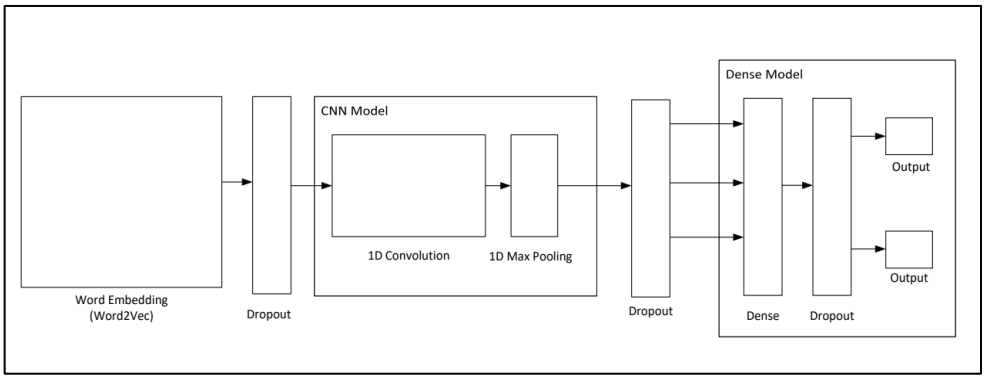


Figure 3.17 The Single CNN Model Architecture Design

The Figure 3.17 depicts the single CNN model architecture design has the exact same parameters with the hybrid CNN-LSTM model. First, the word embedding layer weight uses the trained custom Word2Vec model with my own corpus. The mapped pre-trained Glove Twitter word embedding with own corpus has 200-dimension vectors. Then, the dropout layer is applied after the word embedding layer with 50% dropout rate. The CNN model consisted of one convolution layer and one max pooling layer. The 1D convolution layer and 1D max pooling layer are used. The 1D convolution layer has 64 filter units, 3 kernel size, 1 stride, valid padding, and ReLU activation function. The 1D max pooling layer has 2 pool size and 1 stride. Then, the dropout layer is applied after the CNN model with 50% dropout rate. Further, the CNN model output will be fit to the two layers classifier dense (basic neural network) model. The first dense layer has 128 units and ReLU activation function, followed by dropout layer with 50% dropout rate. The second layer has 2 units and softmax activation function as the output layer.

3.4.2.2.3 Single LSTM Model Design

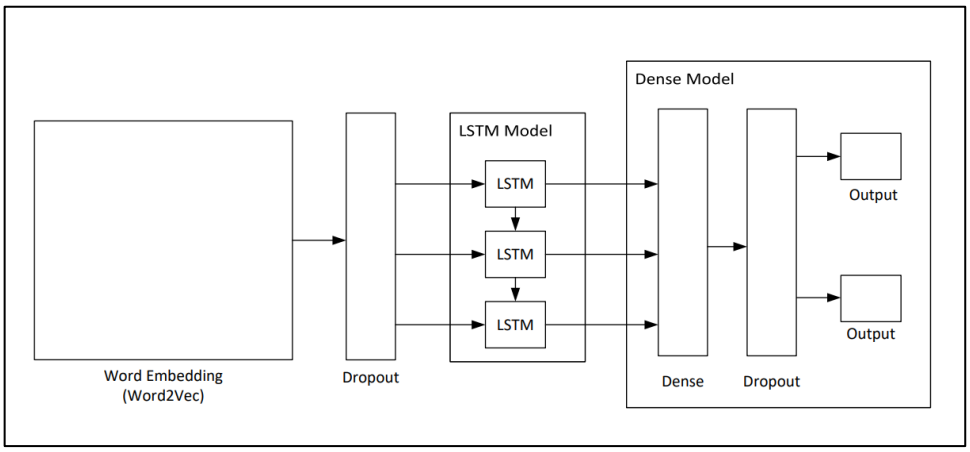


Figure 3.18 The Single LSTM Model Architecture Design

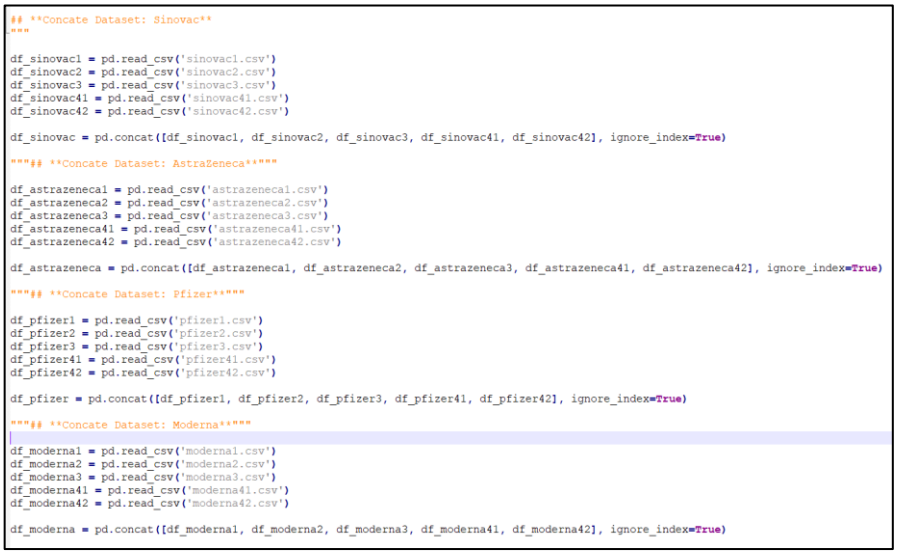
The Figure 3.18 depicts the single LSTM model architecture design also has the exact same parameters with the hybrid CNN-LSTM model. First, the word embedding layer weight uses the trained custom Word2Vec model with my own corpus. The mapped pre-trained Glove Twitter word embedding with own corpus has 200-dimension vectors. Then, the dropout layer is applied after the word embedding layer with 50% dropout rate. Then, the LSTM layer has 128 units, 50% dropout rate, and tanh activation function. Finally, the output from the LSTM model will be fit to the two layers classifier dense (basic neural network) model. The first dense layer has 128 units and ReLU activation function, followed by dropout layer with 50% dropout rate. The second layer has 2 units and softmax activation function as the output layer.

3.4.3 Implementation

```
"""# **Crawling Twitter Dataset**"""

ACCESS_TOKEN = '1491321301377904640-3f34SrhEWeXTNchIPVp7P1ZETxaoN0'
ACCESS_TOKEN_SECRET = 'sc3mB96PjXkR53e5K4g8binVxuohShV1WYZsuxkbofxCH'
CONSUMER_KEY = '77fSLVCOYcm2pX5rwxieKYEI4'
CONSUMER_SECRET = 'nikfO3dn95p5em3hgXUO7SYFuHpH3DS7S0WByOvoDaE3L9zbD1'

auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_TOKEN_SECRET)
api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)

search_sinovac_keywords = '(sinovac OR #sinovac) -moderna -pfizer -astrazeneca lang:en -is:retweet'
search_astrazeneca_keywords = '(astrazeneca OR #astrazeneca) -moderna -pfizer -sinovac lang:en -is:retweet'
search_pfizer_keywords = '(pfizer OR #pfizer) -sinovac -moderna -astrazeneca lang:en -is:retweet'
search_moderna_keywords = '(moderna OR #moderna) -sinovac -pfizer -astrazeneca lang:en -is:retweet'

tweets = tweepy.Cursor(api.search, q=search_astrazeneca_keywords, tweet_mode='extended', lang='en').items(2000)

tweet_texts = []
tweet_dates = []
tweet_langs = []
for tweet in tweets:
  tweet_texts.append(tweet.full_text)
  tweet_dates.append(tweet.lang)
  tweet_langs.append(tweet.created_at)
```

Figure 3.19 The Twitter Data Crawling Code

The Figure 3.19 shows the code to crawl the Twitter data using tweepy librrary. First, I defined several variables to hold the API token and the API key. Second, initialize the tweepy Twitter API authentication *OAuthHandler()* object and *set_access_token()* function by giving the API token and the API key as the parameters. Third, I defined several variables to hold the Twitter API search query by using keyword terms, hashtag, and negation operations to search topics about Sinovac vaccine, Moderna vaccine, Pfizer vaccine, and AstraZeneca vaccine respectively. Forth, I define the crawler variable *tweets* to crawl over based on the given search query on the parameter *q* and I changed it to another search query to adjust what topic that I search for. Finally, the crawled tweets are stored into *pandas* dataframe by taking the tweet texts, tweet dates, and tweet languages as the columns.



```
## **Concate Dataset: Sinovac**
"""

df_sinovac1 = pd.read_csv('sinovac1.csv')
df_sinovac2 = pd.read_csv('sinovac2.csv')
df_sinovac3 = pd.read_csv('sinovac3.csv')
df_sinovac41 = pd.read_csv('sinovac41.csv')
df_sinovac42 = pd.read_csv('sinovac42.csv')

df_sinovac = pd.concat([df_sinovac1, df_sinovac2, df_sinovac3, df_sinovac41, df_sinovac42], ignore_index=True)

"""## **Concate Dataset: AstraZeneca**"""

df_astrazeneca1 = pd.read_csv('astrazeneca1.csv')
df_astrazeneca2 = pd.read_csv('astrazeneca2.csv')
df_astrazeneca3 = pd.read_csv('astrazeneca3.csv')
df_astrazeneca41 = pd.read_csv('astrazeneca41.csv')
df_astrazeneca42 = pd.read_csv('astrazeneca42.csv')

df_astrazeneca = pd.concat([df_astrazeneca1, df_astrazeneca2, df_astrazeneca3, df_astrazeneca41, df_astrazeneca42], ignore_index=True)

"""## **Concate Dataset: Pfizer**"""

df_pfizer1 = pd.read_csv('pfizer1.csv')
df_pfizer2 = pd.read_csv('pfizer2.csv')
df_pfizer3 = pd.read_csv('pfizer3.csv')
df_pfizer41 = pd.read_csv('pfizer41.csv')
df_pfizer42 = pd.read_csv('pfizer42.csv')

df_pfizer = pd.concat([df_pfizer1, df_pfizer2, df_pfizer3, df_pfizer41, df_pfizer42], ignore_index=True)

"""## **Concate Dataset: Moderna**"""

df_moderna1 = pd.read_csv('moderna1.csv')
df_moderna2 = pd.read_csv('moderna2.csv')
df_moderna3 = pd.read_csv('moderna3.csv')
df_moderna41 = pd.read_csv('moderna41.csv')
df_moderna42 = pd.read_csv('moderna42.csv')

df_moderna = pd.concat([df_moderna1, df_moderna2, df_moderna3, df_moderna41, df_moderna42], ignore_index=True)
```

Figure 3.20 The Dataset Merging Code

The Figure 3.20 shows the code to merge all the crawled Twitter data into four dataset topics that imply four topics: Sinovac dataset, AstraZeneca dataset, Pfizer dataset, and Moderna dataset. I used the *pandas concat()* function to merge all the crawled Twitter data into one dataframe.

```
df_sinovac.to_csv('sinovac_merged.csv', index=False)
df_astrazeneca.to_csv('astrazeneca_merged.csv', index=False)
df_pfizer.to_csv('pfizer_merged.csv', index=False)
df_moderna.to_csv('moderna_merged.csv', index=False)
```

Figure 3.21 Save The Merged Dataset Into CSV Format Code

The Figure 3.21 shows the code to save the merged datasets into CSV format by using *pandas to_csv()* function to each dataset. The saved merged dataset naming format is “TOPICNAME_merged.csv”.

```
# Filter only March
df_sinovac = df_sinovac.drop(df_sinovac[df_sinovac['date'] < '2022-03-01 00:00:00'].index)
df_sinovac = df_sinovac.drop(df_sinovac[df_sinovac['date'] > '2022-03-31 24:59:59'].index)
df_sinovac = df_sinovac.drop(df_sinovac[df_sinovac['lang'] != 'en'].index)
df_sinovac = df_sinovac.reset_index(drop=True)
```

Figure 3.22 The Sinovac Dataset Filtering Code

```
# Filter only March
df_astrazeneca = df_astrazeneca.drop(df_astrazeneca[df_astrazeneca['date'] < '2022-03-01 00:00:00'].index)
df_astrazeneca = df_astrazeneca.drop(df_astrazeneca[df_astrazeneca['date'] > '2022-03-31 24:59:59'].index)
df_astrazeneca = df_astrazeneca.drop(df_astrazeneca[df_astrazeneca['lang'] != 'en'].index)
df_astrazeneca = df_astrazeneca.reset_index(drop=True)
```

Figure 3.23 The AstraZeneca Dataset Filtering Code

```
# Filter only March
df_pfizer = df_pfizer.drop(df_pfizer[df_pfizer['date'] < '2022-03-01 00:00:00'].index)
df_pfizer = df_pfizer.drop(df_pfizer[df_pfizer['date'] > '2022-03-31 24:59:59'].index)
df_pfizer = df_pfizer.drop(df_pfizer[df_pfizer['lang'] != 'en'].index)
df_pfizer = df_pfizer.reset_index(drop=True)
```

Figure 3.24 The Pfizer Dataset Filtering Code

```
# Filter only March
df_moderna = df_moderna.drop(df_moderna[df_moderna['date'] < '2022-03-01 00:00:00'].index)
df_moderna = df_moderna.drop(df_moderna[df_moderna['date'] > '2022-03-31 24:59:59'].index)
df_moderna = df_moderna.drop(df_moderna[df_moderna['lang'] != 'en'].index)
df_moderna = df_moderna.reset_index(drop=True)
```

Figure 3.25 The Moderna Dataset Filtering Code

The Figure 3.22, Figure 3.23, Figure 3.24, and Figure 3.25 shows the code to filter the four merged datasets to filter the dataset only on March 2022 period and in English language. I used *pandas drop()* function to remove all tweets that are not in March 2022 and not in English language.

The next process is the dataset pre-processing, where all the pre-processing methods are wrapped in one class named *TweetTextCleaner*. The pre-processing class contains several functions according to the defined pre-processing requirement previously.

```
# Remove html tags in text
def remove_html_tags(self, text):
  cleaned_text = BeautifulSoup(text, 'lxml')
  return cleaned_text.get_text()
```

Figure 3.26 The Dataset Pre-Processing: Removing HTML Tags Code

The Figure 3.26 shows the one of pre-processing function code in *TweetTextCleaner* class to do the removing HTML tags. The removing HTML tags

used the *BeautifulSoup()* class from the *BeautifulSoup* library by passing the given text and the HTML parser "lxml".

```
# Remove 'RT' or 'rt' in text
def remove_retweets(self, text):
  cleaned_text = re.sub(r'\brt\b', '', text)
  return cleaned_text
```

Figure 3.27 The Dataset Pre-Processing: Removing Retweets Code

The Figure 3.27 shows the one of pre-processing function code in *TweetTextCleaner* class to do the removing retweets tags. The removing retweets used *sub()* function from the *re* library by passing the regular expression rule to remove the retweets notation "rt" at the beginning of the text, an empty string as the replacement if matched, and the given text.

```
# Remove URLs in text
def remove_urls(self, text):
  cleaned_text = re.sub('(?:http?\:\/\/|http?\:\/|http?\:|https?\:\/\/|https?\:\/|https?\:|www)\S+', '', text)
  return cleaned_text
```

Figure 3.28 The Dataset Pre-Processing: Removing URLs Code

The Figure 3.28 shows the one of pre-processing function code in *TweetTextCleaner* class to do the removing retweets. The removing URLs used *sub()* function from the *re* library by passing the regular expression rule to remove the URLs (contained "http", "https", "www"), an empty string as the replacement if matched, and the given text.

```
# Remove username with '@' in text
def remove_mentions(self, text):
  cleaned_text = re.sub('@[^\s]+', '', text)
  return cleaned_text
```

Figure 3.29 The Dataset Pre-Processing: Removing Mentions Code

The Figure 3.29 shows the one of pre-processing function code in *TweetTextCleaner* class to do the removing mentions. The removing mentions used *sub()* function from the *re* library by passing the regular expression rule to remove

the mention notation "@" with its username, an empty string as the replacement if matched, and the given text.

```
# Remove hashtags '#' in text
def remove_hashtags(self, text):
  cleaned_text = text.replace('#', '')
  return cleaned_text
```

Figure 3.30 The Dataset Pre-Processing: Removing Hashtags Code

The Figure 3.30 shows the one of pre-processing function code in *TweetTextCleaner* class to do the removing hashtags. The removing hashtags used *replace()* built-in Python string function by passing the regular expression rule to remove the hashtag notation "#" and an empty string as the replacement if matched.

```
# Normalized the accented character in text
def remove_non_ascii(self, text):
  cleanaed_text = text.encode('ascii', 'ignore').decode()
  cleaned_text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
  return cleaned_text
```

Figure 3.31 The Dataset Pre-Processing: Removing Non-ASCII Characters Code

The Figure 3.31 shows the one of pre-processing function code in *TweetTextCleaner* class to do the removing non-ASCII characters. The process is done by encoding the existing non-ASCII characters in the given text using *encode()* with "ignore" parameter and *decode()* built-in Python string function. Then, all non-ASCII in the text will be normalized with *normalize()* function from the *unicodedata* library.

```
# Change the written number in word to actual number in text
def change_word_to_number(self, text):
  changed_text = []
  for word in text.split():
    try:
        changed_text += [str(w2n.word_to_num(word))]
    except ValueError:
        changed_text += [word]
  changed_text = ' '.join(changed_text)
  return changed_text
```

Figure 3.32 The Dataset Pre-Processing: Changing Word to Number Code

The Figure 3.32 shows the one of pre-processing function code in *TweetTextCleaner* class to do the changing word to number. The changing word to number used *word2number* library. The process is done by looping through the split text with *split()* built-in Python string function, changing a number that written in word to an actual number with *word_to_num()* function, ignoring an actual word, and joining all words together into a whole text again.

```
# Remove numbers in text
def remove_numbers(self, text):
  cleaned_text = ''.join([word for word in text if not word.isdigit()])
  return cleaned_text
```

Figure 3.33 The Dataset Pre-Processing: Removing Numbers Code

The Figure 3.33 shows the one of pre-processing function code in *TweetTextCleaner* class to do the removing numbers. The removing numbers used *isdigit()* built-in Python string function. The process is done by looping through the text, removing if it is a number with *word isdigit()* function, ignoring a non-digits, and joining all words together into a whole text again.

```
# Lower case all words in text
def case_folding(self, text):
  return text.lower()
```

Figure 3.34 The Dataset Pre-Processing: Case Folding Code

The Figure 3.34 shows the one of pre-processing function code in *TweetTextCleaner* class to do the text case folding. The case folding turned all words into lower case using *lower()* built-in Python string function.

```
# Expand the existing contractions in text
def expand_contractions(self, text):
  contractions_expander = Contractions(kv_model=contractions_model)
  expanded_text = list(contractions_expander.expand_texts([text], precise=True))
  expanded_text = ' '.join(expanded_text)
  return expanded_text
```

Figure 3.35 The Dataset Pre-Processing: Expanding Contractions Code

The Figure 3.35 shows the one of pre-processing function code in *TweetTextCleaner* class to do the expanding contractions. The expanding contractions used *expand_texts()* function from *pycontractions* library. The process is done by initializing the *Contractions* class with pre-trained Google News Word2Vec model, implementing the *expand_texts()* function with the given text and precise parameter for more accurate result, and joining all words together into a whole text again.

```
# Get the correspond antonym to the given word
def get_antonym(self, word):
  word_antonyms = set()
  # Get the synsets to the given word
  for syn in wordnet.synsets(word):
    # Get the correspond lemmas to the given word
    for lemma in syn.lemmas():
      # Get all the antonyms to the given word
      for antonym in lemma.antonyms():
        word_antonyms.add(antonym.name())

  # Get the relevant antonym
  if len(word_antonyms) != 0:
    return word_antonyms.pop()
  else:
    return None
```

Figure 3.36 The Dataset Pre-Processing: Getting Antonym Word Code

The Figure 3.36 shows the one of pre-processing function code in *TweetTextCleaner* class to do the getting antonym word. The getting antonym word used *NLTK wordnet* library and this function is used on the *replace_negations()* function. The process is done by looping through the lookup word from the given word with *synsets()* function, looping through the lemmas from the given sysnsets word with *lemmas()* function, looping through the antonyms from the given lemmas, retrieving the antonym word from the available antonyms with *name()* function,

returning the first antonym word if the given word has an antonym, and returning *None* value if the given word do not has an antonym.

```
# Replace the negation words with the antonym in text
def replace_negation(self, text):
  replaced_text = []
  index = 0
  tokenized_text = self.tokenize(text)
  token_length = len(tokenized_text)
  while index < token_length:
    current_word = tokenized_text[index]
    # Check if word is negation
    if (current_word == 'not' or current_word == 'no') and index+1 < token_length:
      current_antonym = self.get_antonym(tokenized_text[index+1])
      # Replace if negation word
      if current_antonym:
        # Store the replaced negation word with the antonym
        replaced_text.append(current_antonym)
        index += 2
        continue
    # Store the non-negation word
    replaced_text.append(current_word)
    index += 1

  replaced_text = ' '.join(replaced_text)

  return replaced_text
```

Figure 3.37 The Dataset Pre-Processing: Replacing Negations Code

The Figure 3.37 shows the one of pre-processing function code in *TweetTextCleaner* class to do the replacing negations. The replacing negations used the *get_antonym()* function and the *tokenize()* function. The process is done by looping through the tokenized text, setting the current word, checking if the current word is "not" or "no" then it is a negation word, replacing the next word after "not" or "no" with its antonym, ignoring the word "not" or "no" also its next word after antonym replacement, ignoring if the current word is not "not" or "no" then it is not a negation word, and joining all words together into a whole text again.

```
# Remove punctuations in text
def remove_punctuations(self, text):
  english_punctuations = string.punctuation
  translator = str.maketrans('', '', english_punctuations)
  cleaned_text = text.translate(translator)
  return cleaned_text
```

Figure 3.38 The Dataset Pre-Processing: Removing Punctuations Code

The Figure 3.38 shows the one of pre-processing function code in *TweetTextCleaner* class to do the removing punctuations. The removing punctuations used *maketrans()* built-in Python string function and retrieved the list of punctuations from *string* library. The process is done by initializing the variable

to hold all the English language punctuations with *punctuation* attribute, replacing all the punctuations with an empty string using the *maketrans()* function, executing the punctuation replacement with *translate()* built-in Python string function, and joining all words together into a whole text again.

```
# Tokenize sentence into word
def tokenize(self, text):
  tokenized_text = word_tokenize(text)
  return tokenized_text
```

Figure 3.39 The Dataset Pre-Processing: Tokenizing Sentence Code

The Figure 3.39 shows the one of pre-processing function code in *TweetTextCleaner* class to do the tokenizing sentence. The tokenizing sentence used *word_tokenize()* function from *NLTK tokenize* library by passing the given text to be tokenized into token of words. This function is used for other pre-processing functions such as *replace_negations()* function, *remove_stopwords()* function, and *lemmatize()* function.

```
# Remove stopwords in text
def remove_stopwords(self, text):
  stop_words = stopwords.words('english')
  cleaned_text =  [word for word in self.tokenize(text) if word not in stop_words]
  cleaned_text = ' '.join(cleaned_text)
  return cleaned_text
```

Figure 3.40 The Dataset Pre-Processing: Removing Stopwords Code

The Figure 3.40 shows the one of pre-processing function code in *TweetTextCleaner* class to do the removing stopwords. The removing stopwords used *tokenize()* function and retrieved the list of stopwords from *NLTK stopwords* library. The process is done by initializing the variable to hold all the English stopwords using *words()* function with “english” parameter from *NLTK stopwords* library, ignoring all the existed stopwords, and joining all words together into a whole text again.

```
# Get the correspond word tag
def get_pos_tag(self,word):
  current_word_tag = nltk.pos_tag([word])[0][1][0].upper()
  tag_map = {'J': wordnet.ADJ,
             'N': wordnet.NOUN,
             'V': wordnet.VERB,
             'R': wordnet.ADV}

  return tag_map.get(current_word_tag, wordnet.NOUN)
```

Figure 3.41 The Dataset Pre-Processing: Getting Word's POS Tag Code

The Figure 3.41 shows the one of pre-processing function code in *TweetTextCleaner* class to do the getting the word's POS tag. The getting word's POS tag used *post_tag()* function from *NLTK* library by passing the given word to be matched with the correct POS tag (i.e., adjective, noun, verb, or adverb). Then, the retrieved POS tag mapped with the label "J", "N", "V" or "R" and returned the found label, otherwise, it returned the noun label as the default label. This function is used for other pre-processing function that is *lemmatize()* function.

```
# Lemmatize each word in text
def lemmatize(self, text):
    lemmatizer = WordNetLemmatizer()
    lematized_text = [lemmatizer.lemmatize(word, self.get_pos_tag(word)) for word in self.tokenize(text)]
    lematized_text = [word for word in lematized_text if len(word) > 2]
    lematized_text = ' '.join(lematized_text)
    return lematized_text
```

Figure 3.42 The Dataset Pre-Processing: Lemmatizing Words Code

The Figure 3.42 shows the one of pre-processing function code in *TweetTextCleaner* class to do the lemmatizing words. The lemmatizing words used *tokenize()* function, *get_pos_tag()* function, *lemmatize()* function from *NLTK WordNetLemmatizer* library. The process is done by initializing the *WordNetLemmatizer()* class, looping through the tokenized text, lemmatizing the word according to the retrieve POS tag, and joining all words together into a whole text again. Finally, several words in lemmatized text that only one or two characters is removed to reduce the noises in data.

Figure 3.43 The Dataset Pre-Processing: Sinovac Dataset Implementation Code

Figure 3. 44 The Dataset Pre-Processing: AstraZeneca Dataset Implementation Code

Figure 3. 45 The Dataset Pre-Processing: Pfizer Dataset Implementation Code

Figure 3. 46 The Dataset Pre-Processing: Moderna Dataset Implementation Code

The Figure 3.43, Figure 3.44, Figure 4.45, and Figure 4.46 shows the data pre-processing implementation on four dataset topics. The implementation uses *apply()* built-in *pandas* library function and use all the pre-processing functions from the *text_cleaner* object.

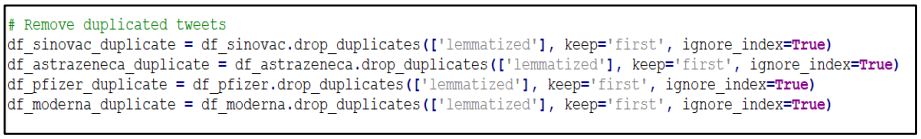


Figure 3.47 The Dataset Pre-Processing: Removing Duplicated Text Code

The Figure 3.46 shows the one of pre-processing to remove duplicated texts in the lemmatized text. The removing duplicated texts used *drop_duplicates()* built-in *pandas* library function by passing the subset column to identify the duplicated values, keeping parameter to keep the first occurrence, and reset the index.

Figure 3.48 The Dataset Pre-Processing: Dataset Selection Code

The Figure 3.48 shows the one of pre-processing to select all four dataset topics into experiment dataset and implementation dataset. The dataset selection used *drop()* built-in pandas library function by passing the conditional mask of which data to be excluded or removed and reset the index. The experiment dataset used 1 March 2022 until 21 March 2022 period, while the implementation dataset used 22 March 2022 until 31 March 2022 period.

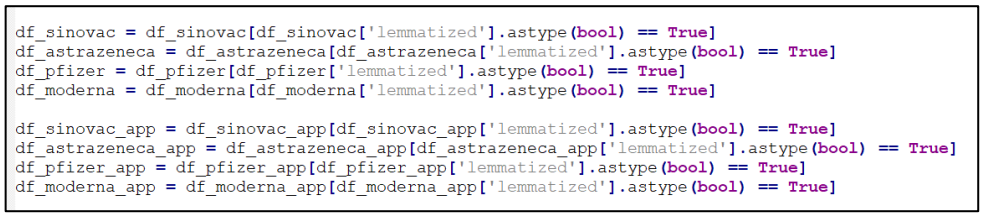


Figure 3.49 The Dataset Pre-Processing: Removing Empty Value Code

The Figure 3.49 shows the one of pre-processing to remove an empty value, because it is possible that the pre-processed texts are resulted empty with several words or characters is cleaned. The experiment dataset and implementation dataset are implemented with removing empty value. The removing empty value used conditional mask, where it returned true if data is not empty and vice versa. Therefore, the empty value that indicated false by *astype()* built-in pandas library function is ignored; selecting only the data that indicated true.

```
df_sinovac.to_csv('sinovac_cleaned.csv', index=False)
df_astrazeneca.to_csv('astrazeneca_cleaned.csv', index=False)
df_pfizer.to_csv('pfizer_cleaned.csv', index=False)
df_moderna.to_csv('moderna_cleaned.csv', index=False)

df_sinovac_app.to_csv('sinovac_app.csv', index=False)
df_astrazeneca_app.to_csv('astrazeneca_app.csv', index=False)
df_pfizer_app.to_csv('pfizer_app.csv', index=False)
df_moderna_app.to_csv('moderna_app.csv', index=False)
```

Figure 3.50 The Dataset Pre-Processing: Saving Dataset Code

The Figure 3.50 shows the last step of the dataset pre-processing to save the pre-processed experiment dataset and the implementation dataset in CSV format. The saving dataset used *to_csv()* built-in pandas library function by passing the file name and the reset index state.

```
def get_sentiment_label(text):
    vader = SentimentIntensityAnalyzer()
    vader_result = vader.polarity_scores(text)

    if vader_result['compound'] >= 0.05:
      labeled_sentiment = 'positive'
    elif vader_result['compound'] > -0.05 and vader_result['compound'] < 0.05:
      labeled_sentiment = 'neutral'
    else:
      labeled_sentiment = 'negative'

    return labeled_sentiment
```

Figure 3.51 The Dataset Labeling: Labeling Function Code

The Figure 3.51 shows the dataset labeling function so it can be re-used for four dataset topics. The dataset labeling used *vaderSentiment* library object and its built-in function. First, the *vaderSentiment* object is initialized with *SentimentIntensityAnalyzer()*. Second, through the initialized object, used *polarity_scores()* built-in vaderSentiment library function to retrieve the sentiment scores. Finally, the sentiment scores are determined through the compund value, where the sentiment is positive if the compound value equal or bigger than 0,05; the sentiment is neutral if the compound value is bigger than -0,05 and smaller than 0,05; and the sentiment is negative if the compound value is equal or smaller than -0,05.

```
# Applying VADER sentiment label
df_sinovac['sentiment'] = df_sinovac['removed_hashtags'].apply(lambda sentiment: get_sentiment_label(sentiment))
df_sinovac = df_sinovac[df_sinovac['sentiment'] != 'neutral']
df_sinovac = df_sinovac[['lemmatized', 'sentiment']].rename(columns={'lemmatized': 'text'})

df_astrazeneca['sentiment'] = df_astrazeneca['removed_hashtags'].apply(lambda sentiment: get_sentiment_label(sentiment))
df_astrazeneca = df_astrazeneca[df_astrazeneca['sentiment'] != 'neutral']
df_astrazeneca = df_astrazeneca[['lemmatized', 'sentiment']].rename(columns={'lemmatized': 'text'})

df_pfizer['sentiment'] = df_pfizer['removed_hashtags'].apply(lambda sentiment: get_sentiment_label(sentiment))
df_pfizer = df_pfizer[df_pfizer['sentiment'] != 'neutral']
df_pfizer = df_pfizer[['lemmatized', 'sentiment']].rename(columns={'lemmatized': 'text'})

df_moderna['sentiment'] = df_moderna['removed_hashtags'].apply(lambda sentiment: get_sentiment_label(sentiment))
df_moderna = df_moderna[df_moderna['sentiment'] != 'neutral']
df_moderna = df_moderna[['lemmatized', 'sentiment']].rename(columns={'lemmatized': 'text'})
```

Figure 3.52 The Dataset Labeling: Applying Labeling Function Code

The Figure 3.52 shows the implementation dataset labeling with built function for four dataset topics. The applying labeling function used *apply()* pandas library built-in function by passing the labeling function to label each data. The removed hashtags on experiment dataset are used because vaderSentiment could give a sentiment label by considering negations, emoticons, contractions, punctuations, slang words, and non-ASCII characters that can give sentiments more accurate. Finally, the labeled four dataset topics is filtered with only positive and negative sentiment to be used and renamed their columns.

```
df_sinovac.to_csv('labeled/sinovac_labeled.csv', index=False)
df_astrazeneca.to_csv('labeled/astrazeneca_labeled.csv', index=False)
df_pfizer.to_csv('labeled/pfizer_labeled.csv', index=False)
df_moderna.to_csv('labeled/moderna_labeled.csv', index=False)
```

Figure 3.53 The Dataset Labeling: Saving Dataset Code

The Figure 3.53 shows the last step of the dataset labeling to save the labeled experiment dataset in CSV format. The saving dataset used *to_csv()* built-in pandas library function by passing the file name and the reset index state.

```python
def split_data(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        train_size=0.9,
                                                        test_size=0.1,
                                                        stratify=y,
                                                        shuffle=True,
                                                        random_state=0)

    return X_train, X_test, y_train, y_test
```

Figure 3.54 The Dataset Splitting: Splitting Function Code

The Figure 3.54 shows the dataset splitting function so it can be re-used for four dataset topics. The dataset splitting used *scikit-learn* library and its built-in function. The dataset is split using *train_test_split()* built-in *scikit-learn* library function, by passing feature data, label data, portion for training data that is 90%, portion for testing data that is 10%, balanced split respective to the label data, shuffle the data before split that is true, and randomization state. The dataset splitting function returned the feature and label for training and testing data.

```python
# Split the Sinovac dataset into training, validation, and testing
X_sinovac = df_sinovac['text'].values
y_sinovac = df_sinovac['sentiment'].values

X_sinovac_train, X_sinovac_test, y_sinovac_train, y_sinovac_test = split_data(X_sinovac, y_sinovac)

df_sinovac_train = pd.DataFrame({'text': X_sinovac_train, 'sentiment': y_sinovac_train})
df_sinovac_test = pd.DataFrame({'text': X_sinovac_test, 'sentiment': y_sinovac_test})

# Split the AstraZeneca dataset into training, validation, and testing
X_astrazeneca = df_astrazeneca['text'].values
y_astrazeneca = df_astrazeneca['sentiment'].values

X_astrazeneca_train, X_astrazeneca_test, y_astrazeneca_train, y_astrazeneca_test = split_data(X_astrazeneca, y_astrazeneca)

df_astrazeneca_train = pd.DataFrame({'text': X_astrazeneca_train, 'sentiment': y_astrazeneca_train})
df_astrazeneca_test = pd.DataFrame({'text': X_astrazeneca_test, 'sentiment': y_astrazeneca_test})

# Split the Pfizer dataset into training, validation, and testing
X_pfizer = df_pfizer['text'].values
y_pfizer = df_pfizer['sentiment'].values

X_pfizer_train, X_pfizer_test, y_pfizer_train, y_pfizer_test = split_data(X_pfizer, y_pfizer)

df_pfizer_train = pd.DataFrame({'text': X_pfizer_train, 'sentiment': y_pfizer_train})
df_pfizer_test = pd.DataFrame({'text': X_pfizer_test, 'sentiment': y_pfizer_test})

# Split the Pfizer dataset into training, validation, and testing
X_moderna = df_moderna['text'].values
y_moderna = df_moderna['sentiment'].values

X_moderna_train, X_moderna_test, y_moderna_train, y_moderna_test = split_data(X_moderna, y_moderna)

df_moderna_train = pd.DataFrame({'text': X_moderna_train, 'sentiment': y_moderna_train})
df_moderna_test = pd.DataFrame({'text': X_moderna_test, 'sentiment': y_moderna_test})
```

Figure 3.55 The Dataset Splitting: Applying Splitting Function Code

The Figure 3.55 shows the implementation dataset splitting through *split_data()* function that built before for four dataset topics. The applying splitting function used *apply()* built-in *pandas* library function by passing the labeling function to label each data. The text and label data on each four experiment dataset

topics are used. Finally, the split four dataset topics is stored into two new dataframes to hold the training and testing data.

```
df_sinovac_train = pd.read_csv('train/sinovac_train.csv')
df_astrazeneca_train = pd.read_csv('train/astrazeneca_train.csv')
df_pfizer_train = pd.read_csv('train/pfizer_train.csv')
df_moderna_train = pd.read_csv('train/moderna_train.csv')

df_sinovac_test = pd.read_csv('test/sinovac_test.csv')
df_astrazeneca_test = pd.read_csv('test/astrazeneca_test.csv')
df_pfizer_test = pd.read_csv('test/pfizer_test.csv')
df_moderna_test = pd.read_csv('test/moderna_test.csv')
```

Figure 3.56 The Dataset Splitting: Saving Dataset Code

The Figure 3.56 shows the last step of the dataset splitting to save the split experiment dataset in CSV format such as the training and testing data for four dataset topics. The saving dataset used *to_csv()* built-in *pandas* library function by passing the file name and the reset index state.

```
class TextAugmentation:
    def __init__(self):
        self.augmenter = EDA()

    def replace_synonym(self, text):
        augmented_text_portion = max(1, int(len(text)*0.1))
        synonym_replaced = self.augmenter.synonym_replacement(text, n=augmented_text_portion)
        return synonym_replaced

    def random_insert(self, text):
        augmented_text_portion = max(1, int(len(text)*0.1))
        random_inserted = self.augmenter.random_insertion(text, n=augmented_text_portion)
        return random_inserted

    def random_swap(self, text):
        augmented_text_portion = max(1, int(len(text)*0.1))
        random_swaped = self.augmenter.random_swap(text, n=augmented_text_portion)
        return random_swaped

    def random_delete(self, text):
        random_deleted = self.augmenter.random_deletion(text, p=0.1)
        return random_deleted

text_augmentation = TextAugmentation()
```

Figure 3.57 The Dataset Augmentation: EDA Class Code

The Figure 3.57 shows the dataset augmentation class that implemented EDA (Easy Data Augmentation) so it can be re-used for four dataset topics. The dataset augmentation used *textagment* library by initializing the *EDA* object and each EDA methods is divided into methods inside the class. The replace synonym augmentation used *synonym_replacement()* built-in EDA object function from the *textaugment* library, by passing the text and 10% alpha for how many words to be replaced by its synonym. The random insertion augmentation used

random_insertion() built-in EDA object function from the *textaugment* library, function by passing the text and 10% alpha for how many new words to be inserted randomly. The random swap augmentation used *random_swap()* built-in EDA object from the *textaugment* library, by passing the text and 10% alpha for how many words to be swapped randomly. The random swap augmentation used *random_swap()* built-in EDA object from the *textaugment* library, by passing the text and 10% alpha for how many words to be swapped randomly.

```
# Data augmentation for positive class in Sinovac training data
synonym_replacement_train_positive = df_sinovac_train_positive['text'].apply(lambda text: text_augmentation.replace_synonym(text))
random_insertion_train_positive = df_sinovac_train_positive['text'].apply(lambda text: text_augmentation.random_insert(text))
random_swap_train_positive = df_sinovac_train_positive['text'].apply(lambda text: text_augmentation.random_swap(text))
random_deletion_train_positive = df_sinovac_train_positive['text'].apply(lambda text: text_augmentation.random_delete(text))

# Data augmentation for negative class in Sinovac training data
synonym_replacement_train_negative = df_sinovac_train_negative['text'].apply(lambda text: text_augmentation.replace_synonym(text))
random_insertion_train_negative = df_sinovac_train_negative['text'].apply(lambda text: text_augmentation.random_insert(text))
random_swap_train_negative = df_sinovac_train_negative['text'].apply(lambda text: text_augmentation.random_swap(text))
random_deletion_train_negative = df_sinovac_train_negative['text'].apply(lambda text: text_augmentation.random_delete(text))
```

Figure 3.58 The Dataset Augmentation: Applying EDA Function for Sinovac Dataset Code

```
# Data augmentation for positive class in AstraZeneca training data
synonym_replacement_train_positive = df_astrazeneca_train_positive['text'].apply(lambda text: text_augmentation.replace_synonym(text))
random_insertion_train_positive = df_astrazeneca_train_positive['text'].apply(lambda text: text_augmentation.random_insert(text))
random_swap_train_positive = df_astrazeneca_train_positive['text'].apply(lambda text: text_augmentation.random_swap(text))
random_deletion_train_positive = df_astrazeneca_train_positive['text'].apply(lambda text: text_augmentation.random_delete(text))

# Data augmentation for negative class in AstraZeneca training data
synonym_replacement_train_negative = df_astrazeneca_train_negative['text'].apply(lambda text: text_augmentation.replace_synonym(text))
random_insertion_train_negative = df_astrazeneca_train_negative['text'].apply(lambda text: text_augmentation.random_insert(text))
random_swap_train_negative = df_astrazeneca_train_negative['text'].apply(lambda text: text_augmentation.random_swap(text))
random_deletion_train_negative = df_astrazeneca_train_negative['text'].apply(lambda text: text_augmentation.random_delete(text))
```

Figure 3.59 The Dataset Augmentation: Applying EDA Function for AstraZenca Dataset Code

```
# Data augmentation for positive class in Pfizer training data
synonym_replacement_train_positive = df_pfizer_train_positive['text'].apply(lambda text: text_augmentation.replace_synonym(text))
random_insertion_train_positive = df_pfizer_train_positive['text'].apply(lambda text: text_augmentation.random_insert(text))
random_swap_train_positive = df_pfizer_train_positive['text'].apply(lambda text: text_augmentation.random_swap(text))
random_deletion_train_positive = df_pfizer_train_positive['text'].apply(lambda text: text_augmentation.random_delete(text))

# Data augmentation for negative class in Pfizer training data
synonym_replacement_train_negative = df_pfizer_train_negative['text'].apply(lambda text: text_augmentation.replace_synonym(text))
random_insertion_train_negative = df_pfizer_train_negative['text'].apply(lambda text: text_augmentation.random_insert(text))
random_swap_train_negative = df_pfizer_train_negative['text'].apply(lambda text: text_augmentation.random_swap(text))
random_deletion_train_negative = df_pfizer_train_negative['text'].apply(lambda text: text_augmentation.random_delete(text))
```

Figure 3.60 The Dataset Augmentation: Applying EDA Function for Pfizer Dataset Code

```
# Data augmentation for positive class in Moderna training data
synonym_replacement_train_positive = df_moderna_train_positive['text'].apply(lambda text: text_augmentation.replace_synonym(text))
random_insertion_train_positive = df_moderna_train_positive['text'].apply(lambda text: text_augmentation.random_insert(text))
random_swap_train_positive = df_moderna_train_positive['text'].apply(lambda text: text_augmentation.random_swap(text))
random_deletion_train_positive = df_moderna_train_positive['text'].apply(lambda text: text_augmentation.random_delete(text))

# Data augmentation for negative class in Moderna training data
synonym_replacement_train_negative = df_moderna_train_negative['text'].apply(lambda text: text_augmentation.replace_synonym(text))
random_insertion_train_negative = df_moderna_train_negative['text'].apply(lambda text: text_augmentation.random_insert(text))
random_swap_train_negative = df_moderna_train_negative['text'].apply(lambda text: text_augmentation.random_swap(text))
random_deletion_train_negative = df_moderna_train_negative['text'].apply(lambda text: text_augmentation.random_delete(text))
```

Figure 3.61 The Dataset Augmentation: Applying EDA Function for Moderna Dataset Code

The Figure 3.58, Figure 3.59, Figure 3.60, and Figure 3.61 shows the implementation dataset augmentation through EDA class that built before for four dataset topics. The applying EDA function used *apply()* built-in *pandas* library

function by passing the each EDA functions to augment each text data. The EDA is implemented both negative and positive sentiment for four training dataset topics. Finally, the augmented four training dataset topics is stored into new dataframes and saved in CSV format.

```
def encode_label(sentiment):
    if sentiment == 'negative':
        return 0
    elif sentiment == 'positive':
        return 1
```

Figure 3.62 The Sentiment Analysis Model Building: Encoding Label Function Code

The Figure 3.62 shows the encoding label function so it can be re-used for four dataset topics. The dataset encoding label function simply change the label into numerical value, if the label is negative sentiment, then encode with number 0, otherwise if the label is positive sentiment, then encode with number 1.

```
# Tokenize the sequence
sinovac_max_length = 100
sinovac_tokenizer = Tokenizer()
sinovac_tokenizer.fit_on_texts(df_sinovac['text'].values)

X_sinovac_train_tokenized = sinovac_tokenizer.texts_to_sequences(df_sinovac_train['text'].values)
X_sinovac_test_tokenized = sinovac_tokenizer.texts_to_sequences(df_sinovac_test['text'].values)

# Pad the sequence
X_sinovac_train_padded = pad_sequences(X_sinovac_train_tokenized, maxlen=sinovac_max_length)
X_sinovac_test_padded = pad_sequences(X_sinovac_test_tokenized, maxlen=sinovac_max_length)

sinovac_num_words = len(sinovac_tokenizer.word_index) + 1

# Encode label
df_sinovac_train['sentiment'] = df_sinovac_train['sentiment'].apply(lambda x: encode_label(x))
df_sinovac_test['sentiment'] = df_sinovac_test['sentiment'].apply(lambda x: encode_label(x))

y_sinovac_train_category = to_categorical(df_sinovac_train['sentiment'])
y_sinovac_test_category = to_categorical(df_sinovac_test['sentiment'])
```

Figure 3.63 The Sentiment Analysis Model Building: Sinovac Data Transformation Code

The Figure 3.63 shows the data transformation for Sinovac dataset to prepare the data into a proper form for the sentiment analysis model. First, all the text data in Sinovac dataset is tokenized using *fit_on_texts()* from *keras* library to build the unique word tokens with its numeric values. Second, each word from the train and test data will be encoded into numeric values based on the built word tokens previously. Third, the tokenized train and test data will be padded to make

the dimension equal with 100 maximum sequences. Finally, after transform the text data, the label data for training and testing also will be encoded with *encode_label()* function and transformed into categorical one-hot encoding with *to_categorical()* from *keras* library.

```
# Tokenize the sequence
astrazeneca_max_length = 100
astrazeneca_tokenizer = Tokenizer()
astrazeneca_tokenizer.fit_on_texts(df_astrazeneca['text'].values)

X_astrazeneca_train_tokenized = astrazeneca_tokenizer.texts_to_sequences(df_astrazeneca_train['text'].values)
X_astrazeneca_test_tokenized = astrazeneca_tokenizer.texts_to_sequences(df_astrazeneca_test['text'].values)

# Pad the sequence
X_astrazeneca_train_padded = pad_sequences(X_astrazeneca_train_tokenized, maxlen=astrazeneca_max_length)
X_astrazeneca_test_padded = pad_sequences(X_astrazeneca_test_tokenized, maxlen=astrazeneca_max_length)

astrazeneca_num_words = len(astrazeneca_tokenizer.word_index) + 1

# Encode label
df_astrazeneca_train['sentiment'] = df_astrazeneca_train['sentiment'].apply(lambda x: encode_label(x))
df_astrazeneca_test['sentiment'] = df_astrazeneca_test['sentiment'].apply(lambda x: encode_label(x))

y_astrazeneca_train_category = to_categorical(df_astrazeneca_train['sentiment'])
y_astrazeneca_test_category = to_categorical(df_astrazeneca_test['sentiment'])
```

Figure 3.64 The Sentiment Analysis Model Building: AstraZeneca Dataset Transformation Code

The Figure 3.64 shows the data transformation for AstraZeneca dataset to prepare the data into a proper form for the sentiment analysis model. First, all the text data in AstraZeneca dataset is tokenized using *fit_on_texts()* from *keras* library to build the unique word tokens with its numeric values. Second, each word from the train and test data will be encoded into numeric values based on the built word tokens previously. Third, the tokenized train and test data will be padded to make the dimension equal with 100 maximum sequences. Finally, after transform the text data, the label data for training and testing also will be encoded with *encode_label()* function and transformed into categorical one-hot encoding with *to_categorical()* from *keras* library.

```
# Tokenize the sequence
pfizer_max_length = 100
pfizer_tokenizer = Tokenizer()
pfizer_tokenizer.fit_on_texts(df_pfizer['text'].values)

X_pfizer_train_tokenized = pfizer_tokenizer.texts_to_sequences(df_pfizer_train['text'].values)
X_pfizer_test_tokenized = pfizer_tokenizer.texts_to_sequences(df_pfizer_test['text'].values)

# Pad the sequence
X_pfizer_train_padded = pad_sequences(X_pfizer_train_tokenized, maxlen=pfizer_max_length)
X_pfizer_test_padded = pad_sequences(X_pfizer_test_tokenized, maxlen=pfizer_max_length)

pfizer_num_words = len(pfizer_tokenizer.word_index) + 1

# Encode label
df_pfizer_train['sentiment'] = df_pfizer_train['sentiment'].apply(lambda x: encode_label(x))
df_pfizer_test['sentiment'] = df_pfizer_test['sentiment'].apply(lambda x: encode_label(x))

y_pfizer_train_category = to_categorical(df_pfizer_train['sentiment'])
y_pfizer_test_category = to_categorical(df_pfizer_test['sentiment'])
```

Figure 3.65 The Sentiment Analysis Model Building: Pfizer Dataset Transformation Code

The Figure 3.65 shows the data transformation for Pfizer dataset to prepare the data into a proper form for the sentiment analysis model. First, all the text data in Pfizer dataset is tokenized using *fit_on_texts()* from *keras* library to build the unique word tokens with correspond numeric values. Second, each word from the train and test data will be encoded into numeric values based on the built word tokens previously. Third, the tokenized train and test data will be padded to make the dimension equal with 100 maximum sequences. Finally, after transform the text data, the label data for training and testing also will be encoded with *encode_label()* function and transformed into categorical one-hot encoding with *to_categorical()* from *keras* library.

```
# Tokenize the sequence
moderna_max_length = 100
moderna_tokenizer = Tokenizer()
moderna_tokenizer.fit_on_texts(df_moderna['text'].values)

X_moderna_train_tokenized = moderna_tokenizer.texts_to_sequences(df_moderna_train['text'].values)
X_moderna_test_tokenized = moderna_tokenizer.texts_to_sequences(df_moderna_test['text'].values)

# Pad the sequence
X_moderna_train_padded = pad_sequences(X_moderna_train_tokenized, maxlen=moderna_max_length)
X_moderna_test_padded = pad_sequences(X_moderna_test_tokenized, maxlen=moderna_max_length)

moderna_num_words = len(moderna_tokenizer.word_index) + 1

# Encode label
df_moderna_train['sentiment'] = df_moderna_train['sentiment'].apply(lambda x: encode_label(x))
df_moderna_test['sentiment'] = df_moderna_test['sentiment'].apply(lambda x: encode_label(x))

y_moderna_train_category = to_categorical(df_moderna_train['sentiment'])
y_moderna_test_category = to_categorical(df_moderna_test['sentiment'])
```

Figure 3.66 The Sentiment Analysis Model Building: Moderna Dataset Transformation Code

The Figure 3.66 shows the data transformation for Moderna dataset to prepare the data into a proper form for the sentiment analysis model. First, all the

text data in Moderna dataset is tokenized using *fit_on_texts()* from *keras* library to build the unique word tokens with correspond numeric values. Second, each word from the train and test data will be encoded into numeric values based on the built word tokens previously. Third, the tokenized train and test data will be padded to make the dimension equal with 100 maximum sequences. Finally, after transform the text data, the label data for training and testing also will be encoded with *encode_label()* function and transformed into categorical one-hot encoding with *to_categorical()* from *keras* library.

```
GLOVE_WORD_EMBEDDING = load_word_embedding('glove.twitter.27B.200d.txt')

def load_word_embedding(file_name):
    embeddings_weight_vector = {}

    with open(os.path.join('', file_name)) as file:
        for line in file:
            values = line.split();
            word = values[0]
            weights = np.asarray(values[1:], dtype='float32')
            embeddings_weight_vector[word] = weights;

    return embeddings_weight_vector

def map_word_embedding(word_index, num_words, embedding_weight_vectors):
    embedding_matrix = np.zeros((num_words, 200))
    for word, index in word_index.items():
        embedding_vector = embedding_weight_vectors.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector

    return embedding_matrix
```

Figure 3.67 The Sentiment Analysis Model Building: Loading Pre-Trained Glove Twitter Word Embedding Code

The Figure 3.67 shows the loading pre-trained Glove Twitter word embedding. The *load_word_embedding()* function is used to load the pre-trained Glove Twitter word embedding file and looped each line to map each word with its vector weights. The *map_word_embedding()* function is used to map each word from the tokenized and encoded sequence with correspond pre-trained Glove Twitter word embedding weights with 200 vectors that represent the words. If a word in the tokenized and encoded sequence exist in the pre-trained Glove Twitter word embedding, then assign them with their corresponded vector weights, otherwise, if the word in the tokenized and encoded sequence in the pre-trained Glove Twitter word embedding, then the correspond word is left with zero value vector weights.

```
# Build hybrid CNN-LSTM model
def build_cnn_lstm_model(embedding_matrix, max_sequence_length):
    # Input layer
    input_layer = Input(shape=(max_sequence_length,))

    # Word embedding layer
    embedding_layer = Embedding(input_dim=embedding_matrix.shape[0],
                                output_dim=embedding_matrix.shape[1],
                                weights=[embedding_matrix],
                                input_length=max_sequence_length,
                                trainable=True)(input_layer)
    dropout_layer = Dropout(rate=0.5)(embedding_layer)

    # CNN model layer
    cnn_layer = Conv1D(filters=64,
                       kernel_size=3,
                       strides=1,
                       activation='relu')(dropout_layer)
    cnn_layer = MaxPooling1D(pool_size=2)(cnn_layer)
    cnn_layer = Dropout(rate=0.5)(cnn_layer)

    # LSTM model layer
    lstm_layer = LSTM(units=128,
                      dropout=0.5,
                      return_sequences=False)(cnn_layer)

    # Dense model layer
    dense_layer = Dense(units=128, activation='relu')(lstm_layer)
    dropout_layer = Dropout(rate=0.5)(dense_layer)

    output_layer = Dense(units=2, activation='softmax')(dropout_layer)

    cnn_lstm_model = Model(inputs=input_layer, outputs=output_layer)

    cnn_lstm_model.compile(loss='categorical_crossentropy',
                           optimizer=Adam(learning_rate=0.001),
                           metrics=['accuracy'])

    return cnn_lstm_model
```

Figure 3.68 The Sentiment Analysis Model Building: Building The Hybrid CNN-LSTM model Function Code

The Figure 3.68 shows the *build_cnn_lstm_model()* function to build the hybrid CNN-LSTM model based on the model architecture design. The function used the *keras* library such as *Input()* class to initialize the input layer, *Embedding()* class to initialize the word embedding layer, *Dropout()* class to initialize the dropout layer, *Conv1D()* class to initialize the one-dimension convolution layer, *MaxPooling1D()* class to initialize the one-dimension max pooling layer, *LSTM()* class to initialize the LSTM layer, and *Dense()* class to initialize the dense or basic neural network layer. All the layers will be chained each other and wrapped in as a single model using *Model()* class; then compiled with *compile()* function by defining the loss function, optimizer algorithm, and accuracy as the training metric.

```
# Build single CNN model
def build_cnn_model(embedding_matrix, max_sequence_length):
    # Input layer
    input_layer = Input(shape=(max_sequence_length,))

    # Word embedding layer
    embedding_layer = Embedding(input_dim=embedding_matrix.shape[0],
                                output_dim=embedding_matrix.shape[1],
                                weights=[embedding_matrix],
                                input_length=max_sequence_length,
                                trainable=True)(input_layer)
    dropout_layer = Dropout(rate=0.5)(embedding_layer)


    # CNN model layer
    cnn_layer = Conv1D(filters=64,
                       kernel_size=3,
                       strides=1,
                       activation='relu')(dropout_layer)
    cnn_layer = MaxPooling1D(pool_size=2)(cnn_layer)
    cnn_layer = Dropout(rate=0.5)(cnn_layer)

    flatten = Flatten()(cnn_layer)

    # Dense model layer
    dense_layer = Dense(units=128, activation='relu')(flatten)
    dropout_layer = Dropout(rate=0.5)(dense_layer)

    output_layer = Dense(units=2, activation='softmax')(dropout_layer)

    cnn_model = Model(inputs=input_layer, outputs=output_layer)

    cnn_model.compile(loss='categorical_crossentropy',
                      optimizer=Adam(learning_rate=0.001),
                      metrics=['accuracy'])

    return cnn_model
```

Figure 3.69 The Sentiment Analysis Model Building: Building The Single CNN model Function Code

The Figure 3.69 shows the *build_cnn_model()* to build the single CNN model based on the model architecture design. The function also used the *keras* library such as *Input()* class to initialize the input layer, *Embedding()* class to initialize the word embedding layer, *Dropout()* class to initialize the dropout layer, *Conv1D()* class to initialize the one-dimension convolution layer, *MaxPooling1D()* class to initialize the one-dimension max pooling layer, *Flatten()* class to initialize the Flatten layer, and *Dense()* class to initialize the dense or basic neural network layer. All the layers will be chained each other and wrapped in as a single model using *Model()* class; then compiled with *compile()* function by defining the loss function, optimizer algorithm, and accuracy as the training metric.

```
# Build single LSTM model
def build_lstm_model(embedding_matrix, max_sequence_length):
    # Input layer
    input_layer = Input(shape=(max_sequence_length,), dtype='int32')

    # Word embedding layer
    embedding_layer = Embedding(input_dim=embedding_matrix.shape[0],
                                output_dim=embedding_matrix.shape[1],
                                weights=[embedding_matrix],
                                input_length=max_sequence_length,
                                trainable=True)(input_layer)
    dropout_layer = Dropout(rate=0.5)(embedding_layer)


    # LSTM model layer
    lstm_layer = LSTM(units=128,
                      dropout=0.5,
                      return_sequences=False)(dropout_layer)

    # Dense model layer
    dense_layer = Dense(units=128, activation='relu')(lstm_layer)
    dropout_layer = Dropout(rate=0.5)(dense_layer)

    output_layer = Dense(units=2, activation='softmax')(dropout_layer)

    lstm_model = Model(inputs=input_layer, outputs=output_layer)

    lstm_model.compile(loss='categorical_crossentropy',
                       optimizer=Adam(learning_rate=0.001),
                       metrics=['accuracy'])

    return lstm_model
```

Figure 3.70 The Sentiment Analysis Model Building: Building The Single LSTM model Function Code

The Figure 3.70 shows the *build_lstm_model()* function to build the single LSTM model based on the model architecture design. The function also used the *keras* library such as *Input()* class to initialize the input layer, *Embedding()* class to initialize the word embedding layer, *Dropout()* class to initialize the dropout layer, *LSTM()* class to initialize the Flatten layer, and *Dense()* class to initialize the dense or basic neural network layer. All the layers will be chained each other and wrapped in as a single model using *Model()* class; then compiled with *compile()* function by defining the loss function, optimizer algorithm, and accuracy as the training metric.

```python
# Visualize training result
def visualize_training(model_history, file_name):
    train_acc = model_history.history['accuracy']
    val_acc = model_history.history['val_accuracy']

    train_loss = model_history.history['loss']
    val_loss = model_history.history['val_loss']

    plt.figure(figsize=(15, 8))
    plt.subplots_adjust(left=0.1,
                        bottom=0.1,
                        right=0.9,
                        top=0.9,
                        wspace=0.4,
                        hspace=0.4)

    plt.subplot(1, 2, 1)
    plt.plot(train_acc, color='tab:green')
    plt.plot(val_acc, color='tab:blue')
    plt.title('TRAINING ACCURACY')
    plt.xlabel('Number of Epoch')

    plt.ylabel('Accuracy')
    plt.legend(['Training Accuracy', 'Validation Accuracy'])

    plt.subplot(1, 2, 2)
    plt.plot(train_loss, color='tab:green')
    plt.plot(val_loss, color='tab:blue')
    plt.title('TRAINING LOSS')
    plt.xlabel('Number of Epoch')
    plt.ylabel('Loss')
    plt.legend(['Training Loss', 'Validation Loss'])

    plt.savefig(os.path.join('assets/', file_name))
    plt.show()
```

Figure 3.71 The Sentiment Analysis Model Building: Visualizing Training Result Function Code

The Figure 3.71 shows the *visualize_training()* function to visualize the training result from the sentiment analysis model. The function used the *matplotlib* library to build two-line graphs. The first line graph visualized the training accuracy and the training loss. The second line graph visualized the validation accuracy and validation loss. Finally, all the line graphs are saved as image.

```
# Building single CNN-LSTM model
with tf.device('/device:GPU:0'):
    sinovac_cnn_lstm_model = build_cnn_lstm_model(SINOVAC_EMBEDDING_MATRIX, SINOVAC_MAX_SEQUENCE)

    sinovac_cnn_lstm_model.summary()

    sinovac_cnn_lstm_history = sinovac_cnn_lstm_model.fit(x=X_sinovac_train,
                                                          y=y_sinovac_train,
                                                          batch_size=128,
                                                          epochs=500,
                                                          validation_split=0.1,
                                                          verbose=1)

# Visualize training result
visualize_training(sinovac_cnn_lstm_history, 'sinovac_training_cnn_lstm.jpg')
```

Figure 3.72 The Sentiment Analysis Model Building: Training The Hybrid CNN-LSTM Model for Sinovac Dataset Code

The Figure 3.72 shows the training process of the hybrid CNN-LSTM model for Sinovac dataset. The training process used the *build_cnn_lstm_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building single CNN model
with tf.device('/device:GPU:0'):
    sinovac_cnn_model = build_cnn_model(SINOVAC_EMBEDDING_MATRIX, SINOVAC_MAX_SEQUENCE)

    sinovac_cnn_model.summary()

    sinovac_cnn_model.compile(loss='categorical_crossentropy',
                              optimizer=Adam(learning_rate=0.001),
                              metrics=['accuracy'])

    sinovac_cnn_history = sinovac_cnn_model.fit(x=X_sinovac_train,
                                                y=y_sinovac_train,
                                                batch_size=128,
                                                epochs=500,
                                                validation_split=0.1,
                                                verbose=1)

# Visualize training result
visualize_training(sinovac_cnn_history, 'sinovac_training_cnn.jpg')
```

Figure 3.73 The Sentiment Analysis Model Building: Training The Single CNN Model for Sinovac Dataset Code

The Figure 3.73 shows the training process of the single CNN model for Sinovac dataset. The training process used the *build_cnn_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the

training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building single LSTM model
with tf.device('/device:GPU:0'):
    sinovac_lstm_model = build_lstm_model(SINOVAC_EMBEDDING_MATRIX, SINOVAC_MAX_SEQUENCE)

    sinovac_lstm_model.summary()

    sinovac_lstm_history = sinovac_lstm_model.fit(x=X_sinovac_train,
                                                  y=y_sinovac_train,
                                                  batch_size=128,
                                                  epochs=500,
                                                  validation_split=0.1,
                                                  verbose=1)

# Visualize training result
visualize_training(sinovac_lstm_history, 'sinovac_training_lstm.jpg')
```

Figure 3.74 The Sentiment Analysis Model Building: Training The Single LSTM Model for Sinovac Dataset Code

The Figure 3.74 shows the training process of the single LSTM model for Sinovac dataset. The training process used the *build_lstm_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building hybrid CNN-LSTM model
with tf.device('/device:GPU:0'):
    astrazeneca_cnn_lstm_model = build_cnn_lstm_model(ASTRAZENECA_EMBEDDING_MATRIX, ASTRAZENECA_MAX_SEQUENCE)

    astrazeneca_cnn_lstm_model.summary()

    astrazeneca_cnn_lstm_model_history = astrazeneca_cnn_lstm_model.fit(x=X_astrazeneca_train,
                                                                        y=y_astrazeneca_train,
                                                                        batch_size=128,
                                                                        epochs=500,
                                                                        validation_split=0.1,
                                                                        verbose=1)

# Visualize training result
visualize_training(astrazeneca_cnn_lstm_model_history, 'astrazeneca_training_cnn_lstm.jpg')
```

Figure 3.75 The Sentiment Analysis Model Building: Training The Hybrid CNN-LSTM Model for AstraZeneca Dataset Code

The Figure 3.75 shows the training process of the hybrid CNN-LSTM model for AstraZeneca dataset. The training process used the *build_cnn_lstm_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while

training, and verbose. Finally, the training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building single CNN model
with tf.device('/device:GPU:0'):
    astrazeneca_cnn_model = build_cnn_model(ASTRAZENECA_EMBEDDING_MATRIX, ASTRAZENECA_MAX_SEQUENCE)

    astrazeneca_cnn_model.summary()

    astrazeneca_cnn_history = astrazeneca_cnn_model.fit(x=X_astrazeneca_train,
                                                        y=y_astrazeneca_train,
                                                        batch_size=128,
                                                        epochs=500,
                                                        validation_split=0.1,
                                                        verbose=1)

# Visualize training result
visualize_training(astrazeneca_cnn_history, 'astrazeneca_training_cnn.jpg')
```

Figure 3.76 The Sentiment Analysis Model Building: Training The Single CNN Model for AstraZeneca Dataset Code

The Figure 3.76 shows the training process of the single CNN model for AstraZeneca dataset. The training process used the *build_cnn_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building single LSTM model
with tf.device('/device:GPU:0'):
    astrazeneca_lstm_model = build_lstm_model(ASTRAZENECA_EMBEDDING_MATRIX, ASTRAZENECA_MAX_SEQUENCE)

    astrazeneca_lstm_model.summary()

    astrazeneca_lstm_history = astrazeneca_lstm_model.fit(x=X_astrazeneca_train,
                                                          y=y_astrazeneca_train,
                                                          batch_size=128,
                                                          epochs=500,
                                                          validation_split=0.1,
                                                          verbose=1)

# Visualize training result
visualize_training(astrazeneca_lstm_history, 'astrazeneca_training_lstm.jpg')
```

Figure 3.77 The Sentiment Analysis Model Building: Training The Single LSTM Model for AstraZeneca Dataset Code

The Figure 3.77 shows the training process of the single LSTM model for AstraZeneca dataset. The training process used the *build_lstm_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the

training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building hybrid CNN-LSTM model
with tf.device('/device:GPU:0'):
    pfizer_cnn_lstm_model = build_cnn_lstm_model(PFIZER_EMBEDDING_MATRIX, PFIZER_MAX_SEQUENCE)

    pfizer_cnn_lstm_model.summary()

    pfizer_cnn_lstm_model_history = pfizer_cnn_lstm_model.fit(x=X_pfizer_train,
                                                              y=y_pfizer_train,
                                                              batch_size=128,
                                                              epochs=500,
                                                              validation_split=0.1,
                                                              verbose=1)

# Visualize training result
visualize_training(pfizer_cnn_lstm_model_history, 'pfizer_training_cnn_lstm.jpg')
```

Figure 3.78 The Sentiment Analysis Model Building: Training The Hybrid CNN-LSTM Model for Pfizer Dataset Code

The Figure 3.78 shows the training process of the hybrid CNN-LSTM model for Pfizer dataset. The training process used the *build_cnn_lstm_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building single CNN model
with tf.device('/device:GPU:0'):
    pfizer_cnn_model = build_cnn_model(PFIZER_EMBEDDING_MATRIX, PFIZER_MAX_SEQUENCE)

    pfizer_cnn_model.summary()

    pfizer_cnn_model_history = pfizer_cnn_model.fit(x=X_pfizer_train,
                                                    y=y_pfizer_train,
                                                    batch_size=128,
                                                    epochs=500,
                                                    validation_split=0.1,
                                                    verbose=1)

# Visualize training result
visualize_training(pfizer_cnn_model_history, 'pfizer_training_cnn.jpg')
```

Figure 3.79 The Sentiment Analysis Model Building: Training The Single CNN Model for Pfizer Dataset Code

The Figure 3.79 shows the training process of the single CNN model for Pfizer dataset. The training process used the *build_cnn_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the

training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building single LSTM model
with tf.device('/device:GPU:0'):
    pfizer_lstm_model = build_lstm_model(PFIZER_EMBEDDING_MATRIX, PFIZER_MAX_SEQUENCE)

    pfizer_lstm_model.summary()

    pfizer_lstm_history = pfizer_lstm_model.fit(x=X_pfizer_train,
                                                y=y_pfizer_train,
                                                batch_size=128,
                                                epochs=500,
                                                validation_split=0.1,
                                                verbose=1)

# Visualize training result
visualize_training(pfizer_lstm_history, 'pfizer_training_lstm.jpg')
```

Figure 3.80 The Sentiment Analysis Model Building: Training The Single LSTM Model for Pfizer Dataset Code

The Figure 3.80 shows the training process of the single LSTM model for Pfizer dataset. The training process used the *build_lstm_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building hybrid CNN-LSTM model
with tf.device('/device:GPU:0'):
    moderna_cnn_lstm_model = build_cnn_lstm_model(MODERNA_EMBEDDING_MATRIX, MODERNA_MAX_SEQUENCE)

    moderna_cnn_lstm_model.summary()

    moderna_cnn_lstm_model_history = moderna_cnn_lstm_model.fit(x=X_moderna_train,
                                                                y=y_moderna_train,
                                                                batch_size=128,
                                                                epochs=500,
                                                                validation_split=0.1,
                                                                verbose=1)

# Visualize training result
visualize_training(moderna_cnn_lstm_model_history, 'moderna_training_cnn_lstm.jpg')
```

Figure 3.81 The Sentiment Analysis Model Building: Training The Hybrid CNN-LSTM Model for Moderna Dataset Code

The Figure 3.81 shows the training process of the hybrid CNN-LSTM model for Moderna dataset. The training process used the *build_cnn_lstm_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally,

the training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building single CNN model
with tf.device('/device:GPU:0'):
    moderna_cnn_model = build_cnn_model(MODERNA_EMBEDDING_MATRIX, MODERNA_MAX_SEQUENCE)

    moderna_cnn_model.summary()

    moderna_cnn_model_history = moderna_cnn_model.fit(x=X_moderna_train,
                                                      y=y_moderna_train,
                                                      batch_size=128,
                                                      epochs=500,
                                                      validation_split=0.1,
                                                      verbose=1)

# Visualize training result
visualize_training(moderna_cnn_model_history, 'moderna_training_cnn.jpg')
```

Figure 3.82 The Sentiment Analysis Model Building: Training The Single CNN Model for Moderna Dataset Code

The Figure 3.82 shows the training process of the single CNN model for Moderna dataset. The training process used the *build_cnn_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the training result will be plotted with *visualize_training()* function by passing the training history.

```
# Building single LSTM model
with tf.device('/device:GPU:0'):
    moderna_lstm_model = build_lstm_model(MODERNA_EMBEDDING_MATRIX, MODERNA_MAX_SEQUENCE)

    moderna_lstm_model.summary()

    moderna_lstm_history = moderna_lstm_model.fit(x=X_moderna_train,
                                                  y=y_moderna_train,
                                                  batch_size=128,
                                                  epochs=500,
                                                  validation_split=0.1,
                                                  verbose=1)

# Visualize training result
visualize_training(moderna_lstm_history, 'moderna_training_lstm.jpg')
```

Figure 3.83 The Sentiment Analysis Model Building: Training The Single LSTM Model for Moderna Dataset Code

The Figure 3.83 shows the training process of the single LSTM model for Moderna dataset. The training process used the *build_lstm_model()* function to initialize the model and *fit()* function from *keras* library to train the model by passing several training parameters such as feature data, label data, batch size, number of epoch, portion of validation while training, and verbose. Finally, the

training result will be plotted with *visualize_training()* function by passing the training history.

```
def visualize_confussion_matrix(y_true, y_pred, title):
    classes = ['Negative', 'Positive']
    cm = confusion_matrix(y_true, y_pred)

    cmap = plt.cm.Blues
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], 'd'), horizontalalignment='center', color='white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

def evaluate_model(y_true, y_pred):
    target_names = ['Negative', 'Positive']
    classification_eval = metrics.classification_report(y_true, y_pred, target_names=target_names)
    print(classification_eval)
```

Figure 3.84 The Sentiment Analysis Model Building: Visualizing Testing and Evaluation Result Function Code Part 1

The Figure 3.84 shows the functions to visualize the testing and evaluation result from the sentiment analysis model. The *visualize_confusion_matrix()* function used the *matplotlib* library to visualize the confusion matrix table. The *evaluate_mode()* function used *classification_report()* function from *scikit-learn* library to calculate the precision, recall, accuracy, and f1-score value respect to the predicted label and the true label.

```
def visualize_metrics(y_pred_cnn_lstm,
                      y_pred_cnn,
                      y_pred_lstm,
                      y_true,
                      file_name):

    accuracies = [accuracy_score(y_true, y_pred_cnn_lstm),
                  accuracy_score(y_true, y_pred_cnn),
                  accuracy_score(y_true, y_pred_lstm),]
    precissions = [precision_score(y_true, y_pred_cnn_lstm),
                   precision_score(y_true, y_pred_cnn),
                   precision_score(y_true, y_pred_lstm)]
    recalls = [recall_score(y_true, y_pred_cnn_lstm),
               recall_score(y_true, y_pred_cnn),
               recall_score(y_true, y_pred_lstm)]
    f1_scores = [f1_score(y_true, y_pred_cnn_lstm),
                 f1_score(y_true, y_pred_cnn),
                 f1_score(y_true, y_pred_lstm)]
    models = ['CNN-LSTM', 'CNN', 'LSTM']

    df_metrics = pd.DataFrame({'Model': models,
                               'Accuracy': accuracies,
                               'Precision': precissions,
                               'Recall': recalls,
                               'F1-Score': f1_scores})

    df_metrics.plot(x="Model",
                    y=["Accuracy", "Precision", "Recall", "F1-Score"],
                    kind="bar",
                    figsize=(15, 10))

    plt.savefig(os.path.join('assets/', file_name))
    plt.show()
```

Figure 3.85 The Sentiment Analysis Model Building: Visualizing Testing and Evaluation Result Function Code Part 2

The Figure 3.85 shows the functions to visualize the precision, recall, accuracy, and f1-score for each trained sentiment analysis models. The *visualize_metrics ()* function used the *matplotlib* library to visualize the precision, recall, accuracy, and f1-score in multi-bar graph; and used *accuracy_score()* function, *precision_score()* function, *recall_score()* function, and *f1_score()* function from scikit-learn library.

```
### **Testing & Evaluation:  CNN-LSTM**
"""

with tf.device('/device:GPU:0'):
    result = sinovac_cnn_lstm_model.evaluate(X_sinovac_test, y_sinovac_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_sinovac_pred_raw = sinovac_cnn_lstm_model.predict(X_sinovac_test)
    y_sinovac_pred_cnn_lstm = np.argmax(y_sinovac_pred_raw, axis=1)
    y_sinovac_true = np.argmax(y_sinovac_test, axis=1)

    visualize_confussion_matrix(y_sinovac_pred_cnn_lstm,
                                y_sinovac_true,
                                'CNN-LSTM Model Confussion Matrix',
                                'sinovac_cnn_lstm_confussion.jpg')

    evaluate_model(y_sinovac_pred_cnn_lstm, y_sinovac_true)

sinovac_cnn_lstm_model.save('model/sinovac_cnn_lstm_model.h5')
```

Figure 3.86 The Sentiment Analysis Model Building: Sinovac Dataset Hybrid CNN-LSTM Model Testing and Evaluation Code

The Figure 3.86 shows the testing and evaluation of the hybrid CNN-LSTM model for Sinovac dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated and visualized with the *visualized_confusion_matrix()* function. Finally, the trained hybrid CNN-LSTM model for Sinovac dataset is saved with *save()* function from *keras* library.

```
"""### **Testing & Evaluation:  CNN**"""

with tf.device('/device:GPU:0'):
    result = sinovac_cnn_model.evaluate(X_sinovac_test, y_sinovac_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_sinovac_pred_raw = sinovac_cnn_model.predict(X_sinovac_test)
    y_sinovac_pred_cnn = np.argmax(y_sinovac_pred_raw, axis=1)
    y_sinovac_true = np.argmax(y_sinovac_test, axis=1)

    visualize_confussion_matrix(y_sinovac_pred_cnn,
                                y_sinovac_true,
                                'CNN Model Confussion Matrix',
                                'sinovac_cnn_confussion.jpg')

    evaluate_model(y_sinovac_pred_cnn, y_sinovac_true)

sinovac_cnn_model.save('model/sinovac_cnn_model.h5')
```

Figure 3.87 The Sentiment Analysis Model Building: Sinovac Dataset Single CNN Model Testing and Evaluation Code

The Figure 3.87 shows the testing and evaluation of the single CNN model for Sinovac dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated

and visualized with the *visualized_confusion_matrix()* function. Finally, the trained single CNN model for Sinovac dataset is saved with *save()* function from *keras* library.

```
"""### **Testing & Evaluation: LSTM**"""

with tf.device('/device:GPU:0'):
    result = sinovac_lstm_model.evaluate(X_sinovac_test, y_sinovac_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_sinovac_pred_raw = sinovac_lstm_model.predict(X_sinovac_test)
    y_sinovac_pred_lstm = np.argmax(y_sinovac_pred_raw, axis=1)
    y_sinovac_true = np.argmax(y_sinovac_test, axis=1)

    visualize_confussion_matrix(y_sinovac_pred_lstm,
                                y_sinovac_true,
                                'LSTM Model Confussion Matrix',
                                'sinovac_lstm_confussion.jpg')

    evaluate_model(y_sinovac_pred_lstm, y_sinovac_true)

sinovac_lstm_model.save('model/sinovac_lstm_model.h5')
```

Figure 3.88 The Sentiment Analysis Model Building: Sinovac Dataset Single LSTM Model Testing and Evaluation Code

The Figure 3.88 shows the testing and evaluation of the single LSTM model for Sinovac dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated and visualized with the *visualized_confusion_matrix ()* function. Finally, the trained single LSTM model for Sinovac dataset is saved with *save()* function from *keras* library.

```
visualize_metrics(y_sinovac_pred_cnn_lstm,
                  y_sinovac_pred_cnn,
                  y_sinovac_pred_lstm,
                  y_sinovac_true,
                  'sinovac_metrics.jpg')
```

Figure 3.89 The Sentiment Analysis Model Building: Visualize Precision, Recall, Accuracy, and F1-Score All Sinovac Dataset Models Code

The Figure 3.89 shows the visualization of precision, recall, accuracy, and f1-score metrics for all trained Sinovac dataset sentiment analysis models. The visualization used *visualize_metrics()* function by passing each predicted label from the hybrid CNN-LSTM, the single CNN model, the single LSTM model for Sinovac dataset; true label; and file name where the visualized graph will be saved.

```
### **Testing & Evaluation:  CNN-LSTM**
"""

with tf.device('/device:GPU:0'):
    result = astrazeneca_cnn_lstm_model.evaluate(X_astrazeneca_test, y_astrazeneca_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_astrazeneca_pred_raw = astrazeneca_cnn_lstm_model.predict(X_astrazeneca_test)
    y_astrazeneca_pred_cnn_lstm = np.argmax(y_astrazeneca_pred_raw, axis=1)
    y_astrazeneca_true = np.argmax(y_astrazeneca_test, axis=1)

    visualize_confussion_matrix(y_astrazeneca_pred_cnn_lstm,
                                y_astrazeneca_true,
                                'CNN-LSTM Model Confussion Matrix',
                                'astrazeneca_cnn_lstm_confussion.jpg')

    evaluate_model(y_astrazeneca_pred_cnn_lstm, y_astrazeneca_true)

astrazeneca_cnn_lstm_model.save('model/astrazeneca_cnn_lstm_model.h5')
```

Figure 3.90 The Sentiment Analysis Model Building: AstraZeneca Dataset Hybrid CNN-LSTM Model Testing and Evaluation Code

The Figure 3.90 shows the testing and evaluation of the hybrid CNN-LSTM model for AstraZeneca dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated and visualized with the *visualized_confusion_matrix()* function. Finally, the trained hybrid CNN-LSTM model for AstraZeneca dataset is saved with *save()* function from *keras* library.

```
"""### **Testing & Evaluation:  CNN**"""

with tf.device('/device:GPU:0'):
    result = astrazeneca_cnn_model.evaluate(X_astrazeneca_test, y_astrazeneca_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_astrazeneca_pred_raw = astrazeneca_cnn_model.predict(X_astrazeneca_test)
    y_astrazeneca_pred_cnn = np.argmax(y_astrazeneca_pred_raw, axis=1)
    y_astrazeneca_true = np.argmax(y_astrazeneca_test, axis=1)

    visualize_confussion_matrix(y_astrazeneca_pred_cnn,
                                y_astrazeneca_true,
                                'CNN Model Confussion Matrix',
                                'astrazeneca_cnn_confusion.jpg')

    evaluate_model(y_astrazeneca_pred_cnn, y_astrazeneca_true)

astrazeneca_cnn_model.save('model/astrazeneca_cnn_model.h5')
```

Figure 3.91 The Sentiment Analysis Model Building: AstraZeneca Dataset Single CNN Model Testing and Evaluation Code

The Figure 3.91 shows the testing and evaluation of the single CNN model for AstraZeneca dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated and visualized with the *visualized_confusion_matrix()* function. Finally, the trained single CNN model for AstraZeneca dataset is saved with *save()* function from *keras* library.

```
"""### **Testing & Evaluation: LSTM**"""

with tf.device('/device:GPU:0'):
    result = astrazeneca_lstm_model.evaluate(X_astrazeneca_test, y_astrazeneca_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_astrazeneca_pred_raw = astrazeneca_lstm_model.predict(X_astrazeneca_test)
    y_astrazeneca_pred_lstm = np.argmax(y_astrazeneca_pred_raw, axis=1)
    y_astrazeneca_true = np.argmax(y_astrazeneca_test, axis=1)

    visualize_confussion_matrix(y_astrazeneca_pred_lstm,
                                y_astrazeneca_true,
                                'LSTM Model Confussion Matrix',
                                'astrazeneca_lstm_confusion.jpg')

    evaluate_model(y_astrazeneca_pred_lstm, y_astrazeneca_true)

astrazeneca_lstm_model.save('model/astrazeneca_lstm_model.h5')
```

Figure 3.92 The Sentiment Analysis Model Building: AstraZeneca Dataset Single LSTM Model Testing and Evaluation Code

The Figure 3.92 shows the testing and evaluation of the single LSTM model for AstraZeneca dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated and visualized with the *visualized_confusion_matrix()* function. Finally, the trained single LSTM model for AstraZeneca dataset is saved with *save()* function from *keras* library.

```
visualize_metrics(y_astrazeneca_pred_cnn_lstm,
                  y_astrazeneca_pred_cnn,
                  y_astrazeneca_pred_lstm,
                  y_astrazeneca_true,
                  'astrazeneca_metrics.jpg')
```

Figure 3.93 The Sentiment Analysis Model Building: Visualize Precision, Recall, Accuracy, and F1-Score All Trained AstraZeneca Dataset Models Code

The Figure 3.93 shows the visualization of precision, recall, accuracy, and f1-score metrics for all trained Sinovac dataset sentiment analysis models. The visualization used *visualize_metrics()* function by passing each predicted label from the hybrid CNN-LSTM, the single CNN model, the single LSTM model for AstraZeneca dataset; true label; and file name where the visualized graph will be saved.

```
### **Testing & Evaluation:  CNN-LSTM**
"""

with tf.device('/device:GPU:0'):
    result = pfizer_cnn_lstm_model.evaluate(X_pfizer_test, y_pfizer_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_pfizer_pred_raw = pfizer_cnn_lstm_model.predict(X_pfizer_test)
    y_pfizer_pred_cnn_lstm = np.argmax(y_pfizer_pred_raw, axis=1)
    y_pfizer_true = np.argmax(y_pfizer_test, axis=1)

    visualize_confussion_matrix(y_pfizer_pred_cnn_lstm,
                                y_pfizer_true,
                                'CNN-LSTM Model Confussion Matrix',
                                'pfizer_cnn_lstm_confusion.jpg')

    evaluate_model(y_pfizer_pred_cnn_lstm, y_pfizer_true)

pfizer_cnn_lstm_model.save('model/pfizer_cnn_lstm_model.h5')
```

Figure 3.94 The Sentiment Analysis Model Building: Pfizer Dataset Hybrid CNN-LSTM Model Testing and Evaluation Code

The Figure 3.94 shows the testing and evaluation of the hybrid CNN-LSTM model for Pfizer dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated and visualized with the *visualized_confusion_matrix()* function. Finally, the trained hybrid CNN-LSTM model for Pfizer dataset is saved with *save()* function from *keras* library.

```
"""### **Testing & Evaluation:  CNN**"""

with tf.device('/device:GPU:0'):
    result = pfizer_cnn_model.evaluate(X_pfizer_test, y_pfizer_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_pfizer_pred_raw = pfizer_cnn_model.predict(X_pfizer_test)
    y_pfizer_pred_cnn = np.argmax(y_pfizer_pred_raw, axis=1)
    y_pfizer_true = np.argmax(y_pfizer_test, axis=1)

    visualize_confussion_matrix(y_pfizer_pred_cnn,
                                y_pfizer_true,
                                'CNN Model Confussion Matrix',
                                'pfizer_cnn_confusion.jpg')

    evaluate_model(y_pfizer_pred_cnn, y_pfizer_true)

pfizer_cnn_model.save('model/pfizer_cnn_model.h5')
```

Figure 3.95 The Sentiment Analysis Model Building: Pfizer Dataset Single CNN Model Testing and Evaluation Code

The Figure 3.95 shows the testing and evaluation of the single CNN model for Pfizer dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated

and visualized with the *visualized_confusion_matrix()* function. Finally, the trained single CNN model for Pfizer dataset is saved with *save()* function from *keras* library.

```
"""### **Testing & Evaluation: LSTM**"""

with tf.device('/device:GPU:0'):
    result = pfizer_lstm_model.evaluate(X_pfizer_test, y_pfizer_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_pfizer_pred_raw = pfizer_lstm_model.predict(X_pfizer_test)
    y_pfizer_pred_lstm = np.argmax(y_pfizer_pred_raw, axis=1)
    y_pfizer_true = np.argmax(y_pfizer_test, axis=1)

    visualize_confussion_matrix(y_pfizer_pred_lstm,
                                y_pfizer_true,
                                'LSTM Model Confussion Matrix',
                                'pfizer_lstm_confusion.jpg')

    evaluate_model(y_pfizer_pred_lstm, y_pfizer_true)

pfizer_lstm_model.save('model/pfizer_lstm_model.h5')
```

Figure 3.96 The Sentiment Analysis Model Building: Pfizer Dataset Single LSTM Model Testing and Evaluation Code

The Figure 3.96 shows the testing and evaluation of the single LSTM model for Pfizer dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated and visualized with the *visualized_confusion_matrix()* function. Finally, the trained single LSTM model for Pfizer dataset is saved with *save()* function from *keras* library.

```
visualize_metrics(y_pfizer_pred_cnn_lstm,
                  y_pfizer_pred_cnn,
                  y_pfizer_pred_lstm,
                  y_pfizer_true,
                  'pfizer_metrics.jpg')
```

Figure 3.97 The Sentiment Analysis Model Building: Visualize Precision, Recall, Accuracy, and F1-Score All Trained Pfizer Dataset Models Code

The Figure 3.97 shows the visualization of precision, recall, accuracy, and f1-score metrics for all trained Pfizer dataset sentiment analysis models. The visualization used *visualize_metrics()* function by passing each predicted label from the hybrid CNN-LSTM, the single CNN model, the single LSTM model for Pfizer dataset; true label; and file name where the visualized graph will be saved.

```
### **Testing & Evaluation:  CNN-LSTM**
"""

with tf.device('/device:GPU:0'):
    result = moderna_cnn_lstm_model.evaluate(X_moderna_test, y_moderna_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_moderna_pred_raw = moderna_cnn_lstm_model.predict(X_moderna_test)
    y_moderna_pred_cnn_lstm = np.argmax(y_moderna_pred_raw, axis=1)
    y_moderna_true = np.argmax(y_moderna_test, axis=1)

    visualize_confussion_matrix(y_moderna_pred_cnn_lstm,
                                y_moderna_true,
                                'CNN-LSTM Model Confussion Matrix',
                                'moderna_cnn_lstm_confusion.jpg')

    evaluate_model(y_moderna_pred_cnn_lstm, y_moderna_true)

moderna_cnn_lstm_model.save('model/moderna_cnn_lstm_model.h5')
```

Figure 3.98 The Sentiment Analysis Model Building: Moderna Dataset Hybrid CNN-LSTM Model Testing and Evaluation Code

The Figure 3.98 shows the testing and evaluation of the hybrid CNN-LSTM model for Moderna dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated and visualized with the *visualized_confusion_matrix()* function. Finally, the trained hybrid CNN-LSTM model for Moderna dataset is saved with *save()* function from *keras* library.

```
"""### **Testing & Evaluation:  CNN**"""

with tf.device('/device:GPU:0'):
    result = moderna_cnn_model.evaluate(X_moderna_test, y_moderna_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_moderna_pred_raw = moderna_cnn_model.predict(X_moderna_test)
    y_moderna_pred_cnn = np.argmax(y_moderna_pred_raw, axis=1)
    y_moderna_true = np.argmax(y_moderna_test, axis=1)

    visualize_confussion_matrix(y_moderna_pred_cnn,
                                y_moderna_true,
                                'CNN Model Confussion Matrix',
                                'moderna_cnn_confusion.jpg')

    evaluate_model(y_moderna_pred_cnn, y_moderna_true)

moderna_cnn_model.save('model/moderna_cnn_model.h5')
```

Figure 3.99 The Sentiment Analysis Model Building: Moderna Dataset Single CNN Model Testing and Evaluation Code

The Figure 3.99 shows the testing and evaluation of the single CNN model for Moderna dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated

and visualized with the *visualized_confusion_matrix()* function. Finally, the trained single CNN model for Moderna dataset is saved with *save()* function from *keras* library.

```
"""### **Testing & Evaluation: LSTM**"""

with tf.device('/device:GPU:0'):
    result = moderna_lstm_model.evaluate(X_moderna_test, y_moderna_test, verbose=1)

print('Accuracy test : {:.2f}%'.format(result[1]*100))
print('Loss test : {:.3}'.format(result[0]))

with tf.device('/device:GPU:0'):
    y_moderna_pred_raw = moderna_lstm_model.predict(X_moderna_test)
    y_moderna_pred_lstm = np.argmax(y_moderna_pred_raw, axis=1)
    y_moderna_true = np.argmax(y_moderna_test, axis=1)

    visualize_confussion_matrix(y_moderna_pred_lstm,
                                y_moderna_true,
                                'LSTM Model Confussion Matrix',
                                'moderna_lstm_confusion.jpg')

    evaluate_model(y_moderna_pred_lstm, y_moderna_true)

moderna_lstm_model.save('model/moderna_lstm_model.h5')
```

Figure 3.100 The Sentiment Analysis Model Building: Moderna Dataset Single LSTM Model Testing and Evaluation Code

The Figure 3.100 shows the testing and evaluation of the single LSTM model for Moderna dataset. First, the model is tested with *evaluate()* function from *keras* library to retrieve the test accuracy and the test loss. Second, the model is evaluated and visualized with the *visualized_confusion_matrix()* function. Finally, the trained single LSTM model for Moderna dataset is saved with *save()* function from *keras* library.

```
visualize_metrics(y_moderna_pred_cnn_lstm,
                  y_moderna_pred_cnn,
                  y_moderna_pred_lstm,
                  y_moderna_true,
                  'moderna_metrics.jpg')
```

Figure 3.101 The Sentiment Analysis Model Building: Visualize Precision, Recall, Accuracy, and F1-Score All Trained Moderna Dataset Models Code

The Figure 3.101 shows the visualization of precision, recall, accuracy, and f1-score metrics for all trained Moderna dataset sentiment analysis models. The visualization used *visualize_metrics()* function by passing each predicted label from

the hybrid CNN-LSTM, the single CNN model, the single LSTM model for Moderna dataset; true label; and file name where the visualized graph will be saved.

3.4.4 Testing

The testing of the first increment is to test and evaluate the sentiment analysis models performance with the unseen data. The confusion matrix is used to evaluate model through TP (True Positive), FN (False Negative), FP (False Positive), and TN (True Negative) and produce evaluation metrics such as recall, precision, and F1- score. The confusion matrix will be map into correspond sentiment labels that are positive sentiment and negative sentiment.

Table 3.3 The Sentiment Analysis Model Confusion Matrix Evaluation Example

Predicted	***Actual***	
	Negative	***Positive***
Negative	*TN (True Negative)*	*FP (False Positive)*
Positive	*FN (False Negative)*	*TP (True Positive)*

Table 3. 4 The Sentiment Analysis Model Recall, Precision, and F1-Score Metrics Example

Model	***Label***	***Metrics***		
		Recall	***Precision***	***F1-Score***
Hybrid CNN-LSTM	*Negative*			
	Positive			
Average				
Single CNN	*Negative*			
	Positive			
Average				
Single LSTM	*Negative*			
	Positive			
Average				

3.5 Incremental 2

In the second increment process, I defined the requirements needed to build the sentiment analysis web application that includes the functions, the hardware,

and the libraries. Then, I designed the sentiment analysis application function flows and the user interface design. After that, I implemented the process according to the design through coding and testing the sentiment analysis model web application with black-box testing. The output of the second increment is the sentiment analysis web application.

3.5.1 Requirement Analysis

The requirement analysis of the second increment covered several aspects: the dataset, the functions, the pages, the hardware, and the libraries. The dataset that will be used is the implementation dataset that already selected from the first increment. Again, the experiment dataset consisted from 22 March 2022 until 31 March 2022 period.

The main functions for the sentiment analysis web application are uploading the implementation data, data pre-processing, and displaying sentiment analysis result. The uploading implementation dataset will be used file input field and selection input field, where the file input field is used to upload the implementation dataset and the selection input field is used to choose a sentiment analysis model. The data pre-processing has same processes from the first increment such as case folding or lowercasing all the words in tweets, removing non-ASCII characters, HTML (HyperText Markup Language) tags, removing retweets notation "RT", removing URLs, removing mentions that contain username (notated with "@"), removing hashtag (notated with "#"), changing the written numbers in words into actual numbers, removing numbers, removing punctuations, expanding contractions, replacing negations, removing stopwords, lemmatizing, and tokenizing. Then, All the four uploaded and pre-processed implementation datasets will be implemented with the chosen sentiment analysis model such as the hybrid CNN-LSTM model, the single CNN model, or the single LSTM model. Finally, the classified four implementation dataset results will be displayed as sentiment analysis summary.

The sentiment analysis web application pages will be consisted of three main pages such as home page, about page, and sentiment analysis page. The home

page will be contained an overview definition that briefly explained the topics to the user about the domain. The about page will be explained of all the technologies and tools behind the system. The sentiment analysis page will be contained the main functions of the sentiment analysis web application; it consisted of six subpages. The first subpage is the sentiment analysis welcome page that gives the instruction to the user about how to use the system. The second subpage is the Sinovac sentiment analysis page that allows the user to upload the Sinovac implementation dataset, choose a sentiment analysis model, and process it. After the Sinovac implementation dataset is processed, the system will show the Sinovac sentiment analysis result page to show the sentiment analysis summary to the user. The third subpage is the AstraZeneca sentiment analysis page that allows the user to upload the Sinovac implementation dataset, choose a sentiment analysis model, and process it. After the AstraZeneca implementation dataset is processed, the system will show the AstraZeneca sentiment analysis result page to show the sentiment analysis summary to the user. The fourth subpage is the Pfizer sentiment analysis page that allows the user to upload the Pfizer implementation dataset, choose a sentiment analysis model, and process it. After the Pfizer implementation dataset is processed, the system will show the Pfizer sentiment analysis result page to show the sentiment analysis summary to the user. The fifth subpage is the Moderna sentiment analysis page that allows the user to upload the Moderna implementation dataset, choose a sentiment analysis model, and process it. After the Moderna implementation dataset is processed, the system will show the Moderna sentiment analysis result page to show the sentiment analysis summary to the user. The sixth last page is the back to the home page that will reset all the sentiment analysis result for the user to quit the main function of the system.

Table 3.5 The Hardware Requirement

No.	***Type***	***Name***
1	*Operating System*	*Windows 10*
2	*Processor*	*Intel i7 9400f 2.90 GHz 6 cores*
3	*Memory*	*16GB*
4	*GPU*	*NVIDIA GeForce GTX 1050Ti*

Table 3.6 The Software Requirement

No.	***Name***	***Type***	***Version***
1	*Visual Studio Code*	*Integrated Development Environment (IDE)*	*-*
2	*Python*	*Programming Language*	*3.6.9*
3	*Pipenv*	*Python Library*	*2022.4.8*
4	*pandas*	*Python Library*	*1.4.2*
5	*regex*	*Python Library*	*2022.4.2*
6	*string*	*Python Library*	*-*
7	*NLTK (Natural Language Tool Kit)*	*Python Library*	*3.7*
8	*unicodedata2*	*Python Library*	*14.0.0*
9	*BeautifulSoap*	*Python Library*	*4.11.1*
10	*pycontractions*	*Python Library*	*2.0.1*
11	*word2number*	*Python Library*	*1.1*
12	*tensorflow*	*Python Library*	*2.8.0*
13	*keras*	*Python Library*	*2.8.0*
14	*flask*	*Python Library*	*2.1.2*
15	*flask-wtf*	*Python Library*	*1.0.1*
16	*plotly*	*Python Library*	*5.7.0*

The Table 3.6 defines all the requirement libraries that will be used in this work. I used *Visual Studio Code* for the *Python* and *flask* IDE, which is, a general code editor. The *Python* programming language will be used throughout the second increment to produce the sentiment analysis web application. The *pipenv* library will be used to create a *Python* virtual environment. The *pandas* library will be used for reading, filtering, concatenating, and grouping the implementation dataset as dataframe. The *regex* library will be used for performing regular expression on removing retweets, removing hashtags, removing mentions, and removing URLs. The *string* library will be used for removing punctuations on the tweet cleaning process. The *NLTK* library will be used for tokenizing texts, lemmatizing words, removing stopwords, and replacing negation with antonyms. The *unicodedata2*

library will be used for removing non-ASCII characters. The *BeautifulSoap* library will be used for removing HTML tags. The *pycontractions* library will be used for expanding contractions. The *word2number* library will be used for replacing written numbers in words to actual numbers. The *tensorflow* and *keras* library will be used for loading the trained sentiment analysis models from the first increment such as the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model. The *flask* library will be used as a Python web development framework. The *flask-wtf* (what the form) library will be used as the *flask* form handling for form processing and form validation. The *plotly* library will be used to visualize the sentiment analysis result with bar plot and pie plot.

3.5.2 Design

The design of the second increment divided into two designs that are the functional flow design and user interface design. The functional flow design is designed with flowchart, depicted all the function flow processes in the sentiment analysis web application. The sentiment analysis web application user interface design will be designed with wireframe design.

3.5.2.1 Function Flow Design

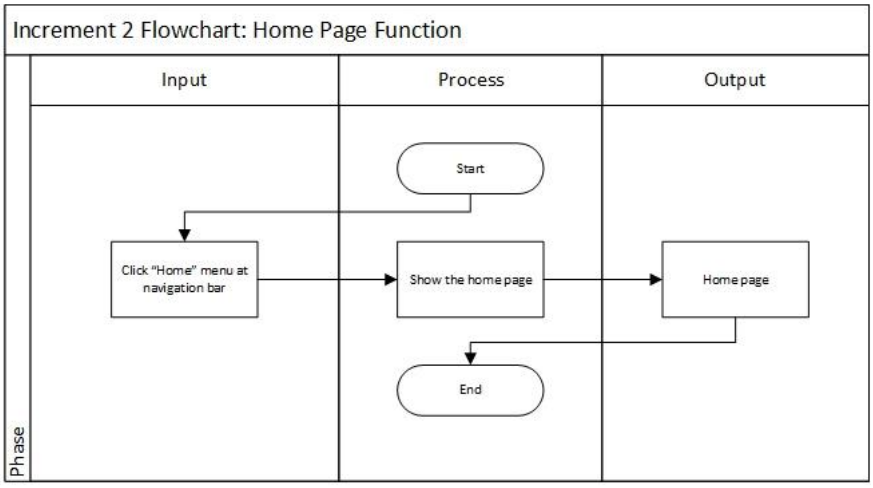


Figure 3.102 The Sentiment Analysis Web Application Home Page Function Flowchart

The Figure 3.102 shows the sentiment analysis web application function flow for accessing the home page. First, the user clicks the “Home” button at the

navigation bar at top of the page. Second, the system renders the home view through defined route. Finally, the home page is displayed on the user interface.

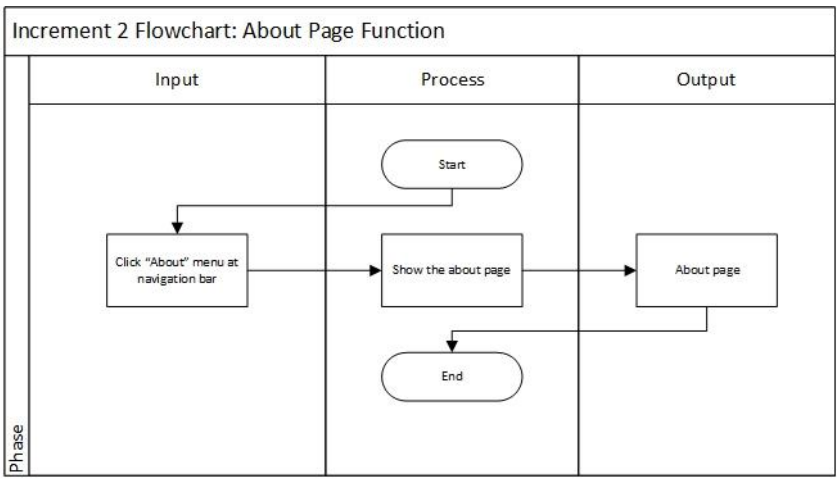


Figure 3.103 The Sentiment Analysis Web Application About Page Function Flowchart

The Figure 3.103 shows the sentiment analysis web application function flow for accessing the about page. First, the user clicks the "About" button at the navigation bar at top of the page. Second, the system renders the about view based on defined route. Finally, the about page is displayed on the user interface.

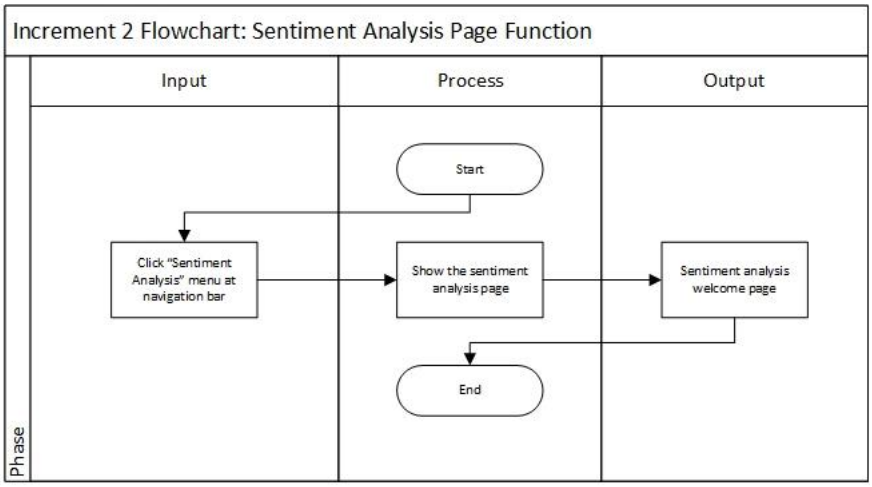


Figure 3.104 The Sentiment Analysis Web Application Sentiment Analysis Page Function Flowchart

The Figure 3.104 shows the sentiment analysis web application function flow for accessing the sentiment analysis page. First, the user clicks the "Sentiment Analysis" button at the navigation bar at top of the page. Second, the system renders

the sentiment analysis view based on defined route. Finally, the sentiment analysis page is displayed on the user interface.

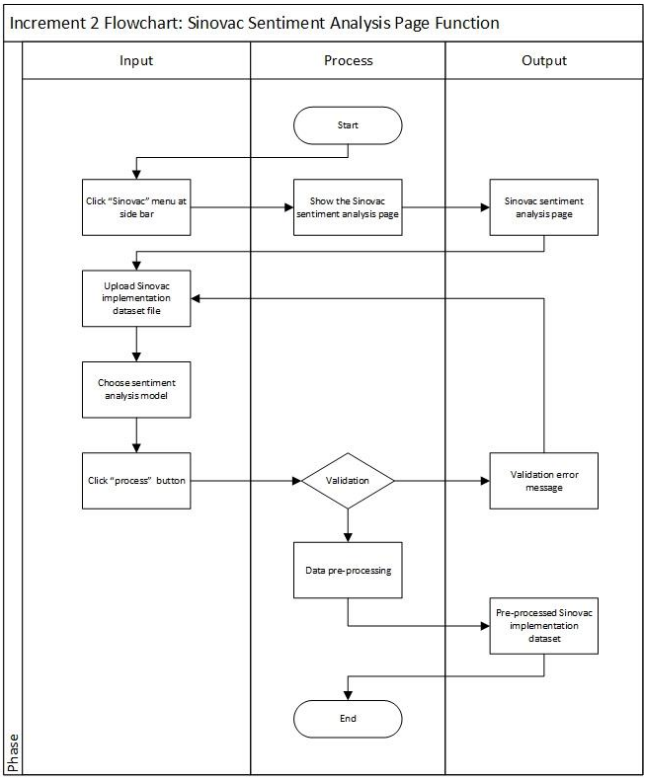


Figure 3.105 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Page Function Flowchart

The Figure 3.105 shows the sentiment analysis web application function flow for accessing and using the Sinovac sentiment analysis page. First, the user clicks the "Sinovac" menu at the side bar on the left of the page. Second, the system renders the Sinovac sentiment analysis view through defined route. Third, the sentiment analysis page is displayed on the user interface with form field. Forth, the user uploads the Sinovac implementation dataset through the file input field. Fifth, the user chooses one sentiment analysis model between the hybrid CNN-LSTM, single CNN model, and single LSTM model through the selection input field to perform a sentiment classification. Sixth, the system validates the user inputs, if the user inputs are invalid, then display validation error messages, otherwise, the system performs data pre-processing. Finally, the system saves the pre-processed Sinovac implementation dataset in CSV format as the output.

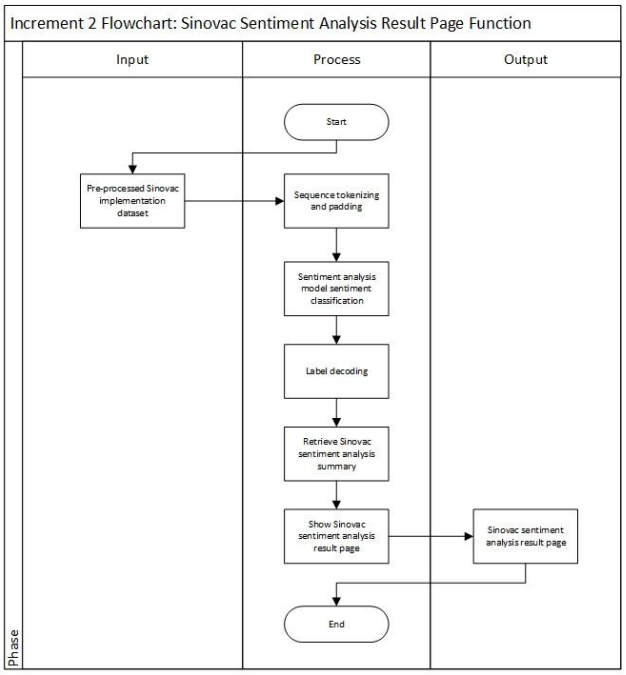


Figure 3.106 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Result Page Function Flowchart

The Figure 3.106 shows the sentiment analysis web application function flow for accessing and using the Sinovac sentiment analysis result page. First, after the Sinovac implementation dataset has pre-processed and saved successfully, the system immediately reads the saved pre-processed Sinovac implementation dataset. Second, the system tokenizes and pads the sequences in the pre-processed Sinovac implementation dataset. Third, the system loads the chosen sentiment analysis model and performs sentiment classification. Forth, the system decodes the classified sentiment label such as negative sentiment for 0 classified label and positive for 1 classified label. Fifth, the system retrieves sentiment analysis result by filtering, aggregating, grouping, and counting for the sentiment analysis summarization. Sixth, the system renders Sinovac sentiment analysis view based on defined route. Finally, the Sinovac sentiment analysis result page is displayed on the user interface with panel data, prediction results, and sentiment analysis visualizations as the sentiment analysis summary.

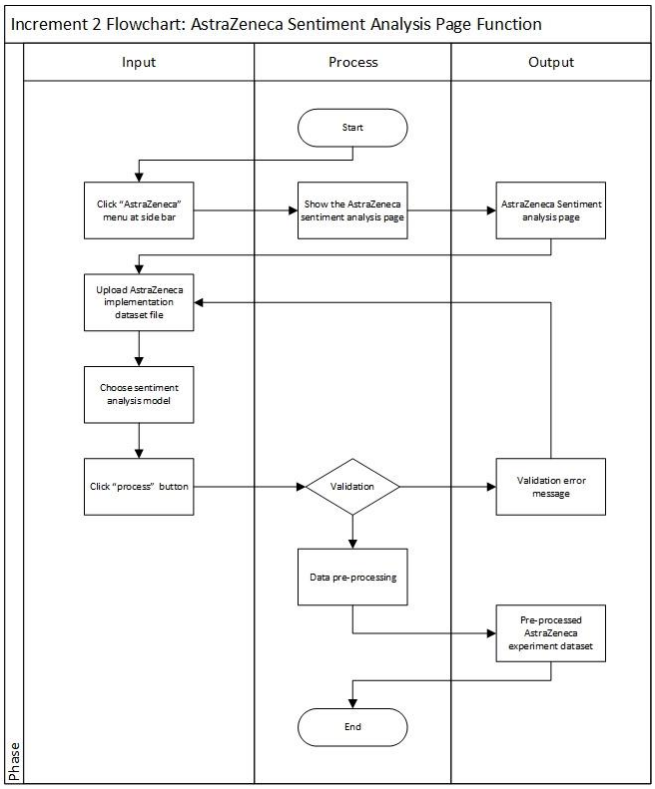


Figure 3.107 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Page Function Flowchart

The Figure 3.107 shows the sentiment analysis web application function flow for accessing and using the AstraZeneca sentiment analysis page. First, the user clicks the "AstraZeneca" menu at the side bar on the left of the page. Second, the system renders the AstraZeneca sentiment analysis view through defined route. Third, the sentiment analysis page is displayed on the user interface with form field. Forth, the user uploads the AstraZeneca implementation dataset through the file input field. Fifth, the user chooses one sentiment analysis model between the hybrid CNN-LSTM, single CNN model, and single LSTM model through the selection input field to perform a sentiment classification. Sixth, the system validates the user inputs, if the user inputs are invalid, then display validation error messages, otherwise, the system performs data pre-processing. Finally, the system saves the pre-processed AstraZeneca implementation dataset in CSV format as the output.

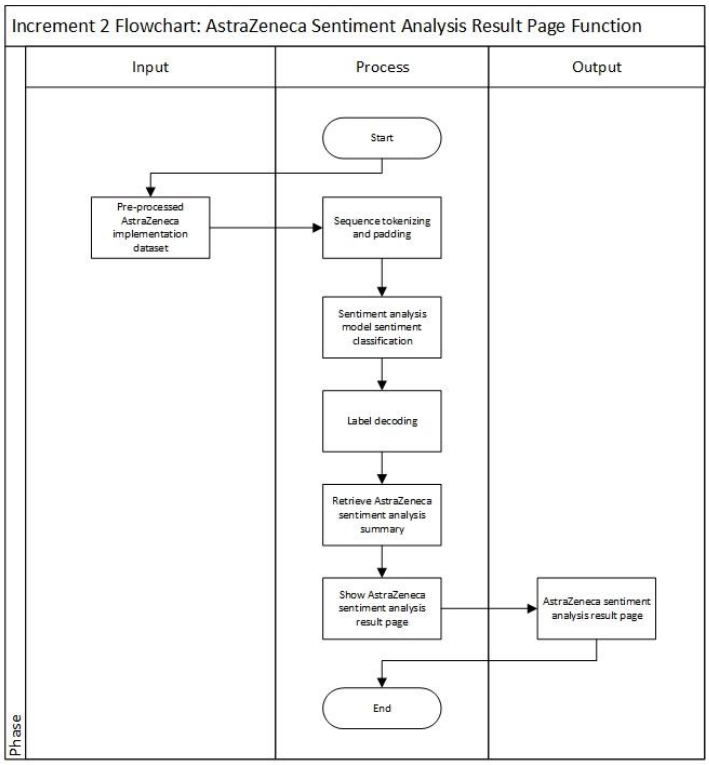


Figure 3.108 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Result Page Function Flowchart

The Figure 3.108 shows the sentiment analysis web application function flow for accessing and using the AstraZeneca sentiment analysis result page. First, after the AstraZeneca implementation dataset has pre-processed and saved successfully, the system immediately reads the saved pre-processed AstraZeneca implementation dataset. Second, the system tokenizes and pads the sequences in the pre-processed AstraZeneca implementation dataset. Third, the system loads the chosen sentiment analysis model and performs sentiment classification. Forth, the system decodes the classified sentiment label such as negative sentiment for 0 classified label and positive for 1 classified label. Fifth, the system retrieves sentiment analysis result by filtering, aggregating, grouping, and counting for the sentiment analysis summarization. Sixth, the system renders AstraZeneca sentiment analysis view based on defined route. Finally, the Sinovac sentiment analysis result page is displayed on the user interface with panel data, prediction results, and sentiment analysis visualizations as the sentiment analysis summary.

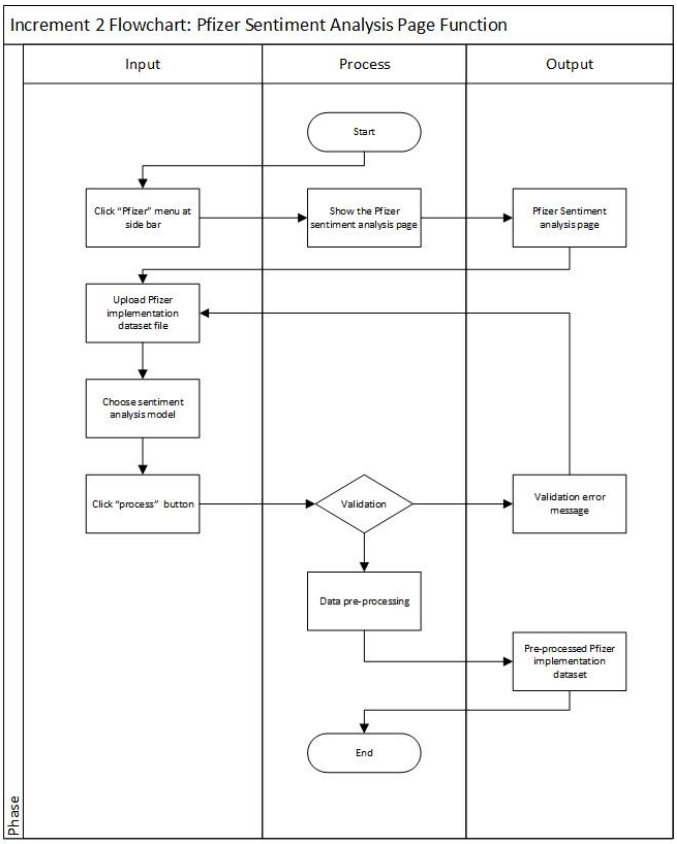


Figure 3.109 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Page Function Flowchart

The Figure 3.109 shows the sentiment analysis web application function flow for accessing and using the Pfizer sentiment analysis page. First, the user clicks the "Pfizer" menu at the side bar on the left of the page. Second, the system renders the Pfizer sentiment analysis view through defined route. Third, the sentiment analysis page is displayed on the user interface with form field. Forth, the user uploads the Pfizer implementation dataset through the file input field. Fifth, the user chooses one sentiment analysis model between the hybrid CNN-LSTM, single CNN model, and single LSTM model through the selection input field to perform a sentiment classification. Sixth, the system validates the user inputs, if the user inputs are invalid, then display validation error messages, otherwise, the system performs data pre-processing. Finally, the system saves the pre-processed Pfizer implementation dataset in CSV format as the output.

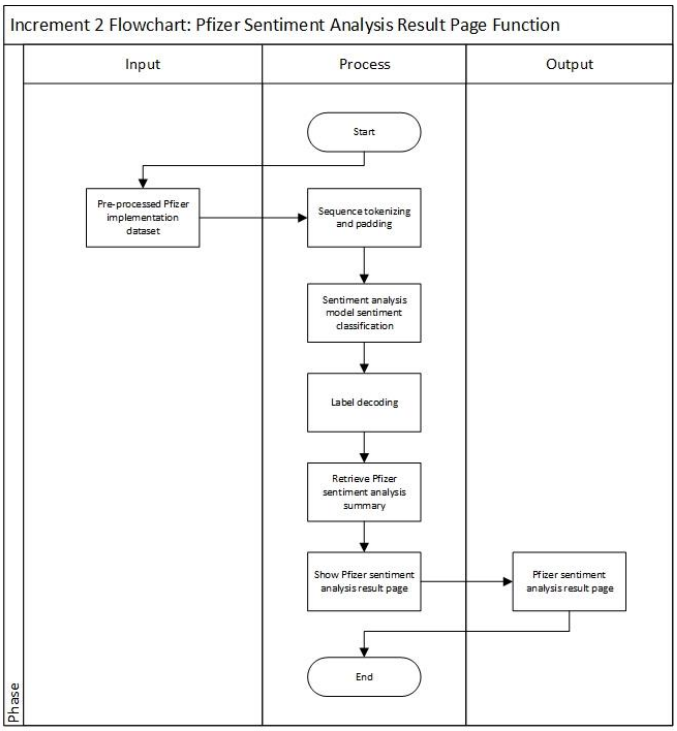


Figure 3.110 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Result Page Function Flowchart

The Figure 3.110 shows the sentiment analysis web application function flow for accessing and using the Pfizer sentiment analysis result page. First, after the Pfizer implementation dataset has pre-processed and saved successfully, the system immediately reads the saved pre-processed Pfizer implementation dataset. Second, the system tokenizes and pads the sequences in the pre-processed Pfizer implementation dataset. Third, the system loads the chosen sentiment analysis model and performs sentiment classification. Forth, the system decodes the classified sentiment label such as negative sentiment for 0 classified label and positive for 1 classified label. Fifth, the system retrieves sentiment analysis result by filtering, aggregating, grouping, and counting for the sentiment analysis summarization. Sixth, the system renders Pfizer sentiment analysis view based on defined route. Finally, the Pfizer sentiment analysis result page is displayed on the user interface with panel data, prediction results, and sentiment analysis visualizations as the sentiment analysis summary.

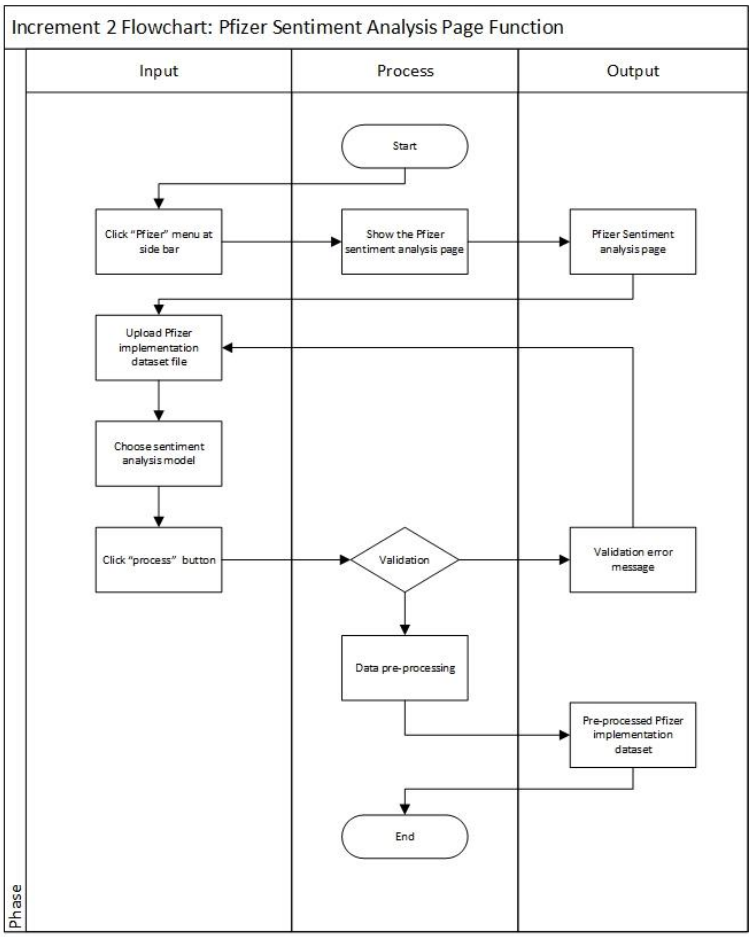


Figure 3.111 The Sentiment Analysis Web Application Moderna Sentiment Analysis Page Function Flowchart

The Figure 3.111 shows the sentiment analysis web application function flow for accessing and using the Moderna sentiment analysis page. First, the user clicks the "Moderna" menu at the side bar on the left of the page. Second, the system renders the Moderna sentiment analysis view through defined route. Third, the sentiment analysis page is displayed on the user interface with form field. Forth, the user uploads the Moderna implementation dataset through the file input field. Fifth, the user chooses one sentiment analysis model between the hybrid CNN-LSTM, single CNN model, and single LSTM model through the selection input field to perform a sentiment classification. Sixth, the system validates the user inputs, if the user inputs are invalid, then display validation error messages, otherwise, the system performs data pre-processing. Finally, the system saves the pre-processed Moderna implementation dataset in CSV format as the output.

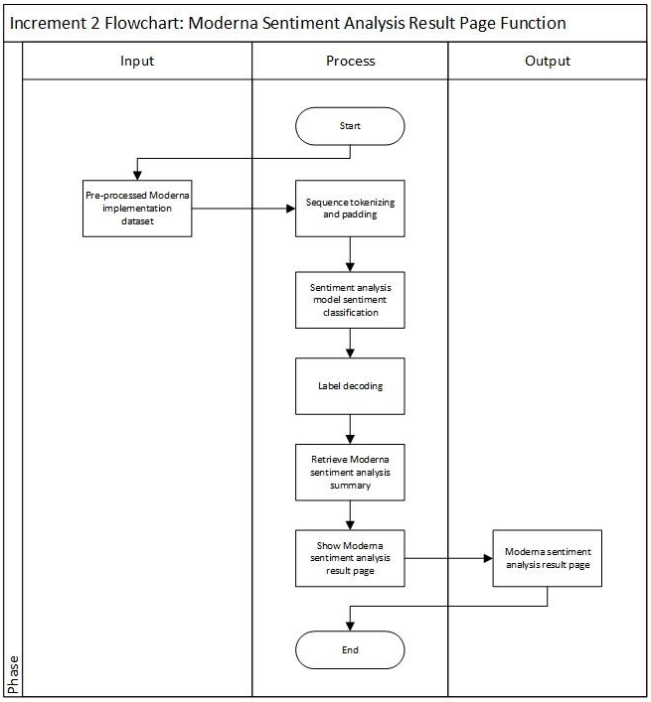


Figure 3.112 The Sentiment Analysis Web Application Moderna Sentiment Analysis Result Page Function Flowchart

The Figure 3.112 shows the sentiment analysis web application function flow for accessing and using the Moderna sentiment analysis result page. First, after the Moderna implementation dataset has pre-processed and saved successfully, the system immediately reads the saved pre-processed Moderna implementation dataset. Second, the system tokenizes and pads the sequences in the pre-processed Moderna implementation dataset. Third, the system loads the chosen sentiment analysis model and performs sentiment classification. Forth, the system decodes the classified sentiment label such as negative sentiment for 0 classified label and positive for 1 classified label. Fifth, the system retrieves sentiment analysis result by filtering, aggregating, grouping, and counting for the sentiment analysis summarization. Sixth, the system renders Moderna sentiment analysis view based on defined route. Finally, the Moderna sentiment analysis result page is displayed on the user interface with panel data, prediction results, and sentiment analysis visualizations as the sentiment analysis summary.

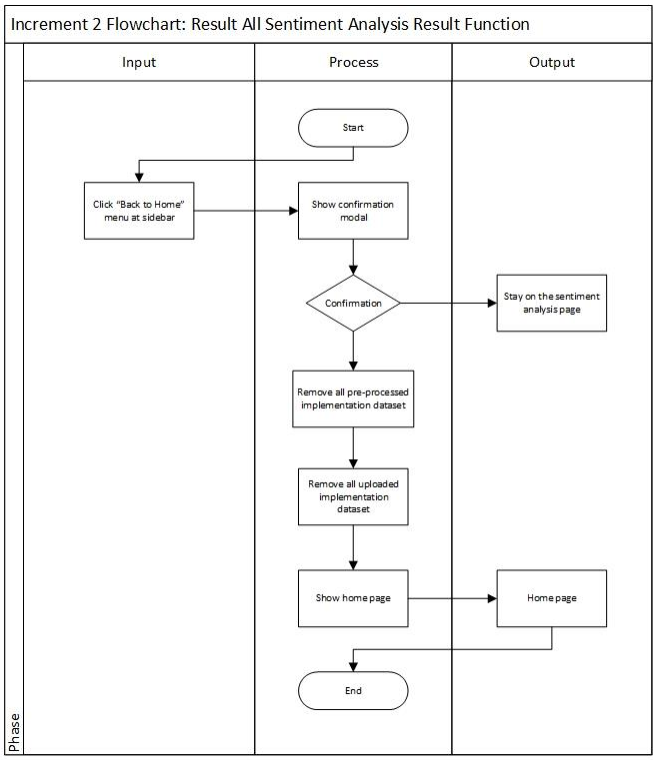


Figure 3.113 The Sentiment Analysis Web Application Reset All Sentiment Analysis Result Function Flowchart

The Figure 3.113 shows the sentiment analysis web application function flow for resetting all sentiment analysis results by quitting the sentiment analysis page. First, the user clicks the “Back to Home” menu at the side bar on the left of the page. Second, the system shows the quit confirmation modal to alert the user before quitting the sentiment analysis page. Third, if the user chooses “cancel”, then it stays on the sentiment analysis page, otherwise, the system resets all the sentiment analysis results. Forth, the system removes all the pre-processed and uploaded implementation datasets. Fifth, the system renders home view based on defined route. Finally, the home page is displayed on the user interface.

3.5.2.1 User Interface Wireframe Design

Figure 3.114 The Sentiment Analysis Web Application Home Page User Interface Wireframe Design

The Figure 3.114 shows the sentiment analysis web application user interface for the home page. The home page has navigation bar at the top, sliding carousels, contents, and footer at the bottom. The home page content explains and presents about the domains to the user such as NLP, sentiment analysis, Sinovac vaccine, AstraZeneca vaccine, Pfizer vaccine, and Moderna vaccine.

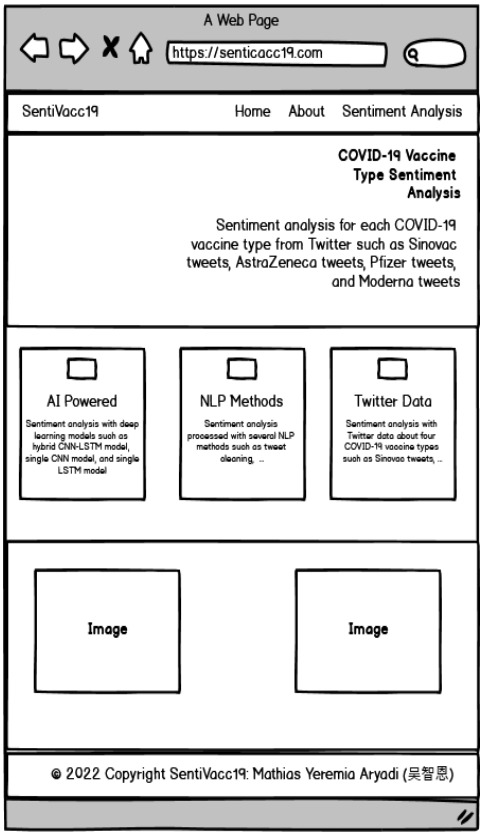


Figure 3.115 The Sentiment Analysis Web Application About Page User Interface Wireframe Design

The Figure 3.115 shows the sentiment analysis web application user interface for the about page. The about page has navigation bar at the top, sliding carousels, contents, and footer at the bottom. The about page content explains and presents about the technologies and tools behind the sentiment analysis web application to the user such as AI technologies, NLP technologies, and Twitter data.

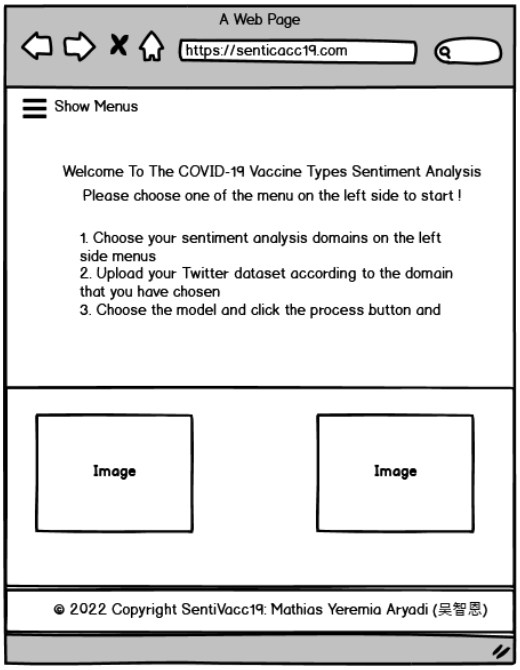


Figure 3.116 The Sentiment Analysis Web Application Sentiment Analysis Page User Interface Wireframe Design

The Figure 3.116 shows the sentiment analysis web application user interface for the sentiment analysis page. The sentiment analysis page has side bar on the left, sliding carousels, content, and footer at the bottom. The sentiment analysis page content explains and presents about the instructions to use the sentiment analysis web application to the user.

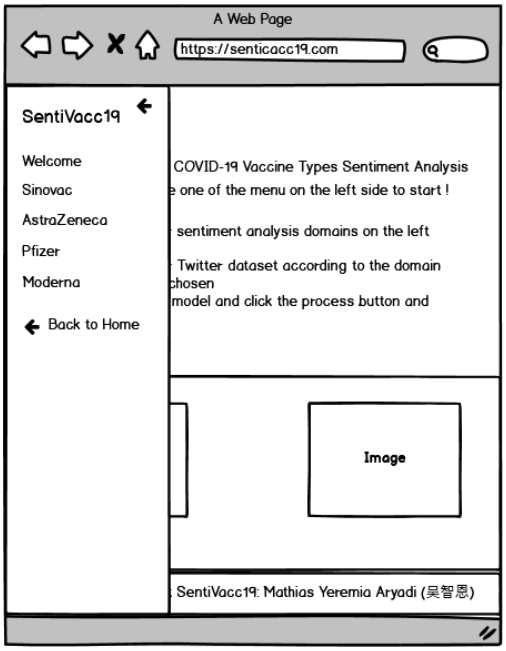


Figure 3.117 The Sentiment Analysis Web Application Sentiment Analysis Page with Opened Sidebar User Interface Wireframe Design

The Figure 3.117 shows the sentiment analysis web application user interface for the sentiment analysis page with opened side bar. The side bar shows all the subpages in the sentiment analysis page such as welcome page, Sinovac sentiment analysis page, AstraZeneca sentiment analysis page, Pfizer sentiment analysis page, Moderna sentiment analysis page, and reset all sentiment analysis result menu.

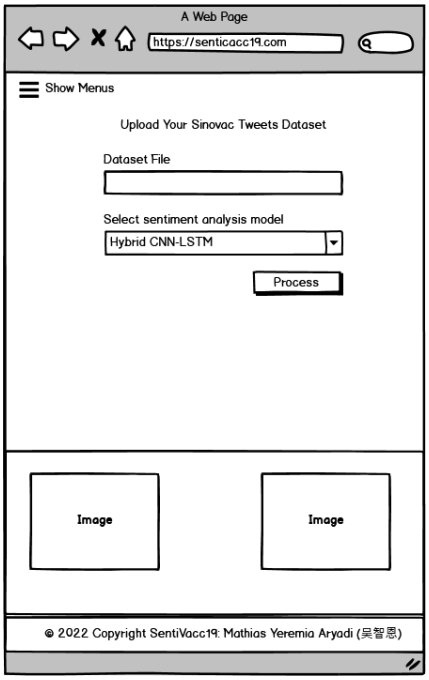


Figure 3.118 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Page User Interface Wireframe Design

The Figure 3.118 shows the sentiment analysis web application user interface for the Sinovac sentiment analysis page. The Sinovac sentiment analysis page has side bar on the left, contents, and footer at the bottom. The Sinovac sentiment analysis page contents have file input field, selection input field, and process button. The file input field contains the uploaded Sinovac implementation dataset, the selection input contains the chosen sentiment analysis model, and the process button processes all the input data.

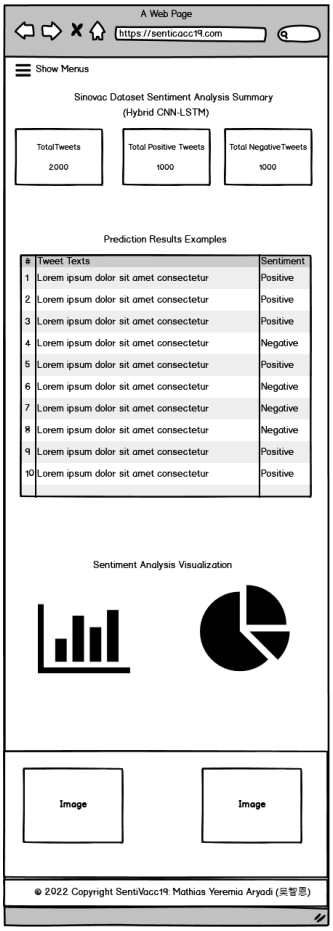


Figure 3.119 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Result Page User Interface Wireframe Design

The Figure 3.119 shows the sentiment analysis web application user interface for the Sinovac sentiment analysis result page. The Sinovac sentiment analysis result page has side bar on the left, contents, and footer at the bottom. The Sinovac sentiment analysis page contents have three data panels, prediction result table, and sentiment analysis visualization. The three data panels consisted of number of Sinovac tweets, number of positive sentiments in Sinovac tweets, and number of negative sentiments in Sinovac tweets. The prediction result table shows the ten examples of classified or predicted Sinovac sentiments. The sentiment analysis visualization visualizes the Sinovac sentiment analysis result with bar plot and pie plot.

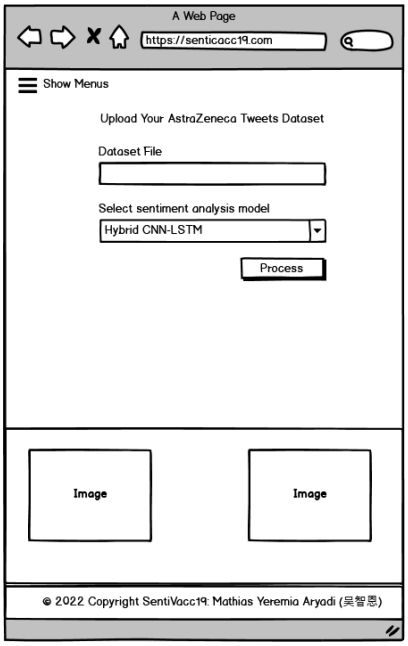


Figure 3.120 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Page User Interface Wireframe Design

The Figure 3.120 shows the sentiment analysis web application user interface for the AstraZeneca sentiment analysis page. The AstraZeneca sentiment analysis page has side bar on the left, contents, and footer at the bottom. The AstraZeneca sentiment analysis page contents have file input field, selection input field, and process button. The file input field contains the uploaded AstraZeneca implementation dataset, the selection input contains the chosen sentiment analysis model, and the process button processes all the input data.

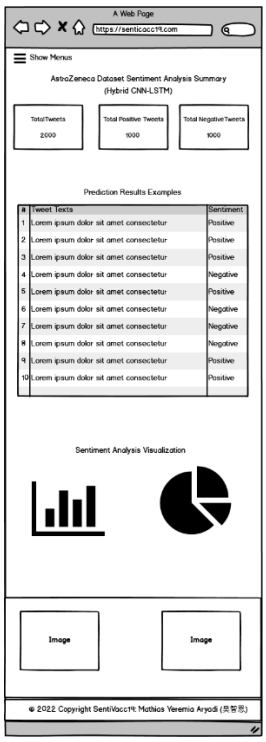


Figure 3.121 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Result Page User Interface Wireframe Design

The Figure 3.121 shows the sentiment analysis web application user interface for the AstraZeneca sentiment analysis result page. The AstraZeneca sentiment analysis result page has side bar on the left, contents, and footer at the bottom. The AstraZeneca sentiment analysis page contents have three data panels, prediction result table, and sentiment analysis visualization. The three data panels consisted of number of AstraZeneca tweets, number of positive sentiments in AstraZeneca tweets, and number of negative sentiments in AstraZeneca tweets. The prediction result table shows the ten examples of classified or predicted AstraZeneca sentiments. The sentiment analysis visualization visualizes the AstraZeneca sentiment analysis result with bar plot and pie plot.

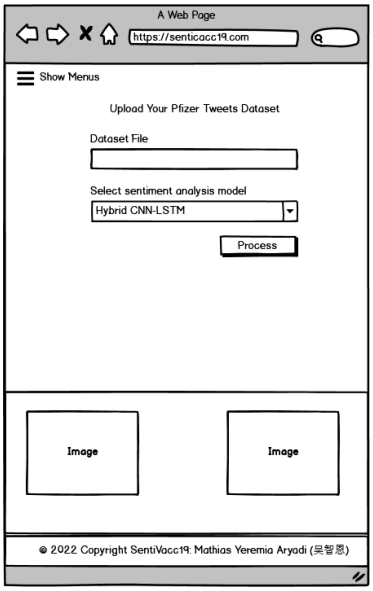


Figure 3.122 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Page User Interface Wireframe Design

The Figure 3.122 shows the sentiment analysis web application user interface for the Pfizer sentiment analysis page. The Pfizer sentiment analysis page has side bar on the left, contents, and footer at the bottom. The Pfizer sentiment analysis page contents have file input field, selection input field, and process button. The file input field contains the uploaded Pfizer implementation dataset, the selection input contains the chosen sentiment analysis model, and the process button processes all the input data.

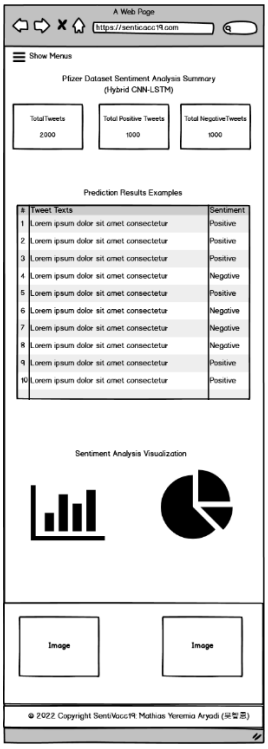


Figure 3.123 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Result Page User Interface Wireframe Design

The Figure 3.123 shows the sentiment analysis web application user interface for the Pfizer sentiment analysis result page. The Pfizer sentiment analysis result page has side bar on the left, contents, and footer at the bottom. The Pfizer sentiment analysis page contents have three data panels, prediction result table, and sentiment analysis visualization. The three data panels consisted of number of Pfizer tweets, number of positive sentiments in Pfizer tweets, and number of negative sentiments in Pfizer tweets. The prediction result table shows the ten examples of classified or predicted Pfizer sentiments. The sentiment analysis visualization visualizes the Pfizer sentiment analysis result with bar plot and pie plot.

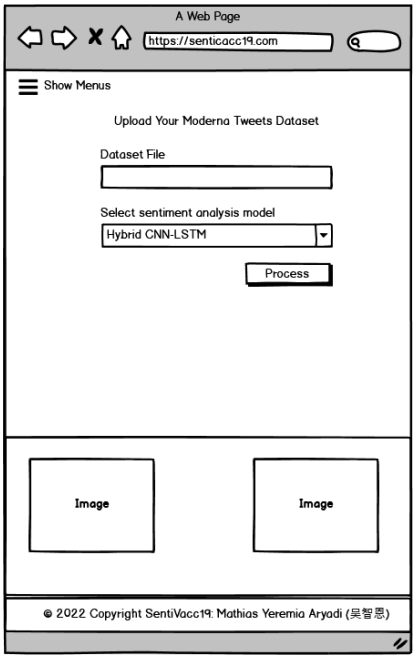


Figure 3.124 The Sentiment Analysis Web Application Moderna Sentiment Analysis Page User Interface Wireframe Design

The Figure 3.124 shows the sentiment analysis web application user interface for the Moderna sentiment analysis page. The Moderna sentiment analysis page has side bar on the left, contents, and footer at the bottom. The Moderna sentiment analysis page contents have file input field, selection input field, and process button. The file input field contains the uploaded Moderna implementation dataset, the selection input contains the chosen sentiment analysis model, and the process button processes all the input data.

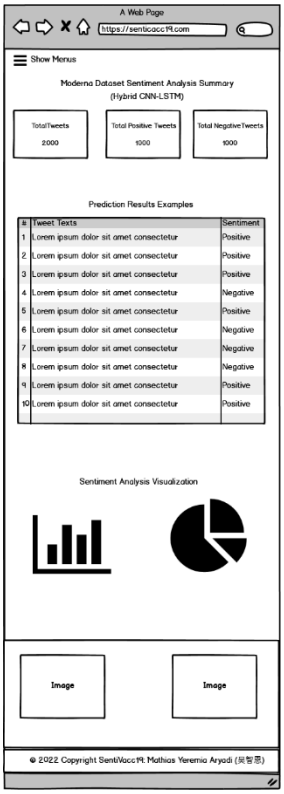


Figure 3.125 The Sentiment Analysis Web Application Moderna Sentiment Analysis Result Page User Interface Wireframe Design

The Figure 3.125 shows the sentiment analysis web application user interface for the Moderna sentiment analysis result page. The Moderna sentiment analysis result page has side bar on the left, contents, and footer at the bottom. The Pfizer sentiment analysis page contents have three data panels, prediction result table, and sentiment analysis visualization. The three data panels consisted of number of Pfizer tweets, number of positive sentiments in Pfizer tweets, and number of negative sentiments in Pfizer tweets. The prediction result table shows the ten examples of classified or predicted Pfizer sentiments. The sentiment analysis visualization visualizes the Pfizer sentiment analysis result with bar plot and pie plot.

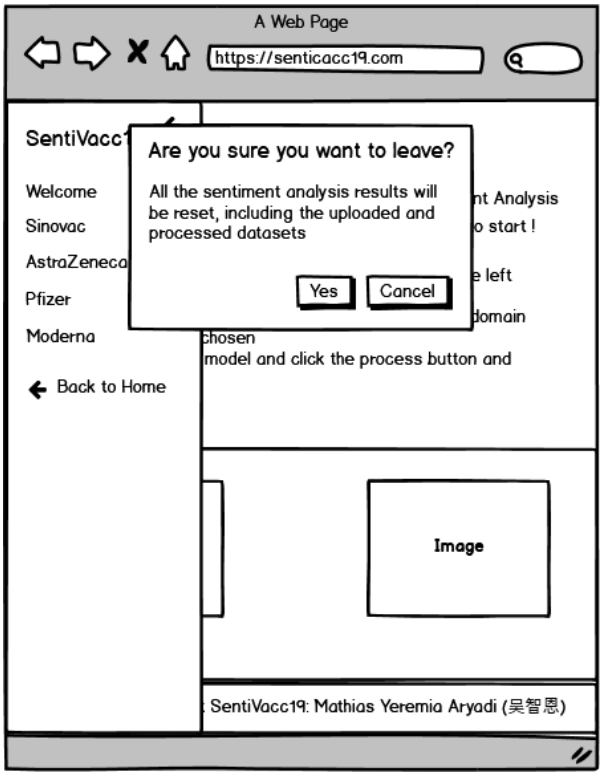


Figure 3.126 The Sentiment Analysis Web Application Reset All Sentiment Analysis Confirmation User Interface Wireframe Design

The Figure 3.126 shows the sentiment analysis web application user interface for quitting from the sentiment analysis page and resetting all the sentiment analysis results. The quitting confirmation modal is showed when the user clicked the “Back to Home” button to alert the user before quitting or leaving the sentiment analysis page.

3.5.3 Implementation

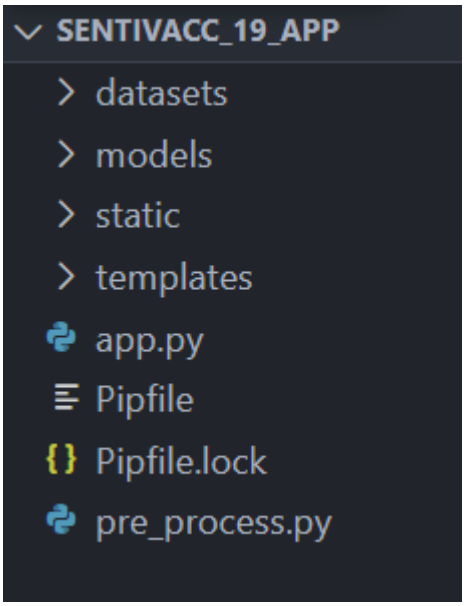


Figure 3.127 The Sentiment Analysis Web Application Flask Web Project Structure

The Figure 3.127 shows the sentiment analysis web application for the *flask* project structure. The *datasets* directory contains all the uploaded and pre-processed implementation datasets in CSV format. The *models* directory contains all the trained sentiment analysis models such as the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model in H5 format for four dataset topics. The *static* directory contains all the static resource for the web application such as *CSS, JavaScript,* and images. The *templates* directory contains all the sentiment analysis web application pages in HTML file. The *app.py* file contains all the *flask* main web application logic code such as routes, form, and handling the implementation dataset. The *pre_process.py* contains all the data pre-processing functions.

```
def select_model_predict(df):
    selected_model_text = ''
    model = None

    if df['model'].iloc[0] == 'cnn-lstm':
        selected_model_text = 'Hybrid CNN-LSTM'
        file_name = df['topic'].iloc[0] + '_cnn_lstm_model.h5'
        model = load_model(os.path.join(app.root_path, 'models', file_name))
        X_test = transform_data(df)
        y_pred_raw = model.predict(X_test)
        y_pred = np.argmax(y_pred_raw, axis=1)
        df['sentiment'] = y_pred
        df['sentiment'] = df['sentiment'].apply(lambda x: dencode_label(x))
        df = df[['text', 'sentiment']]
    elif df['model'].iloc[0] == 'cnn':
        selected_model_text = 'Single CNN Model'
        file_name = df['topic'].iloc[0] + '_cnn_model.h5'
        model = load_model(os.path.join(app.root_path, 'models', file_name))
        X_test = transform_data(df)
        y_pred_raw = model.predict(X_test)
        y_pred = np.argmax(y_pred_raw, axis=1)
        df['sentiment'] = y_pred
        df['sentiment'] = df['sentiment'].apply(lambda x: dencode_label(x))
        df = df[['text', 'sentiment']]
    else:
        selected_model_text = 'Single LSTM Model'
        file_name = df['topic'].iloc[0] + '_lstm_model.h5'
        model = load_model(os.path.join(app.root_path, 'models', file_name))
        X_test = transform_data(df)
        y_pred_raw = model.predict(X_test)
        y_pred = np.argmax(y_pred_raw, axis=1)
        df['sentiment'] = y_pred
        df['sentiment'] = df['sentiment'].apply(lambda x: dencode_label(x))
        df = df[['text', 'sentiment']]

    return df, selected_model_text
```

Figure 3.128 The Sentiment Analysis Web Application Sentiment Analysis Model Implementation Code

The Figure 3.128 shows the sentiment analysis web application for implementing the chosen sentiment analysis models to predicts or classify the unlabeled implementation dataset and produce a sentiment analysis result. First, the selected model text is set based on the chosen sentiment analysis model. Second, the sentiment analysis model is loaded according to the dataset topic. Third, the pre-processed implementation dataset is transformed into token and sequence of positive integers using *transform_data()* function so the model can predict with the proper data form. Forth, the predicted sentiments using *predict()* function are set to the implementation dataset new column and decode the predicted label into "Positive" and "Negative" using *decode_label()* function. Finally, the implementation dataset is filtered by taking the raw tweet texts column and the predicted sentiments column to be shown in the sentiment analysis result page.

```python
# Home view
@app.route('/', methods=['GET'])
def render_index_view():
    title = 'SentiVacc19 | Home'
    return render_template('index.html', title=title)
```

Figure 3.129 The Sentiment Analysis Web Application Home Page Route Code

The Figure 3.129 shows the sentiment analysis web application for implementing the home page function. The home page is defined using *flask* route with "/" URL and *GET* request method. The *render_index_view()* function handle the home page route to display the home page view using *render_template()* function, passing the *index.html* file for the home page and the title of the home page.

```python
# About view
@app.route('/about', methods=['GET'])
def render_about_view():
    title = 'SentiVacc19 | About'
    return render_template('about.html', title=title)
```

Figure 3.130 The Sentiment Analysis Web Application About Page Route Code

The Figure 3.130 shows the sentiment analysis web application for implementing the about page function. The about page is defined using *flask* route with "/about" URL and *GET* request method. The *render_about_view()* function handle the about page route to display the about page view using *render_template()* function, passing the *about.html* file for the about page and the title of the about page.

```python
# Sentiment analysis views
@app.route('/sentiment-analysis', methods=['GET'])
def render_sentiment_analysis_view():
    title = 'SentiVacc19 | Sentiment Analysis'
    return render_template('sentiment-analysis.html', title=title)
```

Figure 3.131 The Sentiment Analysis Web Application Sentiment Analysis Page Route Code

The Figure 3.131 shows the sentiment analysis web application for implementing the sentiment analysis page function. The about page is defined using *flask* route with "/sentiment-analysis" URL and *GET* request method. The

render_sentiment_analysis_view() function handle the sentiment analysis page route to display the sentiment analysis page view using *render_template()* function, passing the *sentiment-analysis.html* file for the sentiment analysis page and the title of the sentiment analysis page.

```
# Sinovac views
@app.route('/sentiment-analysis/sinovac', methods=['GET', 'POST'])
def render_sinovac_view():
    if 'sinovac_session' in session:
        return redirect(url_for('render_sinovac_result_view'))

    title = 'SentiVacc19 | Sentiment Analysis: Sinovac'
    form = SentimentForm()

    if form.validate_on_submit():
        f = form.dataset_file_field.data
        filename = secure_filename(f.filename)
        f.save(os.path.join(app.root_path, 'datasets', filename))

        selected_model = form.model_select_field.data
        df_sinovac = pd.read_csv(os.path.join(app.root_path, 'datasets', filename))
        df_sinovac = set_sentiment_analysis_model(selected_model, df_sinovac)
        df_sinovac = set_topic('sinovac', df_sinovac)
        pre_process(df_sinovac, 'sinovac_cleaned.csv')
        session['sinovac_session'] = 1

        return redirect(url_for('render_sinovac_result_view'))

    return render_template('pages/sinovac.html', title=title, form=form)
```

Figure 3.132 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Page Route Code

The Figure 3.132 shows the sentiment analysis web application for implementing the Sinovac sentiment analysis page function. The Sinovac sentiment analysis page is defined using *flask* route with *sentiment-analysis/sinovac* URL and *GET* and *POST* request method, because the page has form processing. The *render_sinovac_view()* function handle the Sinovac sentiment analysis page route with several logics. First, if the user already uploaded and produced the Sinovac sentiment analysis result through the session, then the user is redirected to the Sinovac sentiment analysis result page, otherwise, the Sinovac sentiment analysis page is shown. Second, perform form validation with *flask-wtf* library using *validate_on_submit()* function that the file is required and has to be in CSV format. Third, if there is no violation from the validation form, the implementation Sinovac dataset is uploaded using *save()* function; set with the chosen sentiment analysis models using *set_sentiment_analysis_model()* function and Sinovac topic using *set_topic()* function; applied with data pre-processing using *pre_process()* function; set the Sinovac session that the user already uploaded and produced the Sinovac sentiment analysis result. Finally, Sinovac sentiment analysis page is displayed

through redirecting to the Sinovac sentiment analysis result page with *redirect()* and *url_for()* function, passing the *render_sinovac_result_view* route function.

```
# Result
@app.route('/sentiment-analysis/sinovac/result', methods=['GET'])
def render_sinovac_result_view():
    if 'sinovac_session' not in session:
        return redirect(url_for('render_sinovac_view'))

    title = 'SentiVacc19 | Sentiment Analysis: Sinovac Result'
    df_sinovac = pd.read_csv(os.path.join(app.root_path, 'datasets', 'sinovac_cleaned.csv'))

    df_sinovac, selected_model_text = select_model_predict(df_sinovac)
    sinovac_data = df_sinovac.copy()
    sinovac_count = sinovac_data['sentiment'].value_counts().rename_axis('sentiment').reset_index(name='count')
    sinovac_tweets = sinovac_data.shape[0]
    sinovac_positive = sinovac_count[sinovac_count['sentiment'] == 'Positive']['count'].values[0]
    sinovac_negative = sinovac_count[sinovac_count['sentiment'] == 'Negative']['count'].values[0]
    sinovac_data = sinovac_data.head(10)
    sinovac_bar_plot = plot_bar(sinovac_count)
    sinovac_pie_plot = plot_pie(sinovac_count)

    return render_template(
        'pages/sinovac-result.html',
        title=title,
        selected_model_text=selected_model_text,
        sinovac_tweets=sinovac_tweets,
        sinovac_positive=sinovac_positive,
        sinovac_negative=sinovac_negative,
        sinovac_data=sinovac_data,
        sinovac_bar_plot=sinovac_bar_plot,
        sinovac_pie_plot=sinovac_pie_plot
    )
```

Figure 3.133 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Result Page Route Code

The Figure 3.133 shows the sentiment analysis web application for implementing the Sinovac sentiment analysis result page function. The Sinovac sentiment analysis result page is defined using *flask* route with */sentiment-analysis/sinovac/result* URL and *GET* request method. The *render_sinovac_result_view()* function handle the Sinovac sentiment analysis result page route with several logics. First, if the user has not uploaded and produced the Sinovac sentiment analysis result yet through the session, then the user is redirected to the Sinovac sentiment analysis page, otherwise, the Sinovac sentiment analysis result page is shown. Second, preform the sentiment analysis model prediction or classification using *select_model_predict()* function. Third, the predicted or classified Sinovac implementation dataset will be implemented aggregating and filtering to retrieve the number of Sinovac tweets, the number of Sinovac positive tweets, and the number of Sinovac negative tweets. Forth, the aggregated predicted or classified Sinovac implementation dataset will be visualized into bar plot and pie plot using *plotly* library through *plot_bar()* function and *plot_pie()* function. Finally, Sinovac sentiment analysis result page is displayed through to the Sinovac sentiment analysis result page view using *render_template()* function, passing the *sinovac-result.html* file for the sentiment analysis page and several Sinovac sentiment analysis summary data.

```python
# AstraZeneca views
@app.route('/sentiment-analysis/astrazeneca', methods=['GET', 'POST'])
def render_astrazeneca_view():
    if 'astrazeneca_session' in session:
        return redirect(url_for('render_astrazeneca_result_view'))

    title = 'SentiVacc19 | Sentiment Analysis: AstraZeneca'
    form = SentimentForm()

    if form.validate_on_submit():
        f = form.dataset_file_field.data
        filename = secure_filename(f.filename)
        f.save(os.path.join(app.root_path, 'datasets', filename))

        selected_model = form.model_select_field.data
        df_astrazeneca = pd.read_csv(os.path.join(app.root_path, 'datasets', filename))
        df_astrazeneca = set_sentiment_analysis_model(selected_model, df_astrazeneca)
        df_astrazeneca = set_topic('astrazeneca', df_astrazeneca)
        pre_process(df_astrazeneca, 'astrazeneca_cleaned.csv')
        session['astrazeneca_session'] = 1

        return redirect(url_for('render_astrazeneca_result_view'))

    return render_template('pages/astrazeneca.html', title=title, form=form)
```

Figure 3.134 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Page Route Code

The Figure 3.134 shows the sentiment analysis web application for implementing the AstraZeneca sentiment analysis page function. The AstraZeneca sentiment analysis page is defined using *flask* route with */sentiment-analysis/astrazeneca* URL and *GET* and *POST* request method, because the page has form processing. The *render_astrazeneca_view()* function handle the AstraZeneca sentiment analysis page route with several logics. First, if the user already uploaded and produced the AstraZeneca sentiment analysis result through the session, then the user is redirected to the AstraZeneca sentiment analysis result page, otherwise, the AstraZeneca sentiment analysis page is shown. Second, perform form validation with *flask-wtf* library using *validate_on_submit()* function that the file is required and has to be in CSV format. Third, if there is no violation from the validation form, the implementation AstraZeneca dataset is uploaded using *save()* function; set with the chosen sentiment analysis models using *set_sentiment_analysis_model()* function and AstraZeneca topic using *set_topic()* function; applied with data pre-processing using *pre_process()* function; set the AstraZeneca session that the user already uploaded and produced the AstraZeneca sentiment analysis result. Finally, AstraZeneca sentiment analysis page is displayed through redirecting to the AstraZeneca sentiment analysis result page with *redirect()* and *url_for()* function, passing the *render_astrazeneca_result_view* route function.

```python
@app.route('/sentiment-analysis/astrazeneca/result', methods=['GET'])
def render_astrazeneca_result_view():
    if 'astrazeneca_session' not in session:
        return redirect(url_for('render_astrazeneca_view'))

    title = 'SentiVacc19 | Sentiment Analysis: AstraZeneca Result'
    df_astrazeneca = pd.read_csv(os.path.join(app.root_path, 'datasets', 'astrazeneca_cleaned.csv'))

    df_astrazeneca, selected_model_text = select_model_predict(df_astrazeneca)
    astrazeneca_data = df_astrazeneca.copy()
    astrazeneca_count = astrazeneca_data['sentiment'].value_counts().rename_axis('sentiment').reset_index(name='count')
    astrazeneca_tweets = astrazeneca_data.shape[0]
    astrazeneca_positive = astrazeneca_count[astrazeneca_count['sentiment'] == 'Positive']['count'].values[0]
    astrazeneca_negative = astrazeneca_count[astrazeneca_count['sentiment'] == 'Negative']['count'].values[0]
    astrazeneca_data = astrazeneca_data.head(10)
    astrazeneca_bar_plot = plot_bar(astrazeneca_count)
    astrazeneca_pie_plot = plot_pie(astrazeneca_count)

    return render_template(
        'pages/astrazeneca-result.html',
        title=title,
        selected_model_text=selected_model_text,
        astrazeneca_tweets=astrazeneca_tweets,
        astrazeneca_positive=astrazeneca_positive,
        astrazeneca_negative=astrazeneca_negative,
        astrazeneca_data=astrazeneca_data,
        astrazeneca_bar_plot=astrazeneca_bar_plot,
        astrazeneca_pie_plot=astrazeneca_pie_plot
    )
```

Figure 3.135 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Result Page Route Code

The Figure 3.135 shows the sentiment analysis web application for implementing the AstraZeneca sentiment analysis result page function. The AstraZeneca sentiment analysis result page is defined using *flask* route with */sentiment-analysis/astrazeneca/result* URL and *GET* request method. The *render_astrazeneca_result_view()* function handle the AstraZeneca sentiment analysis result page route with several logics. First, if the user has not uploaded and produced the AstraZeneca sentiment analysis result yet through the session, then the user is redirected to the AstraZeneca sentiment analysis page, otherwise, the AstraZeneca sentiment analysis result page is shown. Second, preform the sentiment analysis model prediction or classification using *select_model_predict()* function. Third, the predicted or classified AstraZeneca implementation dataset will be implemented aggregating and filtering to retrieve the number of AstraZeneca tweets, the number of AstraZeneca positive tweets, and the number of AstraZeneca negative tweets. Forth, the aggregated predicted or classified AstraZeneca implementation dataset will be visualized into bar plot and pie plot using *plotly* library through *plot_bar()* function and *plot_pie()* function. Finally, AstraZeneca sentiment analysis result page is displayed through to the AstraZeneca sentiment analysis result page view using *render_template()* function, passing the *astrazeneca-result.html* file for the sentiment analysis page and several AstraZeneca sentiment analysis summary data.

```python
# Pfizer views
@app.route('/sentiment-analysis/pfizer', methods=['GET', 'POST'])
def render_pfizer_view():
    if 'pfizer_session' in session:
        return redirect(url_for('render_pfizer_result_view'))

    title = 'SentiVacc19 | Sentiment Analysis: Pfizer'
    form = SentimentForm()

    if form.validate_on_submit():
        f = form.dataset_file_field.data
        filename = secure_filename(f.filename)
        f.save(os.path.join(app.root_path, 'datasets', filename))

        selected_model = form.model_select_field.data
        df_pfizer = pd.read_csv(os.path.join(app.root_path, 'datasets', filename))
        df_pfizer = set_sentiment_analysis_model(selected_model, df_pfizer)
        df_pfizer = set_topic('pfizer', df_pfizer)
        pre_process(df_pfizer, 'pfizer_cleaned.csv')
        session['pfizer_session'] = 1

        return redirect(url_for('render_pfizer_result_view'))

    return render_template('pages/pfizer.html', title=title, form=form)
```

Figure 3.136 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Page Route Code

The Figure 3.136 shows the sentiment analysis web application for implementing the Pfizer sentiment analysis page function. The Pfizer sentiment analysis page is defined using *flask* route with */sentiment-analysis/pfizer* URL and *GET* and *POST* request method, because the page has form processing. The *render_pfizer_view()* function handle the Pfizer sentiment analysis page route with several logics. First, if the user already uploaded and produced the Pfizer sentiment analysis result through the session, then the user is redirected to the Pfizer sentiment analysis result page, otherwise, the Pfizer sentiment analysis page is shown. Second, perform form validation with *flask-wtf* library using *validate_on_submit()* function that the file is required and has to be in CSV format. Third, if there is no violation from the validation form, the implementation Pfizer dataset is uploaded using *save()* function; set with the chosen sentiment analysis models using *set_sentiment_analysis_model()* function and Pfizer topic using *set_topic()* function; applied with data pre-processing using *pre_process()* function; set the Pfizer session that the user already uploaded and produced the Pfizer sentiment analysis result. Finally, Pfizer sentiment analysis page is displayed through redirecting to the Pfizer sentiment analysis result page with *redirect()* and *url_for()* function, passing the *render_pfizer_result_view* route function.

```python
@app.route('/sentiment-analysis/pfizer/result', methods=['GET'])
def render_pfizer_result_view():
    if 'pfizer_session' not in session:
        return redirect(url_for('render_pfizer_view'))

    title = 'SentiVacc19 | Sentiment Analysis: Pfizer Result'
    df_pfizer = pd.read_csv(os.path.join(app.root_path, 'datasets', 'pfizer_cleaned.csv'))

    df_pfizer, selected_model_text = select_model_predict(df_pfizer)
    pfizer_data = df_pfizer.copy()
    pfizer_count = pfizer_data['sentiment'].value_counts().rename_axis('sentiment').reset_index(name='count')
    pfizer_tweets = pfizer_data.shape[0]
    pfizer_positive = pfizer_count[pfizer_count['sentiment'] == 'Positive']['count'].values[0]
    pfizer_negative = pfizer_count[pfizer_count['sentiment'] == 'Negative']['count'].values[0]
    pfizer_data = pfizer_data.head(10)
    pfizer_bar_plot = plot_bar(pfizer_count)
    pfizer_pie_plot = plot_pie(pfizer_count)

    return render_template(
        'pages/pfizer-result.html',
        title=title,
        selected_model_text=selected_model_text,
        pfizer_tweets=pfizer_tweets,
        pfizer_positive=pfizer_positive,
        pfizer_negative=pfizer_negative,
        pfizer_data=pfizer_data,
        pfizer_bar_plot=pfizer_bar_plot,
        pfizer_pie_plot=pfizer_pie_plot
    )
```

Figure 3.137 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Result Page Route Code

The Figure 3.137 shows the sentiment analysis web application for implementing the Pfizer sentiment analysis result page function. The Pfizer sentiment analysis result page is defined using *flask* route with */sentiment-analysis/pfizer/result* URL and *GET* request method. The *render_astrazeneca_result_view()* function handle the Pfizer sentiment analysis result page route with several logics. First, if the user has not uploaded and produced the Pfizer sentiment analysis result yet through the session, then the user is redirected to the Pfizer sentiment analysis page, otherwise, the Pfizer sentiment analysis result page is shown. Second, preform the sentiment analysis model prediction or classification using *select_model_predict()* function. Third, the predicted or classified Pfizer implementation dataset will be implemented aggregating and filtering to retrieve the number of Pfizer tweets, the number of Pfizer positive tweets, and the number of Pfizer negative tweets. Forth, the aggregated predicted or classified Pfizer implementation dataset will be visualized into bar plot and pie plot using *plotly* library through *plot_bar()* function and *plot_pie()* function. Finally, Pfizer sentiment analysis result page is displayed through to the Pfizer sentiment analysis result page view using *render_template()* function, passing the *pfizer-result.html* file for the sentiment analysis page and several Pfizer sentiment analysis summary data.

```python
# Moderna views
@app.route('/sentiment-analysis/moderna', methods=['GET', 'POST'])
def render_moderna_view():
    if 'moderna_session' in session:
        return redirect(url_for('render_moderna_result_view'))

    title = 'SentiVacc19 | Sentiment Analysis: Moderna'
    form = SentimentForm()

    if form.validate_on_submit():
        f = form.dataset_file_field.data
        filename = secure_filename(f.filename)
        f.save(os.path.join(app.root_path, 'datasets', filename))

        selected_model = form.model_select_field.data
        df_moderna = pd.read_csv(os.path.join(app.root_path, 'datasets', filename))
        df_moderna = set_sentiment_analysis_model(selected_model, df_moderna)
        df_moderna = set_topic('moderna', df_moderna)
        pre_process(df_moderna, 'moderna_cleaned.csv')
        session['moderna_session'] = 1

        return redirect(url_for('render_moderna_result_view'))

    return render_template('pages/moderna.html', title=title, form=form)
```

Figure 3.138 The Sentiment Analysis Web Application Moderna Sentiment Analysis Page Route Code

The Figure 3.138 shows the sentiment analysis web application for implementing the Moderna sentiment analysis page function. The Moderna sentiment analysis page is defined using *flask* route with */sentiment-analysis/moderna* URL and *GET* and *POST* request method, because the page has form processing. The *render_moderna_view()* function handle the Moderna sentiment analysis page route with several logics. First, if the user already uploaded and produced the Moderna sentiment analysis result through the session, then the user is redirected to the Moderna sentiment analysis result page, otherwise, the Moderna sentiment analysis page is shown. Second, perform form validation with *flask-wtf* library using *validate_on_submit()* function that the file is required and has to be in CSV format. Third, if there is no violation from the validation form, the implementation Moderna dataset is uploaded using *save()* function; set with the chosen sentiment analysis models using *set_sentiment_analysis_model()* function and Moderna topic using *set_topic()* function; applied with data pre-processing using *pre_process()* function; set the Moderna session that the user already uploaded and produced the Moderna sentiment analysis result. Finally, Moderna sentiment analysis page is displayed through redirecting to the Moderna sentiment analysis result page with *redirect()* and *url_for()* function, passing the *render_moderna_result_view* route function.

```python
@app.route('/sentiment-analysis/moderna/result/', methods=['GET'])
def render_moderna_result_view():
    if 'moderna_session' not in session:
        return redirect(url_for('render_moderna_view'))

    title = 'SentiVacc19 | Sentiment Analysis: Moderna Result'
    df_moderna = pd.read_csv(os.path.join(app.root_path, 'datasets', 'moderna_cleaned.csv'))

    df_moderna, selected_model_text = select_model_predict(df_moderna)
    moderna_data = df_moderna.copy()
    moderna_count = moderna_data['sentiment'].value_counts().rename_axis('sentiment').reset_index(name='count')
    moderna_tweets = moderna_data.shape[0]
    moderna_positive = moderna_count[moderna_count['sentiment'] == 'Positive']['count'].values[0]
    moderna_negative = moderna_count[moderna_count['sentiment'] == 'Negative']['count'].values[0]
    moderna_data = moderna_data.head(10)
    moderna_bar_plot = plot_bar(moderna_count)
    moderna_pie_plot = plot_pie(moderna_count)

    return render_template(
        'pages/moderna-result.html',
        title=title,
        selected_model_text=selected_model_text,
        moderna_tweets=moderna_tweets,
        moderna_positive=moderna_positive,
        moderna_negative=moderna_negative,
        moderna_data=moderna_data,
        moderna_bar_plot=moderna_bar_plot,
        moderna_pie_plot=moderna_pie_plot
    )
```

Figure 3.139 The Sentiment Analysis Web Application Moderna Sentiment Analysis Result Page Route Code

The Figure 3.139 shows the sentiment analysis web application for implementing the Moderna sentiment analysis result page function. The Moderna sentiment analysis result page is defined using *flask* route with */sentiment-analysis/moderna/result* URL and *GET* request method. The *render_moderna_result_view()* function handle the Moderna sentiment analysis result page route with several logics. First, if the user has not uploaded and produced the Moderna sentiment analysis result yet through the session, then the user is redirected to the Moderna sentiment analysis page, otherwise, the Moderna sentiment analysis result page is shown. Second, preform the sentiment analysis model prediction or classification using *select_model_predict()* function. Third, the predicted or classified Moderna implementation dataset will be implemented aggregating and filtering to retrieve the number of Moderna tweets, the number of Moderna positive tweets, and the number of Moderna negative tweets. Forth, the aggregated predicted or classified Moderna implementation dataset will be visualized into bar plot and pie plot using *plotly* library through *plot_bar()* function and *plot_pie()* function. Finally, Moderna sentiment analysis result page is displayed through to the Moderna sentiment analysis result page view using *render_template()* function, passing the *moderna-result.html* file for the sentiment analysis page and several Moderna sentiment analysis summary data.

```
# Back to home and reset
@app.route('/sentiment-analysis/reset', methods=['GET'])
def remove_all_files():
    datasets_path = os.path.join(app.root_path, 'datasets')

    for file in os.scandir(datasets_path):
        os.remove(file.path)

    session.pop('sinovac_session', None)
    session.pop('astrazeneca_session', None)
    session.pop('pfizer_session', None)
    session.pop('moderna_session', None)

    return redirect(url_for('render_index_view'))
```

Figure 3.140 The Sentiment Analysis Web Application Reset All Sentiment Analysis Result Route Code

The Figure 3.140 shows the sentiment analysis web application for implementing the reset all the sentiment analysis result function. The reset all the sentiment analysis result is defined using *flask* route with */sentiment-analysis/reset* URL and *GET* request method. The *remove_all_files()* function handle the reset all the sentiment analysis result route with several logics. First, remove all the uploaded, pre-processed, and classified implementation datasets inside the *datasets* directory. Second, reset all the session so the user cannot access the sentiment analysis result pages. Finally, the home page is displayed through redirecting to the home page with *redirect()* and *url_for()* function, passing the *render_index_view* route function.

```
{% extends 'layouts/base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row gy-5 mt-5">
        <div class="col-12 col-md-6 text-center text-md-start">
            <h1 class="fw-bold">Natural Language Processing</h1>
            <p class="fs-4 px-3 px-md-0">
                Refers to the branch of computer science—and more specifically, the branch of artificial
                intelligence or AI—concerned with giving computers the ability to understand text and spoken
                words in much the same way human beings can.
            </p>
        </div>
        <div class="col-12 col-md-6">
            <img src="{{ url_for('static', filename='img/natural-language-processing.png') }}"
                class="img-fluid d-none d-md-block" alt="natural-language-processing">
        </div>
    </div>

    <div class="row gy-5 mt-md-5">
        <div class="col-12 col-md-6 justify-content-center d-none d-md-block">
            <img src="{{ url_for('static', filename='img/sentiment-analysis.jpg') }}" class="img-fluid"
                alt="sentiment-analysis">
        </div>
        <div class="col-12 col-md-6 text-center text-md-start">
            <h1 class="fw-bold">Sentiment Analysis</h1>
            <p class="fs-5 px-3 px-md-0 mt-3">
                Sentiment analysis, also called opinion mining, is the field of study that
                analyzes people's opinions, sentiments, evaluations, appraisals, attitudes,
                and emotions towards entities such as products, services, organizations,
                individuals, issues, events, topics, and their attributes.
            </p>

        </div>
    </div>
</div>

<div id="carouselExampleSlidesOnly" class="carousel slide mt-5 carousel-3" data-bs-ride="carousel">
    <div class="carousel-inner">
        <div class="carousel-item active">
            <img src="{{ url_for('static', filename='img/carousel-3.avif') }}" class="d-block w-100" alt="carousel-3">
            <div class="carousel-caption text-center d-flex flex-column justify-content-center top-0">
                <h1 class="fw-bold">Which are types of COVID-19 vaccine topics that used in this
                    sentiment analysis?</h1>
                <p class="mt-4">The sentiment analysis topics are about Sinovac vaccine, AstraZeneca
                    vaccine, Pfizer vaccine, and Moderna vaccine</p>
            </div>
        </div>
    </div>
</div>

<div class="container mb-5">
    <div class="row gy-5 mt-5">
        <div class="col-12 text-center">
            <h1 class="fw-bold">What is Sinovac Vaccine?</h1>
            <p class="fs-4 px-3 px-md-0 mt-3 mt-md-5">
                Sinovac is a vaccine produced by a Chinese biopharmaceutical company, Sinovac BioTech. This vaccine,
                which has another name, CoronaVac, is a type of whole virus vaccine that uses the inactivated
                SARS-CoV-2 virus. The inactivated virus is no longer able to infect the body, but it can trigger the
                formation of immunity in our body.
            </p>
        </div>
        <div class="col-12 text-center">
            <h1 class="fw-bold">What is AstraZeneca Vaccine?</h1>
            <p class="fs-4 px-3 px-md-0 mt-3 mt-md-5">
                The AstraZeneca or Oxford-AstraZeneca vaccine is a vaccine produced by a British biopharmaceutical
                company and the University of Oxford. This vaccine is a type of viral vector vaccine that utilizes
                the chimpanzee Adenovirus (which is weakened until it's harmless), to deliver the spike protein from
                Covid-19 into our cells and trigger it to build antibodies.
            </p>
        </div>

        <div class="col-12 text-center">
            <h1 class="fw-bold">What is Pfizer Vaccine?</h1>
            <p class="fs-4 px-3 px-md-0 mt-3 mt-md-5">
                The Pfizer-BioNTech COVID-19 vaccine also contains a harmless piece of messenger RNA (mRNA). The
                COVID-19 mRNA teaches cells in the body how to create an immune response to the virus that causes
                COVID-19. This response helps protect you from getting sick with COVID-19 in the future.
            </p>
        </div>

        <div class="col-12 text-center">
            <h1 class="fw-bold">What is Moderna Vaccine?</h1>
            <p class="fs-4 px-3 px-md-0 mt-3 mt-md-5">
                CoronaVac is a COVID-19 vaccine produced by Sinovac Biotech, a China-based pharmaceutical
                company with headquarters in Beijing. The company focuses specifically on the development and
                manufacturing of vaccines to target human infectious diseases.
            </p>
        </div>
    </div>
</div>
{% endblock %}
```

Figure 3.141 The Sentiment Analysis Web Application Home Page Code

The Figure 3.141 shows the sentiment analysis web application for the home page. Based on the design, the home page content has several description or definition about the domains such as NLP, sentiment analysis, Sinovac vaccine, AstraZeneca vaccine, Pfizer vaccine, and Moderna vaccine.

```
{% extends 'layouts/base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row mt-5">
        <div class="col-12 col-md-6 col-lg-4 text-center mb-4">
            <div class="card shadow p-3">
                <div class="card-body">
                    <h3 class="card-title">
                        <i class="fa-solid fa-brain fa-3x"></i>
                    </h3>
                    <h3 class="fw-bold mt-3">AI Powered</h3>
                    <p class="card-text mt-3">
                        Sentiment analysis with deep learning models such as hybrid CNN-LSTM model,
                        single CNN
                        model, and single LSTM model
                    </p>
                </div>
            </div>
        </div>
        <div class="col-12 col-md-6 col-lg-4 text-center mb-4">
            <div class="card shadow p-4">
                <div class="card-body">
                    <h3 class="card-title">
                        <i class="fa-solid fa-user-gear fa-3x"></i>
                    </h3>
                    <h3 class="fw-bold mt-3">NLP Methods</h3>
                    <p class="card-text mt-3">
                        Sentiment analysis processed with several NLP methods such as tweet cleaning, expanding
                        contractions, removing punctuations, case-folding, negation handling, removing stopwords,
                        and word lemmatizing
                    </p>
                </div>
            </div>
        </div>
        <div class="col-12 col-md-6 col-lg-4 text-center mb-5">
            <div class="card shadow p-4">
                <div class="card-body">
                    <h3 class="card-title">
                        <i class="fa-brands fa-twitter fa-3x"></i>
                    </h3>
                    <h3 class="fw-bold mt-3">Twitter Data</h3>
                    <p class="card-text mt-3">
                        Sentiment analysis with Twitter data about four COVID-19 vaccine types such as Sinovac
                        tweets, AstraZeneca tweets, Pfizer tweets, and Moderna tweets crawled using Twitter API
                    </p>
                </div>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

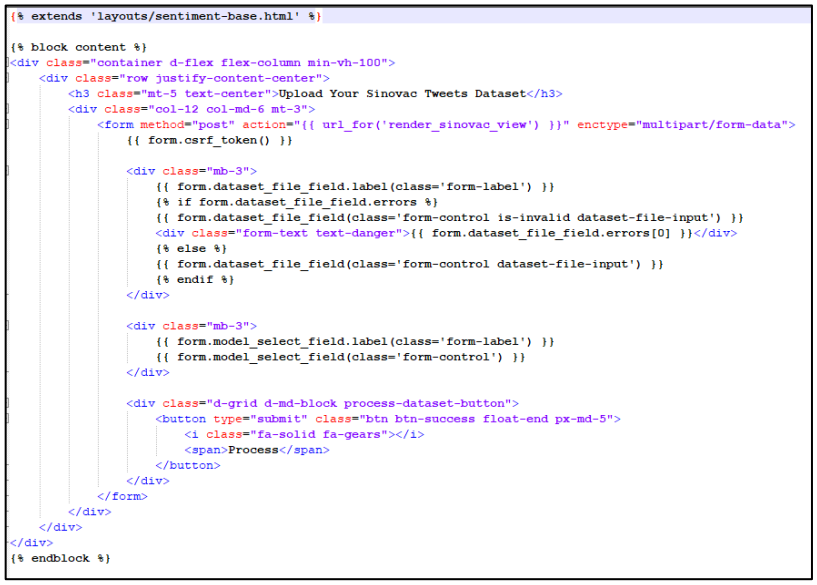
Figure 3.142 The Sentiment Analysis Web Application About Page Code

The Figure 3.142 shows the sentiment analysis web application for the about page. Based on the design, the about page content has several description or definition about the technologies, tools, or data used behind the application such as AI, NLP, and Twitter data.

```
{% extends 'layouts/sentiment-base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row">
        <div class="col-12 text-center ">
            <h3 class="mt-5">Welcome To The COVID-19 Vaccine Types Sentiment Analysis</h3>
            <p>Please choose one of the menu on the left side to start !</p>
        </div>
        <div class="col-12 col-md-7 mt-3">
            <ol class="list-group list-group-numbered">
                <li class="list-group-item">Choose your sentiment analysis domains on the left side menus
                </li>
                <li class="list-group-item">Upload your Twitter dataset according to the domain that you have chosen
                </li>
                <li class="list-group-item">Choose the model and click the process button and wait for a moment</li>
                <li class="list-group-item">See the sentiment analysis results</li>
            </ol>
        </div>
    </div>
</div>
{% endblock %}
```

Figure 3.143 The Sentiment Analysis Web Application Sentiment Analysis Page Code

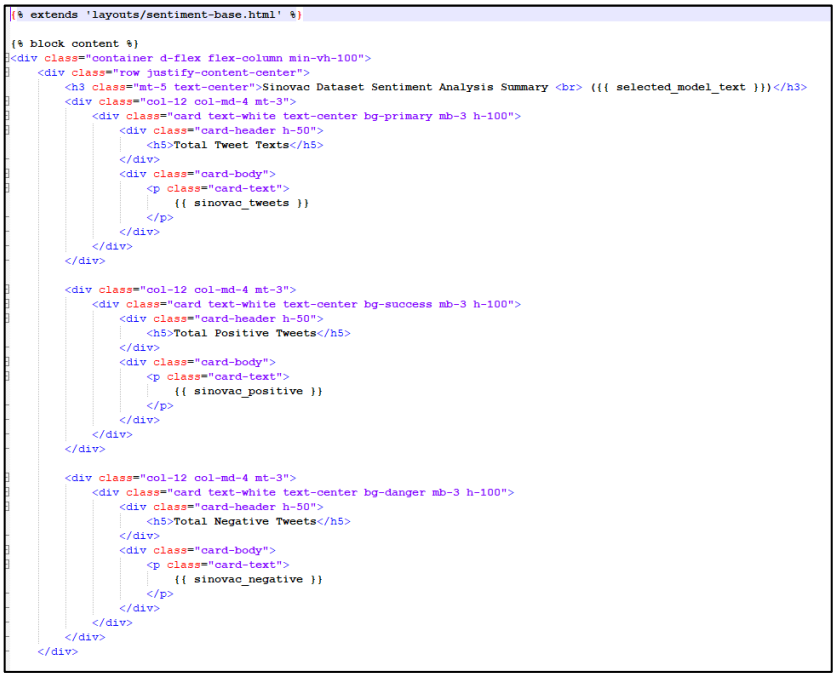
The Figure 3.143 shows the sentiment analysis web application for the sentiment analysis page. Based on the design, the sentiment analysis page content has instruction for the user to use the application. The instructions are listed with list group item.



```
{% extends 'layouts/sentiment-base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row justify-content-center">
        <h3 class="mt-5 text-center">Upload Your Sinovac Tweets Dataset</h3>
        <div class="col-12 col-md-6 mt-3">
            <form method="post" action="{{ url_for('render_sinovac_view') }}" enctype="multipart/form-data">
                {{ form.csrf_token() }}

                <div class="mb-3">
                    {{ form.dataset_file_field.label(class='form-label') }}
                    {% if form.dataset_file_field.errors %}
                    {{ form.dataset_file_field(class='form-control is-invalid dataset-file-input') }}
                    <div class="form-text text-danger">{{ form.dataset_file_field.errors[0] }}</div>
                    {% else %}
                    {{ form.dataset_file_field(class='form-control dataset-file-input') }}
                    {% endif %}
                </div>

                <div class="mb-3">
                    {{ form.model_select_field.label(class='form-label') }}
                    {{ form.model_select_field(class='form-control') }}
                </div>

                <div class="d-grid d-md-block process-dataset-button">
                    <button type="submit" class="btn btn-success float-end px-md-5">
                        <i class="fa-solid fa-gears"></i>
                        <span>Process</span>
                    </button>
                </div>
            </form>
        </div>
    </div>
</div>
{% endblock %}
```

Figure 3.144 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Page Code

The Figure 3.144 shows the sentiment analysis web application for the Sinovac sentiment analysis page. Based on the design, the Sinovac sentiment analysis page content has the file input field and the selection input field for the user to upload the Sinovac implementation dataset and choose the sentiment analysis model. Both the file input field and the selection input field are built with *flask-wtf* that defined in the *app.py*. The *multipart/form-data* is defined so the form can process file upload.



```
{% extends 'layouts/sentiment-base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row justify-content-center">
        <h3 class="mt-5 text-center">Sinovac Dataset Sentiment Analysis Summary <br> ({{ selected_model_text }})</h3>
        <div class="col-12 col-md-4 mt-3">
            <div class="card text-white text-center bg-primary mb-3 h-100">
                <div class="card-header h-50">
                    <h5>Total Tweet Texts</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">
                        {{ sinovac_tweets }}
                    </p>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-4 mt-3">
            <div class="card text-white text-center bg-success mb-3 h-100">
                <div class="card-header h-50">
                    <h5>Total Positive Tweets</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">
                        {{ sinovac_positive }}
                    </p>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-4 mt-3">
            <div class="card text-white text-center bg-danger mb-3 h-100">
                <div class="card-header h-50">
                    <h5>Total Negative Tweets</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">
                        {{ sinovac_negative }}
                    </p>
                </div>
            </div>
        </div>
    </div>
```

Figure 3.145 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Result Page Code Part 1

The Figure 3.145 shows the sentiment analysis web application for the Sinovac sentiment analysis result page. Based on the design, the first part of the Sinovac sentiment analysis result page content has three data panels. The first panel presents the number of Sinovac tweets with *sinovac_tweets* variable passed from the route. The second panel presents the number of Sinovac positive tweets with *sinovac_positive* variable passed from the route. The third panel presents the number of Sinovac negative tweets with *sinovac_negative* variable passed from the route.

```
<div class="row justify-content-center mt-3">
    <h3 class="mt-5 text-center">Prediction Results Examples</h3>
    <div class="col-12 col-md-12 mt-3">
        <div class="card mb-3 shadow">
            <div class="card-body">
                <table class="table table-hover">
                    <thead>
                        <tr>
                            <th scope="col">#</th>
                            <th scope="col">Tweet Texts</th>
                            <th scope="col">Sentiment</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for index, data in sinovac_data.iterrows() %}
                        <tr>
                            <th scope="row">{{ index+1 }}</th>
                            <td>{{ data['text'] }}</td>
                            <td>
                                {% if data['sentiment'] == 'Positive' %}
                                <span class="badge bg-success">{{ data['sentiment'] }}</span>
                                {% else %}
                                <span class="badge bg-danger">{{ data['sentiment'] }}</span>
                                {% endif %}
                            </td>
                        </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
```

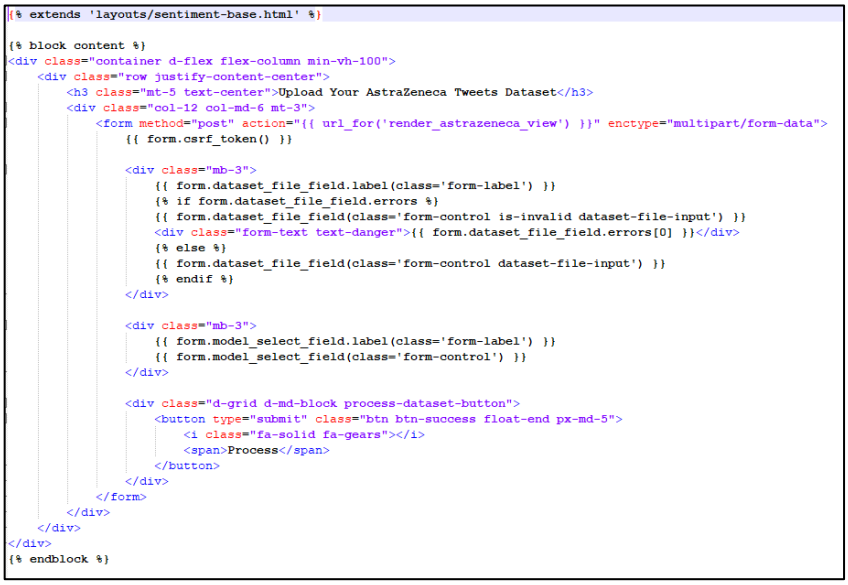
Figure 3.146 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Result Page Code Part 2

The Figure 3.146 shows the sentiment analysis web application for the Sinovac sentiment analysis result page. Based on the design, the second part of the Sinovac sentiment analysis result page content has prediction result table. The prediction result table displays the ten examples classified sentiment of the Sinovac implementation dataset with *for* loop.

```
    <div class="row justify-content-center mt-3 mb-3">
        <h3 class="mt-5 text-center">Sentiment Analysis Visualization</h3>
        <div class="col-12 col-md-6 mt-3">
            <div class="card text-center mb-3 shadow">
                <div class="card-body">
                    <div id="sinovac-bar-plot"></div>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-6 mt-3">
            <div class="card text-center mb-3 shadow">
                <div class="card-body">
                    <div id="sinovac-pie-plot"></div>
                </div>
            </div>
        </div>
    </div>
</div>

<script src="https://cdn.plot.ly/plotly-2.11.1.min.js"></script>
<script>
    const sinovac_bar_graph = {{ sinovac_bar_plot | safe }};
    Plotly.newPlot('sinovac-bar-plot', sinovac_bar_graph, {}, { responsive: true });

    const sinovac_pie_graph = {{ sinovac_pie_plot | safe }};
    Plotly.newPlot('sinovac-pie-plot', sinovac_pie_graph, {}, { responsive: true });
</script>
{% endblock %}
```

Figure 3.147 The Sentiment Analysis Web Application Sinovac Sentiment Analysis Result Page Code Part 3

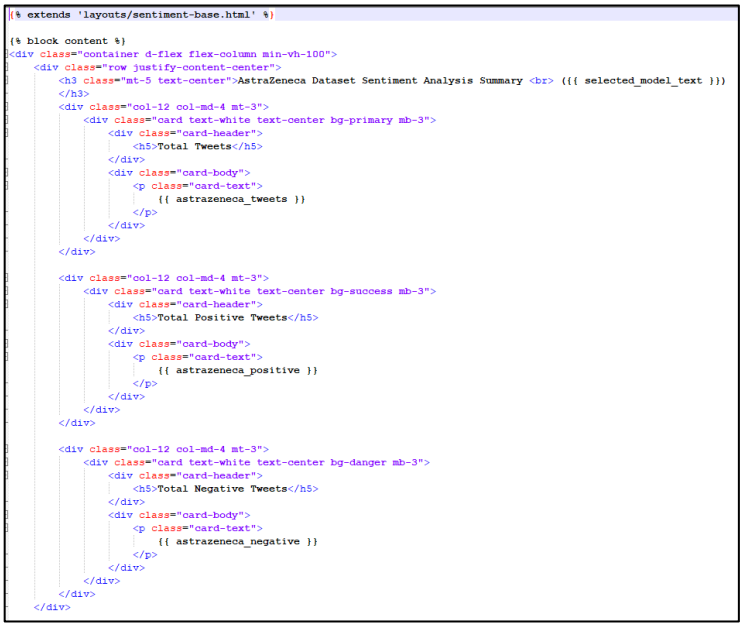
The Figure 3.147 shows the sentiment analysis web application for the Sinovac sentiment analysis result page. Based on the design, the third part of the Sinovac sentiment analysis result page content has sentiment analysis visualization. The sentiment analysis visualization visualizes the classified Sinovac implementation data into bar plot and pie plot using the *plotly* library.



```
{% extends 'layouts/sentiment-base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row justify-content-center">
        <h3 class="mt-5 text-center">Upload Your AstraZeneca Tweets Dataset</h3>
        <div class="col-12 col-md-6 mt-3">
            <form method="post" action="{{ url_for('render_astrazeneca_view') }}" enctype="multipart/form-data">
                {{ form.csrf_token() }}

                <div class="mb-3">
                    {{ form.dataset_file_field.label(class='form-label') }}
                    {% if form.dataset_file_field.errors %}
                    {{ form.dataset_file_field(class='form-control is-invalid dataset-file-input') }}
                    <div class="form-text text-danger">{{ form.dataset_file_field.errors[0] }}</div>
                    {% else %}
                    {{ form.dataset_file_field(class='form-control dataset-file-input') }}
                    {% endif %}
                </div>

                <div class="mb-3">
                    {{ form.model_select_field.label(class='form-label') }}
                    {{ form.model_select_field(class='form-control') }}
                </div>

                <div class="d-grid d-md-block process-dataset-button">
                    <button type="submit" class="btn btn-success float-end px-md-5">
                        <i class="fa-solid fa-gears"></i>
                        <span>Process</span>
                    </button>
                </div>
            </form>
        </div>
    </div>
</div>
{% endblock %}
```

Figure 3.148 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Page Code

The Figure 3.148 shows the sentiment analysis web application for the AstraZeneca sentiment analysis page. Based on the design, the AstraZeneca sentiment analysis page content has the file input field and the selection input field for the user to upload the AstraZeneca implementation dataset and choose the sentiment analysis model. Both the file input field and the selection input field are built with *flask-wtf* that defined in the *app.py*. The *multipart/form-data* is defined so the form can process file upload.



```
{% extends 'layouts/sentiment-base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row justify-content-center">
        <h3 class="mt-5 text-center">AstraZeneca Dataset Sentiment Analysis Summary <br> ({{ selected_model_text }})
        </h3>
        <div class="col-12 col-md-4 mt-3">
            <div class="card text-white text-center bg-primary mb-3">
                <div class="card-header">
                    <h5>Total Tweets</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">
                        {{ astrazeneca_tweets }}
                    </p>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-4 mt-3">
            <div class="card text-white text-center bg-success mb-3">
                <div class="card-header">
                    <h5>Total Positive Tweets</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">
                        {{ astrazeneca_positive }}
                    </p>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-4 mt-3">
            <div class="card text-white text-center bg-danger mb-3">
                <div class="card-header">
                    <h5>Total Negative Tweets</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">
                        {{ astrazeneca_negative }}
                    </p>
                </div>
            </div>
        </div>
    </div>
```

Figure 3.149 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Result Page Code Part 1

The Figure 3.149 shows the sentiment analysis web application for the AstraZeneca sentiment analysis result page. Based on the design, the first part of the AstraZeneca sentiment analysis result page content has three data panels. The first panel presents the number of AstraZeneca tweets with *astrazeneca_tweets* variable passed from the route. The second panel presents the number of AstraZeneca positive tweets with *astrazeneca_positive* variable passed from the route. The third panel presents the number of AstraZeneca negative tweets with *astrazeneca_negative* variable passed from the route.

```
<div class="row justify-content-center mt-5">
    <h3 class="mt-5 text-center">Prediction Results</h3>
    <div class="col-12 col-md-12 mt-3">
        <div class="card mb-3 shadow">
            <div class="card-body">
                <table class="table table-hover">
                    <thead>
                        <tr>
                            <th scope="col">#</th>
                            <th scope="col">Tweet Texts</th>
                            <th scope="col">Sentiment</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for index, data in astrazeneca_data.iterrows() %}
                        <tr>
                            <th scope="row">{{ index+1 }}</th>
                            <td>{{ data['text'] }}</td>
                            <td>
                                {% if data['sentiment'] == 'Positive' %}
                                <span class="badge bg-success">{{ data['sentiment'] }}</span>
                                {% else %}
                                <span class="badge bg-danger">{{ data['sentiment'] }}</span>
                                {% endif %}
                            </td>
                        </tr>
                        {% endfor %}
                    </tbody>
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
```

Figure 3.150 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Result Page Code Part 2

The Figure 3.150 shows the sentiment analysis web application for the AstraZeneca sentiment analysis result page. Based on the design, the second part of the AstraZeneca sentiment analysis result page content has prediction result table. The prediction result table displays the ten examples classified sentiment of the AstraZeneca implementation dataset with *for* loop.

```
    <div class="row justify-content-center mt-5">
        <h3 class="mt-5 text-center">Sentiment Analysis Visualization</h3>
        <div class="col-12 col-md-6 mt-3">
            <div class="card text-center mb-3 shadow">
                <div class="card-body">
                    <div id="astrazeneca-bar-plot"></div>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-6 mt-3">
            <div class="card text-center mb-3 shadow">
                <div class="card-body">
                    <div id="astrazeneca-pie-plot"></div>
                </div>
            </div>
        </div>
    </div>
</div>

<script src="https://cdn.plot.ly/plotly-2.11.1.min.js"></script>
<script>
    const astrazeneca_bar_graph = {{ astrazeneca_bar_plot | safe }};
    Plotly.newPlot('astrazeneca-bar-plot', astrazeneca_bar_graph, {}, { responsive: true });

    const astrazeneca_pie_graph = {{ astrazeneca_pie_plot | safe }};
    Plotly.newPlot('astrazeneca-pie-plot', astrazeneca_pie_graph, {}, { responsive: true });
</script>
{% endblock %}
```

Figure 3.151 The Sentiment Analysis Web Application AstraZeneca Sentiment Analysis Result Page Code Part 3

The Figure 3.151 shows the sentiment analysis web application for the AstraZeneca sentiment analysis result page. Based on the design, the third part of the AstraZeneca sentiment analysis result page content has sentiment analysis visualization. The sentiment analysis visualization visualizes the classified AstraZeneca implementation data into bar plot and pie plot using the *plotly* library.

```
{% extends 'layouts/sentiment-base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row justify-content-center">
        <h3 class="mt-5 text-center">Upload Your Pfizer Tweets Dataset</h3>
        <div class="col-12 col-md-6 mt-3">
            <form method="post" action="{{ url_for('render_pfizer_view') }}" enctype="multipart/form-data">
                {{ form.csrf_token() }}

                <div class="mb-3">
                    {{ form.dataset_file_field.label(class='form-label') }}
                    {% if form.dataset_file_field.errors %}
                    {{ form.dataset_file_field(class='form-control is-invalid dataset-file-input') }}
                    <div class="form-text text-danger">{{ form.dataset_file_field.errors[0] }}</div>
                    {% else %}
                    {{ form.dataset_file_field(class='form-control dataset-file-input') }}
                    {% endif %}
                </div>

                <div class="mb-3">
                    {{ form.model_select_field.label(class='form-label') }}
                    {{ form.model_select_field(class='form-control') }}
                </div>

                <div class="d-grid d-md-block process-dataset-button">
                    <button type="submit" class="btn btn-success float-end px-md-5">
                        <i class="fa-solid fa-gears"></i>
                        <span>Process</span>
                    </button>
                </div>
            </form>
        </div>
    </div>
</div>
{% endblock %}
```

Figure 3.152 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Page Code

The Figure 3.152 shows the sentiment analysis web application for the Pfizer sentiment analysis page. Based on the design, the Pfizer sentiment analysis page content has the file input field and the selection input field for the user to upload the Pfizer implementation dataset and choose the sentiment analysis model.

Both the file input field and the selection input field are built with *flask-wtf* that defined in the *app.py*. The *multipart/form-data* is defined so the form can process file upload.

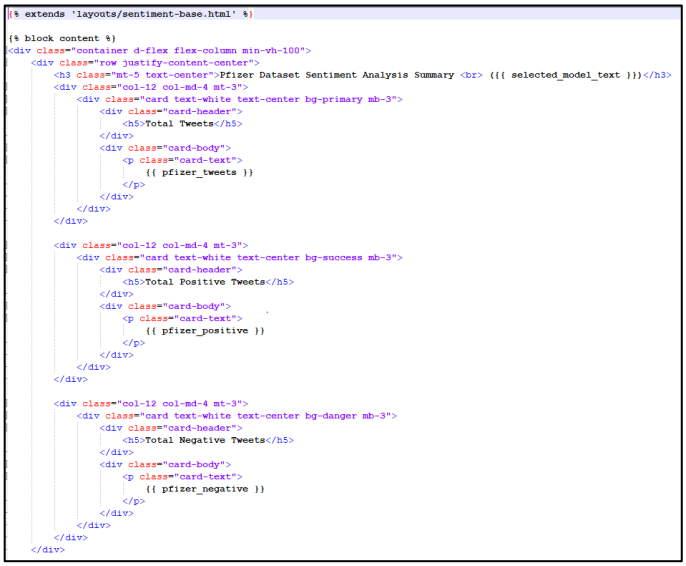


Figure 3.153 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Result Page Code Part 1

The Figure 3.153 shows the sentiment analysis web application for the Pfizer sentiment analysis result page. Based on the design, the first part of the Pfizer sentiment analysis result page content has three data panels. The first panel presents the number of Pfizer tweets with *pfizer_tweets* variable passed from the route. The second panel presents the number of Pfizer positive tweets with *pfizer_positive* variable passed from the route. The third panel presents the number of Pfizer negative tweets with *pfizer_negative* variable passed from the route.

```
<div class="row justify-content-center mt-5">
    <h3 class="mt-5 text-center">Prediction Results</h3>
    <div class="col-12 col-md-12 mt-3">
        <div class="card mb-3 shadow">
            <div class="card-body">
                <table class="table table-hover">
                    <thead>
                        <tr>
                            <th scope="col">#</th>
                            <th scope="col">Tweet Texts</th>
                            <th scope="col">Sentiment</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for index, data in pfizer_data.iterrows() %}
                        <tr>
                            <th scope="row">{{ index+1 }}</th>
                            <td>{{ data['text'] }}</td>
                            <td>
                                {% if data['sentiment'] == 'Positive' %}
                                <span class="badge bg-success">{{ data['sentiment'] }}</span>
                                {% else %}
                                <span class="badge bg-danger">{{ data['sentiment'] }}</span>
                                {% endif %}
                            </td>
                        </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
```

Figure 3.154 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Result Page Code Part 2

The Figure 3.154 shows the sentiment analysis web application for the Pfizer sentiment analysis result page. Based on the design, the second part of the Pfizer sentiment analysis result page content has prediction result table. The prediction result table displays the ten examples classified sentiment of the Pfizer implementation dataset with *for* loop.

```
    <div class="row justify-content-center mt-5">
        <h3 class="mt-5 text-center">Sentiment Analysis Visualization</h3>
        <div class="col-12 col-md-6 mt-3">
            <div class="card text-center mb-3 shadow">
                <div class="card-body">
                    <div id="pfizer-bar-plot"></div>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-6 mt-3">
            <div class="card text-center mb-3 shadow">
                <div class="card-body">
                    <div id="pfizer-pie-plot"></div>
                </div>
            </div>
        </div>
    </div>
</div>

<script src="https://cdn.plot.ly/plotly-2.11.1.min.js"></script>
<script>
    const pfizer_bar_graph = {{ pfizer_bar_plot | safe }};
    Plotly.newPlot('pfizer-bar-plot', pfizer_bar_graph, {}, { responsive: true });

    const pfizer_pie_graph = {{ pfizer_pie_plot | safe }};
    Plotly.newPlot('pfizer-pie-plot', pfizer_pie_graph, {}, { responsive: true });
</script>
{% endblock %}
```

Figure 3.155 The Sentiment Analysis Web Application Pfizer Sentiment Analysis Result Page Code Part 3

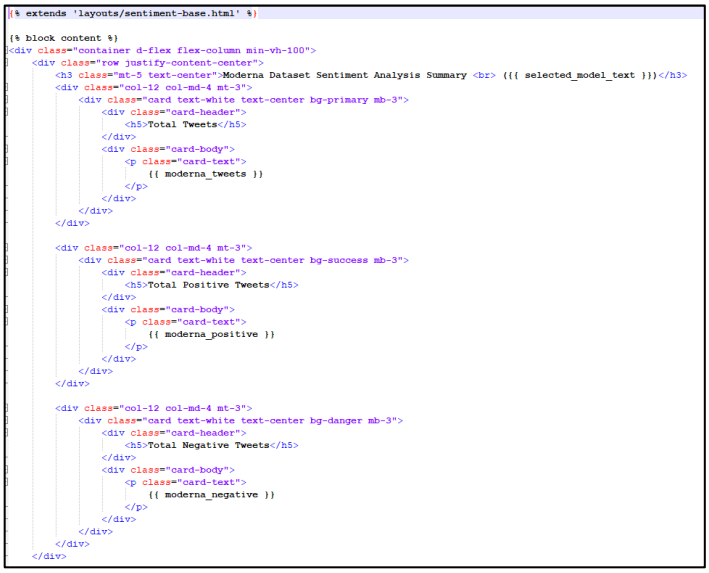
The Figure 3.151 shows the sentiment analysis web application for the Pfizer sentiment analysis result page. Based on the design, the third part of the

Pfizer sentiment analysis result page content has sentiment analysis visualization. The sentiment analysis visualization visualizes the classified Pfizer implementation data into bar plot and pie plot using the *plotly* library.

```
{% extends 'layouts/sentiment-base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row justify-content-center">
        <h3 class="mt-5 text-center">Upload Your Moderna Tweets Dataset</h3>
        <div class="col-12 col-md-6 mt-3">
            <form method="post" action="{{ url_for('render_moderna_view') }}" enctype="multipart/form-data">
                {{ form.csrf_token() }}

                <div class="mb-3">
                    {{ form.dataset_file_field.label(class='form-label') }}
                    {% if form.dataset_file_field.errors %}
                    {{ form.dataset_file_field(class='form-control is-invalid dataset-file-input') }}
                    <div class="form-text text-danger">{{ form.dataset_file_field.errors[0] }}</div>
                    {% else %}
                    {{ form.dataset_file_field(class='form-control dataset-file-input') }}
                    {% endif %}
                </div>

                <div class="mb-3">
                    {{ form.model_select_field.label(class='form-label') }}
                    {{ form.model_select_field(class='form-control') }}
                </div>

                <div class="d-grid d-md-block process-dataset-button">
                    <button type="submit" class="btn btn-success float-end px-md-5">
                        <i class="fa-solid fa-gears"></i>
                        <span>Process</span>
                    </button>
                </div>
            </form>
        </div>
    </div>
</div>
{% endblock %}
```

Figure 3.156 The Sentiment Analysis Web Application Moderna Sentiment Analysis Page Code

The Figure 3.156 shows the sentiment analysis web application for the Moderna sentiment analysis page. Based on the design, the Moderna sentiment analysis page content has the file input field and the selection input field for the user to upload the Moderna implementation dataset and choose the sentiment analysis model. Both the file input field and the selection input field are built with *flask-wtf* that defined in the *app.py*. The *multipart/form-data* is defined so the form can process file upload.



```
{% extends 'layouts/sentiment-base.html' %}

{% block content %}
<div class="container d-flex flex-column min-vh-100">
    <div class="row justify-content-center">
        <h3 class="mt-5 text-center">Moderna Dataset Sentiment Analysis Summary <br> ({{ selected_model_text }})</h3>
        <div class="col-12 col-md-4 mt-3">
            <div class="card text-white text-center bg-primary mb-3">
                <div class="card-header">
                    <h5>Total Tweets</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">
                        {{ moderna_tweets }}
                    </p>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-4 mt-3">
            <div class="card text-white text-center bg-success mb-3">
                <div class="card-header">
                    <h5>Total Positive Tweets</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">
                        {{ moderna_positive }}
                    </p>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-4 mt-3">
            <div class="card text-white text-center bg-danger mb-3">
                <div class="card-header">
                    <h5>Total Negative Tweets</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">
                        {{ moderna_negative }}
                    </p>
                </div>
            </div>
        </div>
    </div>
```

Figure 3.157 The Sentiment Analysis Web Application Moderna Sentiment Analysis Result Page Code Part 1

The Figure 3.157 shows the sentiment analysis web application for the Moderna sentiment analysis result page. Based on the design, the first part of the Moderna sentiment analysis result page content has three data panels. The first panel presents the number of Moderna tweets with *moderna_tweets* variable passed from the route. The second panel presents the number of Moderna positive tweets with *moderna_positive* variable passed from the route. The third panel presents the number of Moderna negative tweets with *moderna_negative* variable passed from the route.

```
<div class="row justify-content-center mt-5">
    <h3 class="mt-5 text-center">Prediction Results</h3>
    <div class="col-12 col-md-12 mt-3">
        <div class="card mb-3 shadow">
            <div class="card-body">
                <table class="table table-hover">
                    <thead>
                        <tr>
                            <th scope="col">#</th>
                            <th scope="col">Tweet Texts</th>
                            <th scope="col">Sentiment</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for index, data in moderna_data.iterrows() %}
                        <tr>
                            <th scope="row">{{ index+1 }}</th>
                            <td>{{ data['text'] }}</td>
                            <td>
                                {% if data['sentiment'] == 'Positive' %}
                                <span class="badge bg-success">{{ data['sentiment'] }}</span>
                                {% else %}
                                <span class="badge bg-danger">{{ data['sentiment'] }}</span>
                                {% endif %}
                            </td>
                        </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
```

Figure 3.158 The Sentiment Analysis Web Application Moderna Sentiment Analysis Result Page Code Part 2

The Figure 3.158 shows the sentiment analysis web application for the Moderna sentiment analysis result page. Based on the design, the second part of the Moderna sentiment analysis result page content has prediction result table. The prediction result table displays the ten examples classified sentiment of the Moderna implementation dataset with *for* loop.

```
    <div class="row justify-content-center mt-5">
        <h3 class="mt-5 text-center">Sentiment Analysis Visualization</h3>
        <div class="col-12 col-md-6 mt-3">
            <div class="card text-center mb-3 shadow">
                <div class="card-body">
                    <div id="moderna-bar-plot"></div>
                </div>
            </div>
        </div>

        <div class="col-12 col-md-6 mt-3">
            <div class="card text-center mb-3 shadow">
                <div class="card-body">
                    <div id="moderna-pie-plot"></div>
                </div>
            </div>
        </div>
    </div>
</div>

<script src="https://cdn.plot.ly/plotly-2.11.1.min.js"></script>
<script>
    const moderna_bar_graph = {{ moderna_bar_plot | safe }};
    Plotly.newPlot('moderna-bar-plot', moderna_bar_graph, {}, { responsive: true });

    const moderna_pie_graph = {{ moderna_pie_plot | safe }};
    Plotly.newPlot('moderna-pie-plot', moderna_pie_graph, {}, { responsive: true });
</script>
{% endblock %}
```

Figure 3.159 The Sentiment Analysis Web Application Moderna Sentiment Analysis Result Page Code Part 3

The Figure 3.158 shows the sentiment analysis web application for the Moderna sentiment analysis result page. Based on the design, the third part of the Moderna sentiment analysis result page content has sentiment analysis visualization. The sentiment analysis visualization visualizes the classified Moderna implementation data into bar plot and pie plot using the *plotly* library.

3.5.4 Testing

The testing of the second increment is to test and evaluate the sentiment analysis web application functions. The basic black-box testing table will be used to test each function by define the test case name, the input test, the expected output, the test result, and the summary.

Table 3.7 The Sentiment Analysis Web Application Black-Box Testing Table

Test Scenario	***Test Case***	***Expected Output***	***Test Result***	***Summary***
Home Page	*Open the home page by clicking the home menu in navigation bar*	*Show the home page*		

About Page	*Open the about page by clicking the about menu in navigation bar*	*Show the about page*
Sentiment Analysis Page	*Open the about page by clicking the sentiment analysis menu in navigation bar*	*Show the sentiment analysis welcome page*
Sinovac Sentiment Analysis Page	*Open the Sinovac sentiment analysis page by clicking the Sinovac menu in sidebar*	*Show the Sinovac sentiment analysis page*
Sinovac Sentiment Analysis Form Validation	*Make the file input field empty and upload a non-CSV format*	*Show the form validation error message*
Sinovac Sentiment Analysis Result Page	*Upload the proper Sinovac dataset file, choose the sentiment analysis model, and clicked the process button*	*Show the Sinovac sentiment analysis result page with overview panels, example of classified sentiment, and the visualization*
AstraZeneca Sentiment Analysis Page	*Open the AstraZeneca sentiment analysis page by clicking the AstraZeneca menu in sidebar*	*Show the AstraZeneca sentiment analysis page*

AstraZeneca Sentiment Analysis Form Validation	*Make the file input field empty and upload a non-CSV format*	*Show the form validation error message*
AstraZeneca Sentiment Analysis Result Page	*Upload the proper AstraZeneca dataset file, choose the sentiment analysis model, and clicked the process button*	*Show the AstraZeneca sentiment analysis result page with overview panels, example of classified sentiment, and the visualization*
Pfizer Sentiment Analysis Page	*Open the Pfizer sentiment analysis page by clicking the Pfizer menu in sidebar*	*Show the Pfizer sentiment analysis page*
Pfizer Sentiment Analysis Form Validation	*Make the file input field empty and upload a non-CSV format*	*Show the form validation error message*
Pfizer Sentiment Analysis Result Page	*Upload the proper Pfizer dataset file, choose the sentiment analysis model, and clicked the process button*	*Show the Pfizer sentiment analysis result page with overview panels, example of classified sentiment, and the visualization*
Moderna Sentiment Analysis Page	*Open the Moderna sentiment analysis page by clicking the*	*Show the Moderna sentiment analysis page*

	Moderna menu in sidebar	
Moderna Sentiment Analysis Form Validation	*Make the file input field empty and upload a non-CSV format*	*Show the form validation error message*
Moderna Sentiment Analysis Result Page	*Upload the proper Moderna dataset file, choose the sentiment analysis model, and clicked the process button*	*Show the Moderna sentiment analysis result page with overview panels, example of classified sentiment, and the visualization*
Reset Sentiment Analysis Result and Back to the Home Page	*Reset all the sentiment analysis result by clicking the back to home menu in sidebar*	*Delete all the uploaded, pre-processed, and classified data and file, then open the home page*

CHAPTER 4
RESULT AND DISCUSSION

4.1 Incremental 1

This section covers the result from the first increment processes that are the dataset crawling process, the dataset pre-processing process, the dataset labeling process, the dataset splitting process, the dataset augmentation process, the training result of the sentiment analysis model process, and the testing or evaluation of the sentiment analysis model process. Then, this section also discussed about the analysis on the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model training and testing result by comparing them.

4.1.1 Dataset

The first increment processes are begun with the Twitter data crawling process. The Figure 4.1 is the result of the Twitter data crawling for each tweet about COVID-19 vaccine type saved into CSV format. The file naming format is the topic name and followed by the order of the weekly crawled, for example, “sinovac1.csv” means crawled tweets about Sinovac vaccine on the first week.

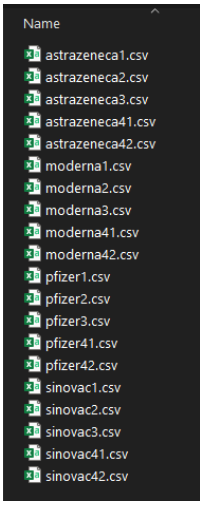


Figure 4.1 Twitter Data Crawling Process Result

The next process is to merge the crawled Twitter data on the Figure 4.1 into four dataset topics based on their topic. The Table 4.1 shows the merged Twitter data result, where there are four raw datasets: Sinovac dataset, AstraZeneca dataset,

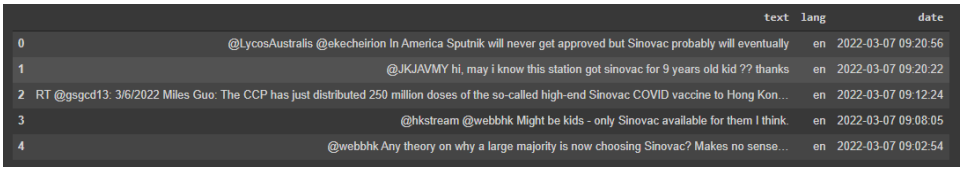
Pfizer dataset, and Moderna dataset; with the total of crawled tweets described with total rows. The Table 4.1 also describes that the Sinovac dataset has 9697 raw tweets, the AstraZeneca dataset has 8651 raw tweets, the Pfizer dataset has 10000 raw tweets, and Moderna dataset has 10000 raw tweets.

Table 4.1 The Number of Tweets of The Dataset Merging Process Result

No.	***Dataset***	***Number of Tweets***
1	*Sinovac*	*9697*
2	*AstraZeneca*	*8651*
3	*Pifzer*	*10000*
4	*Moderna*	*10000*

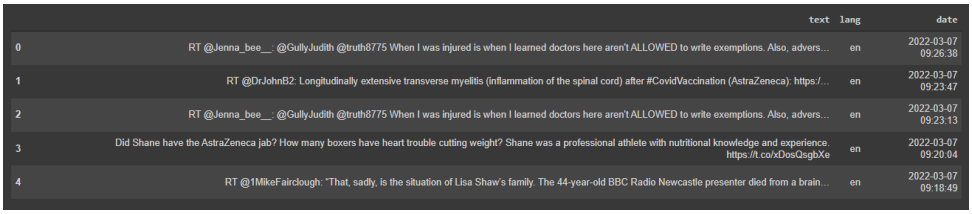
In addition, the Figure 4.2, Figure 4.3, Figure 4.4, and Figure 4.5 shows the example results from the merged Sinovac dataset, AstraZeneca dataset, Pfizer dataset, and Moderna dataset. Unfortunately, there is a limitation from Twitter API, where only the retweets are truncated with "…" symbol, therefore there are several tweet texts that are not in full text. The truncated tweets are calculated for each four dataset topics and resulting in 15% truncated tweets in the Sinovac dataset, 23% truncated tweets in the AstraZeneca dataset, 31% truncated tweets in the Pfizer dataset, and 21% truncated tweets in the Moderna dataset. From the calculated

truncated tweets, I decided to keep the all the four dataset topics because there are not too much truncated text (the threshold is 70%).



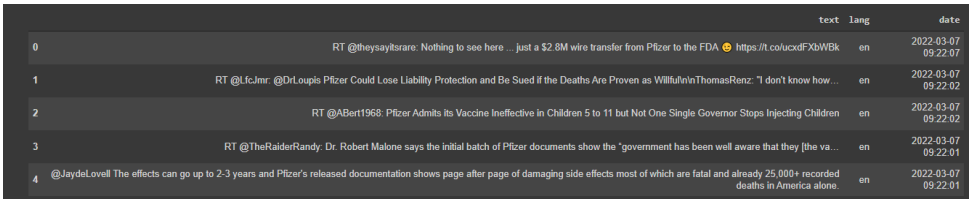
	text	lang	date
0	@LycosAustralis @ekecheirion In America Sputnik will never get approved but Sinovac probably will eventually	en	2022-03-07 09:20:56
1	@JKJAVMY hi, may i know this station got sinovac for 9 years old kid ?? thanks	en	2022-03-07 09:20:22
2	RT @gsgcd13: 3/6/2022 Miles Guo: The CCP has just distributed 250 million doses of the so-called high-end Sinovac COVID vaccine to Hong Kon…	en	2022-03-07 09:12:24
3	@hkstream @webbhk Might be kids - only Sinovac available for them I think.	en	2022-03-07 09:08:05
4	@webbhk Any theory on why a large majority is now choosing Sinovac? Makes no sense…	en	2022-03-07 09:02:54

Figure 4.2 The Merged Sinovac Dataset Result



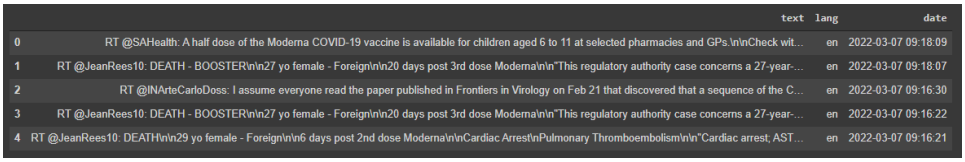
	text	lang	date
0	RT @Jenna_bee__: @GullyJudith @truth8775 When I was injured is when I learned doctors here aren't ALLOWED to write exemptions. Also, advers…	en	2022-03-07 09:26:38
1	RT @DrJohnB2: Longitudinally extensive transverse myelitis (inflammation of the spinal cord) after #CovidVaccination (AstraZeneca): https:/…	en	2022-03-07 09:23:47
2	RT @Jenna_bee__: @GullyJudith @truth8775 When I was injured is when I learned doctors here aren't ALLOWED to write exemptions. Also, advers…	en	2022-03-07 09:23:13
3	Did Shane have the AstraZeneca jab? How many boxers have heart trouble cutting weight? Shane was a professional athlete with nutritional knowledge and experience. https://t.co/xDosQsgbXe	en	2022-03-07 09:20:04
4	RT @1MikeFairclough: "That, sadly, is the situation of Lisa Shaw's family. The 44-year-old BBC Radio Newcastle presenter died from a brain…	en	2022-03-07 09:18:49

Figure 4.3 The Merged AstraZeneca Dataset Result



	text	lang	date
0	RT @theysayitsrare: Nothing to see here ... just a $2.8M wire transfer from Pfizer to the FDA 😉 https://t.co/ucxdFXbWBk	en	2022-03-07 09:22:07
1	RT @LfcJmr: @DrLoupis Pfizer Could Lose Liability Protection and Be Sued if the Deaths Are Proven as Willful\n\nThomasRenz: "I don't know how…	en	2022-03-07 09:22:02
2	RT @ABert1968: Pfizer Admits its Vaccine Ineffective in Children 5 to 11 but Not One Single Governor Stops Injecting Children	en	2022-03-07 09:22:02
3	RT @TheRaiderRandy: Dr. Robert Malone says the initial batch of Pfizer documents show the "government has been well aware that they [the va…	en	2022-03-07 09:22:01
4	@JaydeLovell The effects can go up to 2-3 years and Pfizer's released documentation shows page after page of damaging side effects most of which are fatal and already 25,000+ recorded deaths in America alone.	en	2022-03-07 09:22:01

Figure 4.4 The Merged Pfizer Dataset Result



	text	lang	date
0	RT @SAHealth: A half dose of the Moderna COVID-19 vaccine is available for children aged 6 to 11 at selected pharmacies and GPs.\n\nCheck wit…	en	2022-03-07 09:18:09
1	RT @JeanRees10: DEATH - BOOSTER\n\n27 yo female - Foreign\n\n20 days post 3rd dose Moderna\n\n"This regulatory authority case concerns a 27-year-…	en	2022-03-07 09:18:07
2	RT @INArteCarloDoss: I assume everyone read the paper published in Frontiers in Virology on Feb 21 that discovered that a sequence of the C…	en	2022-03-07 09:16:30
3	RT @JeanRees10: DEATH - BOOSTER\n\n27 yo female - Foreign\n\n20 days post 3rd dose Moderna\n\n"This regulatory authority case concerns a 27-year-…	en	2022-03-07 09:16:22
4	RT @JeanRees10: DEATH\n\n29 yo female - Foreign\n\n6 days post 2nd dose Moderna\n\nCardiac Arrest\nPulmonary Thromboembolism\n\n"Cardiac arrest; AST…	en	2022-03-07 09:16:21

Figure 4.5 The Merged Moderna Dataset Result

After crawled and retrieved the Twitter data for each COVID-19 vaccine type topic, the dataset filtering process is implemented to make sure the dataset only in March 2022 period. The Table 4.2 shows the number of tweets from the four dataset topics is decreasing after I filtered the period. This imply that there are some tweets that are not in March 2022 period. The Table 4.2 also describes after they are filtered, the Sinovac dataset has 9270 raw tweets, the AstaZeneca dataset has 8443 raw tweets, the Pfizer dataset has 8000 raw tweets, and the Moderna dataset has 9249 raw tweets.

Table 4.2 The After Dataset Filtering Process Result

No.	***Dataset***	***Date Before Filter***	***Date After Filter***	***Number of Tweets After Filtering***
1	*Sinovac*	*27 Feb – 01 Apr 2022*	*01 Mar – 31 Mar 2022*	*9270*
2	*AstraZeneca*	*04 Mar – 01 Apr 2022*	*04 Mar – 31 Mar 2022*	*8443*
3	*Pifzer*	*07 Mar – 01 Apr 2022*	*07 Mar – 29 Mar 2022*	*8000*
4	*Moderna*	*05 Mar – 01 Apr 2022*	*05 Mar – 31 Mar 2022*	*9249*

After crawled and retrieved the Twitter data for each COVID-19 vaccine type topic, the dataset filtering process is implemented to make sure the dataset only in March 2022 period. The Table 4.2 shows the number of tweets from the four dataset topics is decreasing after I filtered the period. This imply that there are some tweets that are not in March 2022 period. The Table 4.2 also describes after they are filtered, the Sinovac dataset has 9270 raw tweets, the AstaZeneca dataset has 8443 raw tweets, the Pfizer dataset has 8000 raw tweets, and the Moderna dataset has 9249 raw tweets.

The filtered datasets will be applied with tweet pre-processing that are tweet cleaning and text normalization; the tweet cleaning includes removing non-ASCII characters, removing HTML tags, removing retweets, removing links, removing mentions, removing hashtags, changing word to number, removing numbers, and removing punctuations; the text normalization includes case folding, expanding contractions, replacing negations, removing stopwords, and words lemmatizing. The last processes of the dataset pre-processing are implemented with removing duplicate tweets and removing empty values.

Table 4.3 The Sinovac Dataset Pre-Processing Result Samples

No.	***Raw Tweets***	***Pre-Processed Tweets***
1	*@LycosAustralis @ekecheirion In America Sputnik will never get approved but Sinovac probably will eventually*	*america sputnik never get approve sinovac probably eventually*
2	*@JKJAVMY hi, may i know this station got sinovac for 9 years old kid ?? thanks*	*hi may know station get sinovac year old kid thanks*
3	*RT @gsgcd13: 3/6/2022 Miles Guo: The CCP has just distributed 250 million doses of the so-called high-end Sinovac COVID vaccine to Hong Kon…*	*mile guo ccp distribute dos socalled highend sinovac covid vaccine hong kon*
4	*@hkstream @webbhk Might be kids - only Sinovac available for them I think.*	*might kid sinovac available think*
5	*@webbhk Any theory on why a large majority is now choosing Sinovac? Makes no sense…*	*theory large majority choose sinovac make sense*

The Table 4.3 shows the Sinovac dataset raw tweets and result samples after pre-processing process through the tweet cleaning, text normalization, duplicate tweets removal, and empty values removal.

Table 4.4 The AstraZeneca Dataset Pre-Processing Result Samples

No.	***Raw Tweets***	***Pre-Processed Tweets***
1	*RT @Jenna_bee__: @GullyJudith @truth8775 When I was injured is when I learned doctors here aren't ALLOWED to write exemptions. Also, advers…*	*injured learn doctor prevent write exemption also advers*
2	*RT @DrJohnB2: Longitudinally extensive transverse myelitis (inflammation of the spinal cord) after #CovidVaccination (AstraZeneca): https:/…*	*longitudinally extensive transverse myelitis inflammation spinal cord covidvaccination astrazeneca*
3	*Did Shane have the AstraZeneca jab? How many boxers have heart trouble cutting weight? Shane was a*	*shane astrazeneca jab many boxer heart trouble cut weight*

	professional athlete with nutritional knowledge and experience. https://t.co/xDosQsgbXe	*shane professional athlete nutritional knowledge experience*
4	*RT @1MikeFairclough: “That, sadly, is the situation of Lisa Shaw’s family. The 44-year-old BBC Radio Newcastle presenter died from a brain…*	*sadly situation lisa shaw family yearold bbc radio newcastle presenter die brain*
5	*@CovidBlindWife Far too many 1 year #covidVaccine damaged people - many of us nowhere near recovery #astrazeneca now rebranded Vaxzevria (previously COVID-19 Vaccine AstraZeneca) & opening a factory in #Cairo 🤢@Beck_Sall @HowardGriffiths @ake2306 @M7Rach @GardenerSpike @hibbsy1973*	*far many year covidvaccine damage people many u nowhere near recovery astrazeneca rebranded vaxzevria previously covid vaccine astrazeneca opening factory cairo*

The Table 4.4 shows the AstraZeneca dataset raw tweets and result samples after pre-processing process through the tweet cleaning, text normalization, duplicate tweets removal, and empty values removal.

Table 4.5 The Pfizer Dataset Pre-Processing Result Samples

No.	***Raw Tweets***	***Pre-Processed Tweets***
1	*RT @theysayitsrare: Nothing to see here ... just a $2.8M wire transfer from Pfizer to the FDA 😉 https://t.co/ucxdFXbWBk*	*nothing see wire transfer pfizer fda*
2	*RT @LfcJmr: @DrLoupis Pfizer Could Lose Liability Protection and Be Sued if the Deaths Are Proven as Willful\n\nThomasRenz: "I don't know how…*	*pfizer could lose liability protection sue death proven willful thomasrenz ignore*
3	*RT @ABert1968: Pfizer Admits its Vaccine Ineffective in Children 5 to 11*	*pfizer admits vaccine ineffective child double governor stop inject child*

	but Not One Single Governor Stops Injecting Children	
4	*RT @TheRaiderRandy: Dr. Robert Malone says the initial batch of Pfizer documents show the "government has been well aware that they [the va…*	*dr robert malone say initial batch pfizer document show government well aware va*
5	*@JaydeLovell The effects can go up to 2-3 years and Pfizer's released documentation shows page after page of damaging side effects most of which are fatal and already 25,000+ recorded deaths in America alone.*	*effect go year pfizer release documentation show page page damage side effect fatal already record death america alone*

The Table 4.5 shows the Pfizer dataset raw tweets and result samples after pre-processing process through the tweet cleaning, text normalization, duplicate tweets removal, and empty values removal.

Table 4.6 The Moderna Dataset Pre-Processing Result Samples

No.	***Raw Tweets***	***Pre-Processed Tweets***
1	*RT @SAHealth: A half dose of the Moderna COVID-19 vaccine is available for children aged 6 to 11 at selected pharmacies and GPs.\n\nCheck wit…*	*half dose moderna covid vaccine available child age select pharmacy gps check wit*
2	*RT @JeanRees10: DEATH - BOOSTER\n\n27 yo female - Foreign\n\n20 days post 3rd dose Moderna\n\n"This regulatory authority case concerns a 27-year-…*	*death booster yo female foreign day post rd dose moderna regulatory authority case concern year*
3	*RT @INArteCarloDoss: I assume everyone read the paper published in Frontiers in Virology on Feb 21 that discovered that a sequence of the C…*	*assume everyone read paper publish frontier virology feb discover sequence c*
4	*RT @JeanRees10: DEATH\n\n29 yo female - Foreign\n\n6 days post 2nd dose Moderna\n\nCardiac Arrest\nPulmonary*	*death yo female foreign day post nd dose moderna cardiac arrest pulmonary thromboembolism cardiac arrest ast*

	Thromboembolism\n\n"Cardiac arrest; AST…	
5	*RT @NoniTwit1: @Wiep13396680 @rinsjan @LidwienNews @J41963735 @breaknieuws @TwietterNatie @nieuws_in @ine66H @BilderbergsS @Observeerder SM…*	*sm*

The Table 4.6 shows the Moderna dataset raw tweets and result samples after pre-processing process through the tweet cleaning, text normalization, duplicate tweets removal, and empty values removal.

Table 4.7 The Number of Tweets Before and After Dataset Pre-Processing

No.	***Dataset***	***Number of Tweets Before Pre-Processing***	***Number of Tweets After Pre-Processing***
1	*Sinovac*	*9270*	*3366*
2	*AstraZeneca*	*8443*	*2909*
3	*Pfizer*	*8000*	*2622*
4	*Moderna*	*9249*	*3754*

The Table 4.7 shows the number of Tweets before and after pre-processing on Sinovac dataset, AstraZeneca dataset, Pfizer dataset, and Moderna dataset. The Sinovac dataset has 9270 tweets before pre-processing and 3366 tweets after pre-processing. The AstraZeneca dataset has 8443 tweets before pre-processing and 2909 tweets after pre-processing. The Pfizer dataset has 8000 tweets before pre-processing and 2622 tweets after pre-processing. The Moderna dataset has 9249 tweets before pre-processing and 3754 tweets after pre-processing.

Table 4.8 The Number of Tweets of The Dataset Selection Process Result

No.	***Dataset***	***Number of Tweets for Experiment Dataset***	***Number of Tweets for Implementation Dataset***
1	*Sinovac*	*2271*	*1094*
2	*AstraZeneca*	*1487*	*1421*

3	*Pfizer*	*1351*	*1270*
4	*Moderna*	*1986*	*1767*

The Table 4.8 shows the result of the dataset selection process, as it described before, the datasets will be selected into the experiment dataset and implementation dataset. The experiment dataset used the pre-processed dataset for the sentiment analysis building such as training data and testing data. The implementation dataset used the raw dataset for the sentiment analysis application. Both of them is implemented with remove any empty values. The Sinovac dataset for the experiment has 2271 tweets and for the implementation has 1094 tweets. The AstraZeneca dataset for the experiment has 1487 tweets and for the implementation has 1421 tweets. The Pfizer dataset for the experiment has 1351 tweets and for the implementation has 1270 tweets. The Moderna dataset for the experiment has 1986 tweets and for the implementation has 1767 tweets.

Table 4.9 The Sinovac Dataset Labeling Process Result Samples

No.	***Tweets***	***Sentiment***
1	*america sputnik never get approve sinovac probably eventually*	*Negative*
2	*hi may know station get sinovac year old kid thanks*	*Positive*
3	*theory large majority choose sinovac make sense*	*Negative*
4	*misinform sinovac use omicron even little*	*Negative*
5	*sinovac coronavac weak protection boost mrna vaccine perhaps well*	*Positive*

The Table 4.9 shows the dataset labeling result samples for the pre-processed Sinovac experiment dataset using vaderSentiment library as sentiment analysis lexicon-based labeling. The result of the dataset labeling process resulted in negative sentiment and positive sentiment.

Table 4.10 The AstraZeneca Dataset Labeling Process Result Samples

No.	***Pre-Processed Tweets***	***Sentiment***
1	*injured learn doctor prevent write exemption also advers*	*Negative*
2	*shane astrazeneca jab many boxer heart trouble cut weight shane professional athlete nutritional knowledge experience*	*Positive*
3	*sadly situation lisa shaw family yearold bbc radio newcastle presenter die brain*	*Negative*
4	*far many year damage people many nowhere near recovery rebranded vaxzevria previously covid vaccine astrazeneca opening factory*	*Negative*
5	*air ronaldo score header ground astrazeneca thursday night slap tbh*	*Positive*

The Table 4.10 shows the dataset labeling result samples for the pre-processed AstraZeneca experiment dataset using vaderSentiment library as sentiment analysis lexicon-based labeling. The result of the dataset labeling process resulted in negative sentiment and positive sentiment.

Table 4.11 The Pfizer Dataset Labeling Process Result Samples

No.	***Pre-Processed Tweets***	***Sentiment***
1	*pfizer could lose liability protection sue death proven willful thomasrenz ignore*	*Negative*
2	*pfizer admits vaccine ineffective child married governor stop inject child*	*Negative*
3	*robert malone say initial batch pfizer document show government well aware*	*Positive*
4	*effect year pfizer release documentation show page page damage side effect fatal already record death america alone*	*Negative*
5	*soooo many must watch video follow release pfizer document jumped top list*	*Positive*

The Table 4.11 shows the dataset labeling result samples for the pre-processed Pfizer experiment dataset using vaderSentiment library as sentiment analysis lexicon-based labeling. The result of the dataset labeling process resulted in negative sentiment and positive sentiment.

Table 4.12 The Moderna Dataset Labeling Process Result Samples

No.	***Pre-Processed Tweets***	***Sentiment***
1	*death booster female foreign day post dose moderna regulatory authority case concern year*	*Negative*
2	*death female foreign day post dose moderna cardiac arrest pulmonary thromboembolism cardiac arrest ast*	*Negative*
3	*moderna stock crash loss top insider sell million dollar sharesowners sell share spend*	*Negative*
4	*death female foreign day post unk dose moderna case report physician describes occurrence*	*Negative*
5	*end heart enlargement palpitation tinnitus dizziness neurological issue paindifficulty walk*	*Positive*

The Table 4.12 shows the dataset labeling result samples for the pre-processed Moderna experiment dataset using vaderSentiment library as sentiment analysis lexicon-based labeling. The result of the dataset labeling process resulted in negative sentiment and positive sentiment.

Table 4.13 The Number of Tweets After Selected Binary Sentiment and Removed Short Text

No.	***Dataset***	***Negative and Positive Sentiment Only***	***Finalized Experiment Data***
1	*Sinovac*	*1716*	*1663*
2	*AstraZeneca*	*969*	*920*
3	*Pfizer*	*962*	*912*
4	*Moderna*	*1444*	*1325*

The Table 4.13 shows the dataset the selection labeled dataset where I only take the positive and negative sentiment in this work, then, I removed several short

texts that is below five words. The final Sinovac experiment dataset after selected binary sentiment and after removed the short text is 1663 tweets. The final AstraZeneca experiment dataset after selected binary sentiment and after removed the short text is 920 tweets. The final Pfizer experiment dataset after selected binary sentiment and after removed the short text is 912 tweets. The final Moderna experiment dataset after selected binary sentiment and after removed the short text is 1325 tweets.

Table 4.14 The Number of Tweets After The Dataset Splitting Process Result

No.	***Dataset***	***Number of Tweets in Training Data***	***Number of Tweets in Testing Data***
1	*Sinovac*	*1496*	*167*
2	*AstraZeneca*	*828*	*92*
3	*Pfizer*	*820*	*92*
4	*Moderna*	*1192*	*113*

The Table 4.14 shows the result of the dataset splitting process, as it described before, the dataset splitting process is resulted in training data, validation data, and testing data. The dataset splitting process used the pre-processed and labeled dataset. The Sinovac dataset has 1496 tweets for training data and 167 tweets for testing data. The AstraZeneca dataset has 828 tweets for training data and 92 tweets for testing data. The Pfizer dataset has 820 tweets for training data and 92 tweets for testing data. The Moderna dataset has 1192 tweets for training data and 113 tweets for testing data.

Table 4.15 The Number of Tweets Before and After The Dataset Augmentation Process Result

No.	***Dataset***	***Number of Tweets in Training Data***	
		Before Data Augmentation	***After Data Augmentation***
1	*Sinovac*	*1496*	*7480*
2	*AstraZeneca*	*828*	*4140*
3	*Pfizer*	*820*	*4100*
4	*Moderna*	*1192*	*5960*

The Table 4.15 shows the result of the dataset augmentation process, as it described before, the dataset augmentation process used the EDA approach such as synonym replacement, random swap, random insertion, and random deletion. The dataset augmentation process used only the training data and validation data. The Sinovac training data has 1496 tweets before data augmentation and has 7480 tweets after data augmentation. The AstraZeneca training data has 828 tweets before data augmentation and has 4140 tweets after data augmentation. The Pfizer training data has 820 tweets before data augmentation and has 4100 tweets after data augmentation. The Moderna training data has 1192 tweets before data augmentation and has 5960 tweets after data augmentation.

Table 4.16 The Training and Testing Size After The Sequence Tokenizing and Padding Process Result

No.	***Dataset***	***Padded Training Sequence Data Size***	***Padded Testing Sequence Data Size***
1	*Sinovac*	*(7480, 100)*	*(167, 100)*
2	*AstraZeneca*	*(4140, 100)*	*(92, 100)*
3	*Pfizer*	*(4100, 100)*	*(92, 100)*
4	*Moderna*	*(5960, 100)*	*(113, 100)*

Table 4.17 The Training and Testing After The Converting Label Data to Categorical Process Result

No.	***Dataset***	***Padded Training Label Data Size***	***Padded Testing Label Data Size***
1	*Sinovac*	*(7480, 2)*	*(167, 2)*
2	*AstraZeneca*	*(4140, 2)*	*(92, 2)*
3	*Pfizer*	*(4100, 2)*	*(92, 2)*
4	*Moderna*	*(5960, 2)*	*(113, 2)*

The Table 4.16 shows the result after the four dataset topics transformation, where each sequence is tokenized and padded. All the four padded datasets have same size of the column that is 100 maximum sequence length and already in a proper form to be trained in the sentiment analysis models. The Table 4.17 shows

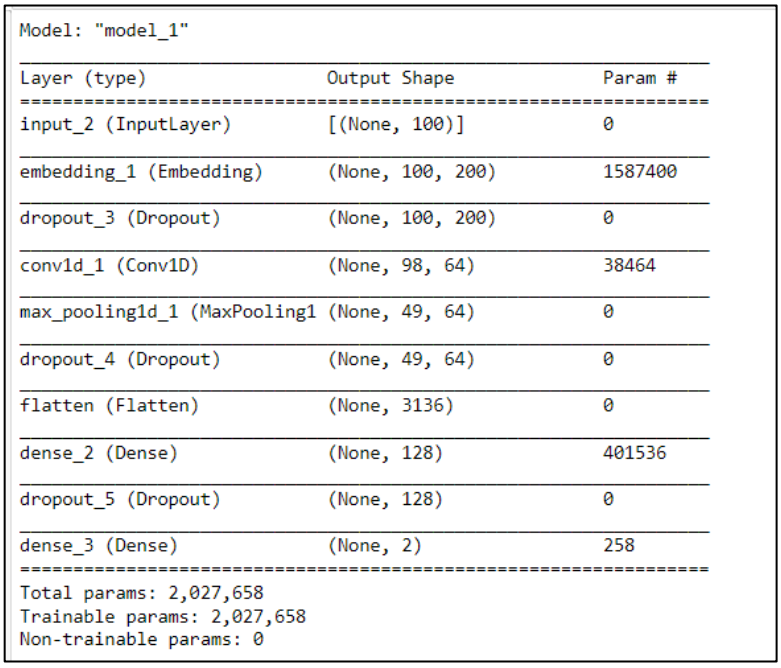
after the label data transformation for training and testing, where each class is transformed into categorical one-hot encoding. Therefore, all the four transformed label data have one-hot encoding size that is two columns for positive sentiment and negative sentiment.

4.1.2 Building Model

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding (Embedding)        (None, 100, 200)          1587400
_________________________________________________________________
dropout (Dropout)            (None, 100, 200)          0
_________________________________________________________________
conv1d (Conv1D)              (None, 98, 64)            38464
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 49, 64)            0
_________________________________________________________________
dropout_1 (Dropout)          (None, 49, 64)            0
_________________________________________________________________
lstm (LSTM)                  (None, 128)               98816
_________________________________________________________________
dense (Dense)                (None, 128)               16512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 1,741,450
Trainable params: 1,741,450
Non-trainable params: 0
```

Figure 4.6 The Sinovac Dataset Hybrid CNN-LSTM Model Training Parameters

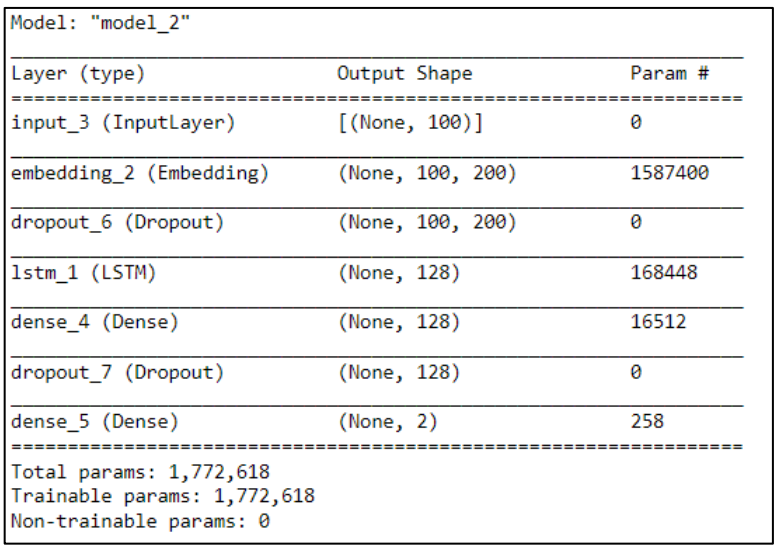
The Figure 4.6 shows the result of building the hybrid CNN-LSTM model for the Sinovac dataset, the result consisted of total parameters and trainable parameters. The total parameters have 1,741,450 parameters and the trainable parameters have 1,741,450 parameters.



```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding_1 (Embedding)      (None, 100, 200)          1587400
_________________________________________________________________
dropout_3 (Dropout)          (None, 100, 200)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 98, 64)            38464
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 49, 64)            0
_________________________________________________________________
dropout_4 (Dropout)          (None, 49, 64)            0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               401536
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 2,027,658
Trainable params: 2,027,658
Non-trainable params: 0
```

Figure 4.7 The Sinovac Dataset Single CNN Model Training Parameters

The Figure 4.7 shows the result of building the single CNN model for the Sinovac dataset, the result consisted of total parameters and trainable parameters. The total parameters have 2,027,658 parameters and the trainable parameters have 2,027,658 parameters.

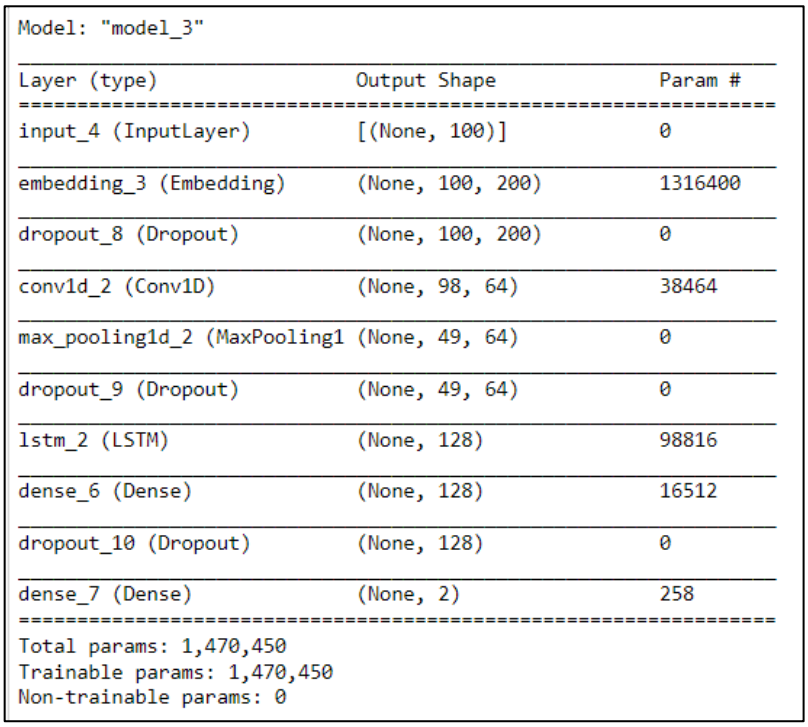


```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding_2 (Embedding)      (None, 100, 200)          1587400
_________________________________________________________________
dropout_6 (Dropout)          (None, 100, 200)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               168448
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 258
=================================================================
Total params: 1,772,618
Trainable params: 1,772,618
Non-trainable params: 0
```

Figure 4.8 The Sinovac Dataset Single LSTM Model Training Parameters

The Figure 4.8 shows the result of building the single LSTM model for the Sinovac dataset, the result consisted of total parameters and trainable parameters.

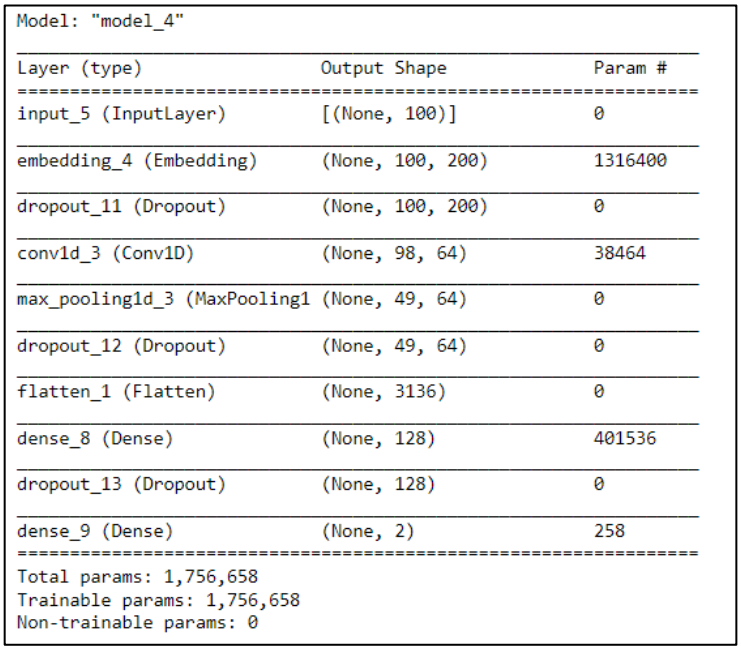
The total parameters have 1,772,618 parameters and the trainable parameters have 1,772,618 parameters.



```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding_3 (Embedding)      (None, 100, 200)          1316400
_________________________________________________________________
dropout_8 (Dropout)          (None, 100, 200)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 98, 64)            38464
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 49, 64)            0
_________________________________________________________________
dropout_9 (Dropout)          (None, 49, 64)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               98816
_________________________________________________________________
dense_6 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 258
=================================================================
Total params: 1,470,450
Trainable params: 1,470,450
Non-trainable params: 0
```

Figure 4.9 The AstraZeneca Dataset Hybrid CNN-LSTM Model Training Parameters

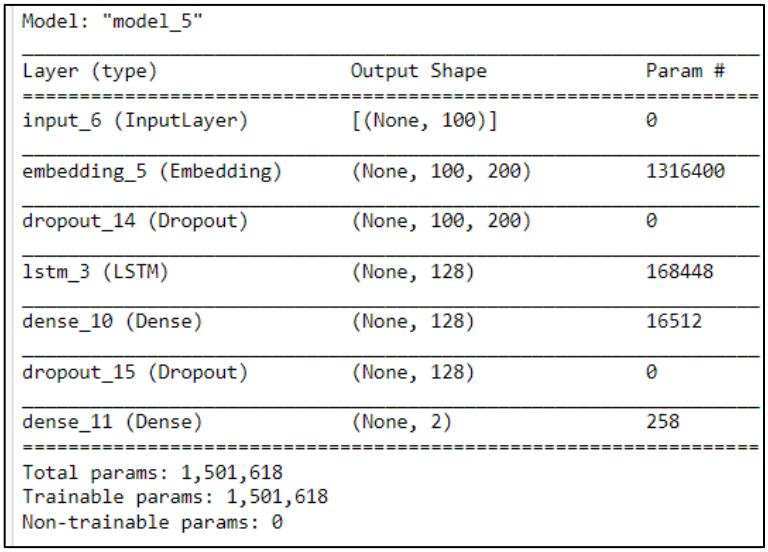
The Figure 4.9 shows the result of building the hybrid CNN-LSTM model for the AstraZeneca dataset, the result consisted of total parameters and trainable parameters. The total parameters have 1,470,450 parameters and the trainable parameters have 1,470,450 parameters.



```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding_4 (Embedding)      (None, 100, 200)          1316400
_________________________________________________________________
dropout_11 (Dropout)         (None, 100, 200)          0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 98, 64)            38464
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 49, 64)            0
_________________________________________________________________
dropout_12 (Dropout)         (None, 49, 64)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               401536
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 258
=================================================================
Total params: 1,756,658
Trainable params: 1,756,658
Non-trainable params: 0
```

Figure 4.10 The AstraZeneca Dataset Single CNN Model Training Parameters

The Figure 4.10 shows the result of building the single CNN model for the Sinovac dataset, the result consisted of total parameters and trainable parameters. The total parameters have 1,756,658 parameters and the trainable parameters have 1,756,658 parameters.

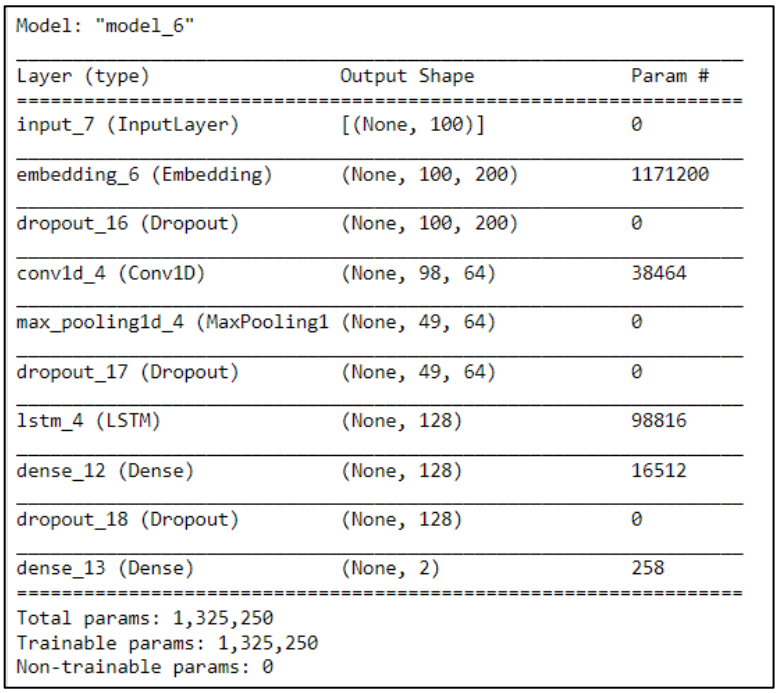


```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding_5 (Embedding)      (None, 100, 200)          1316400
_________________________________________________________________
dropout_14 (Dropout)         (None, 100, 200)          0
_________________________________________________________________
lstm_3 (LSTM)                (None, 128)               168448
_________________________________________________________________
dense_10 (Dense)             (None, 128)               16512
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 258
=================================================================
Total params: 1,501,618
Trainable params: 1,501,618
Non-trainable params: 0
```

Figure 4.11 The AstraZeneca Dataset Single LSTM Model Training Parameters

The Figure 4.11 shows the result of building the single LSTM model for the AstraZeneca dataset, the result consisted of total parameters and trainable

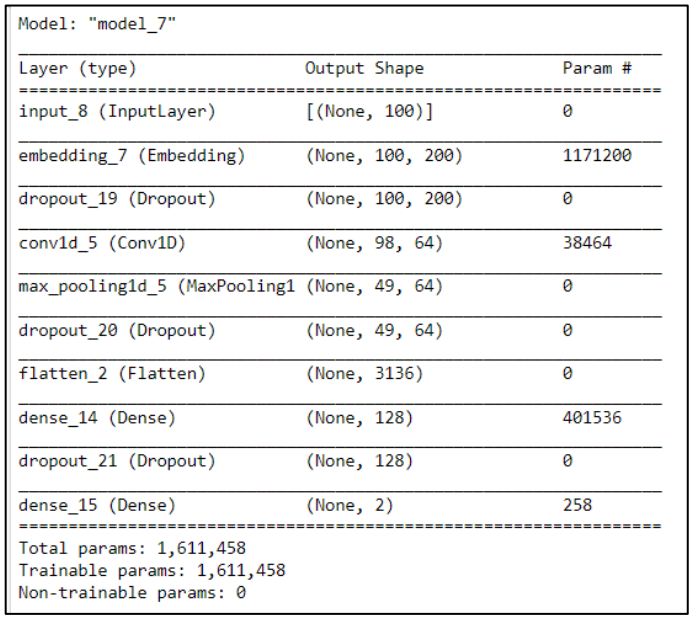
parameters. The total parameters have 1,501,618 parameters and the trainable parameters have 1,501,618 parameters.



```
Model: "model_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding_6 (Embedding)      (None, 100, 200)          1171200
_________________________________________________________________
dropout_16 (Dropout)         (None, 100, 200)          0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 98, 64)            38464
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 49, 64)            0
_________________________________________________________________
dropout_17 (Dropout)         (None, 49, 64)            0
_________________________________________________________________
lstm_4 (LSTM)                (None, 128)               98816
_________________________________________________________________
dense_12 (Dense)             (None, 128)               16512
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_13 (Dense)             (None, 2)                 258
=================================================================
Total params: 1,325,250
Trainable params: 1,325,250
Non-trainable params: 0
```

Figure 4.12 The Pfizer Dataset Hybrid CNN-LSTM Model Training Parameters

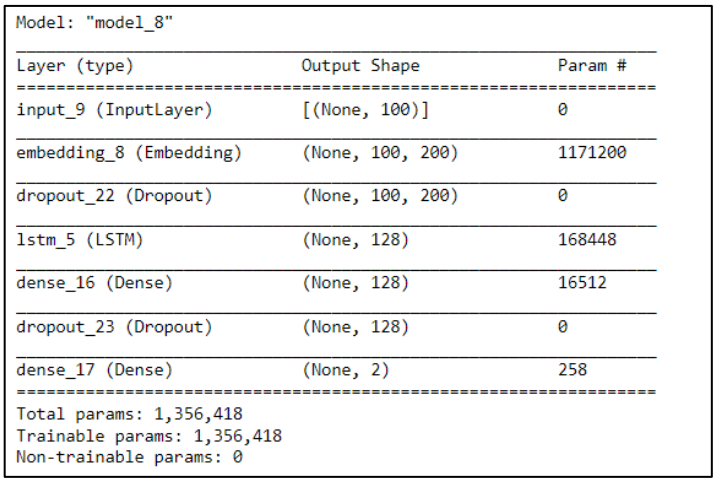
The Figure 4.12 shows the result of building the hybrid CNN-LSTM model for the Pfizer dataset, the result consisted of total parameters and trainable parameters. The total parameters have 1,325,250 parameters and the trainable parameters have 1,325,250 parameters.



```
Model: "model_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding_7 (Embedding)      (None, 100, 200)          1171200
_________________________________________________________________
dropout_19 (Dropout)         (None, 100, 200)          0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 98, 64)            38464
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 49, 64)            0
_________________________________________________________________
dropout_20 (Dropout)         (None, 49, 64)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               401536
_________________________________________________________________
dropout_21 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_15 (Dense)             (None, 2)                 258
=================================================================
Total params: 1,611,458
Trainable params: 1,611,458
Non-trainable params: 0
```

Figure 4.13 The Pfizer Dataset Single CNN Model Training Parameters

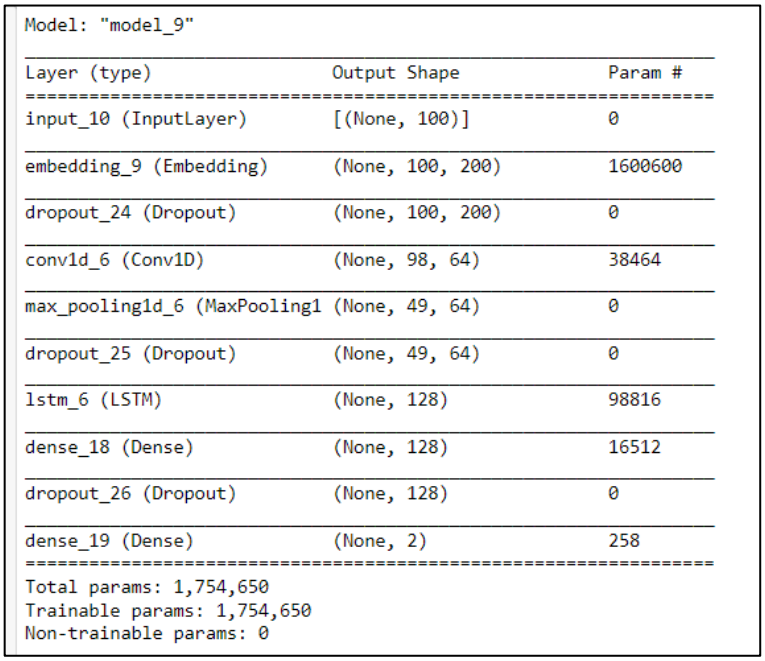
The Figure 4.13 shows the result of building the single CNN model for the Pfizer dataset, the result consisted of total parameters and trainable parameters. The total parameters have 1,611,458 parameters and the trainable parameters have 1,611,458 parameters.



```
Model: "model_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
embedding_8 (Embedding)      (None, 100, 200)          1171200
_________________________________________________________________
dropout_22 (Dropout)         (None, 100, 200)          0
_________________________________________________________________
lstm_5 (LSTM)                (None, 128)               168448
_________________________________________________________________
dense_16 (Dense)             (None, 128)               16512
_________________________________________________________________
dropout_23 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_17 (Dense)             (None, 2)                 258
=================================================================
Total params: 1,356,418
Trainable params: 1,356,418
Non-trainable params: 0
```

Figure 4.14 The Pfizer Dataset Single LSTM Model Training Parameters

The Figure 4.14 shows the result of building the single LSTM model for the Pfizer dataset, the result consisted of total parameters and trainable parameters. The total parameters have 1,501,618 parameters and the trainable parameters have 1,501,618 parameters.



```
Model: "model_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        [(None, 100)]             0
_________________________________________________________________
embedding_9 (Embedding)      (None, 100, 200)          1600600
_________________________________________________________________
dropout_24 (Dropout)         (None, 100, 200)          0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 98, 64)            38464
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 49, 64)            0
_________________________________________________________________
dropout_25 (Dropout)         (None, 49, 64)            0
_________________________________________________________________
lstm_6 (LSTM)                (None, 128)               98816
_________________________________________________________________
dense_18 (Dense)             (None, 128)               16512
_________________________________________________________________
dropout_26 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 2)                 258
=================================================================
Total params: 1,754,650
Trainable params: 1,754,650
Non-trainable params: 0
```

Figure 4.15 The Moderna Dataset Hybrid CNN-LSTM Model Training Parameters

The Figure 4.15 shows the result of building the hybrid CNN-LSTM model for the Moderna dataset, the result consisted of total parameters and trainable parameters. The total parameters have 1,754,650 parameters and the trainable parameters have 1,754,650 parameters.

```
Model: "model_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 100)]             0
_________________________________________________________________
embedding_10 (Embedding)     (None, 100, 200)          1600600
_________________________________________________________________
dropout_27 (Dropout)         (None, 100, 200)          0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 98, 64)            38464
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 49, 64)            0
_________________________________________________________________
dropout_28 (Dropout)         (None, 49, 64)            0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_20 (Dense)             (None, 128)               401536
_________________________________________________________________
dropout_29 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_21 (Dense)             (None, 2)                 258
=================================================================
Total params: 2,040,858
Trainable params: 2,040,858
Non-trainable params: 0
```

Figure 4.16 The Moderna Dataset Single CNN Model Training Parameters

The Figure 4.16 shows the result of building the single CNN model for the Moderna dataset, the result consisted of total parameters and trainable parameters. The total parameters have 2,040,858 parameters and the trainable parameters have 2,040,858 parameters.

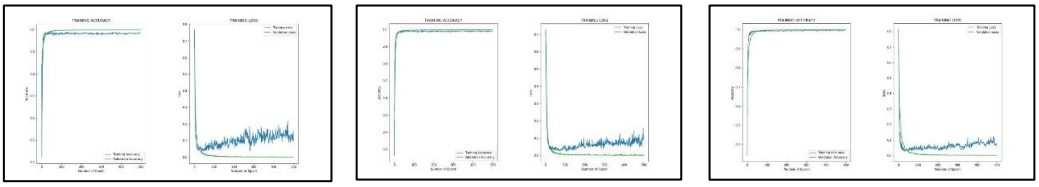
```
Model: "model_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_12 (InputLayer)        [(None, 100)]             0
_________________________________________________________________
embedding_11 (Embedding)     (None, 100, 200)          1600600
_________________________________________________________________
dropout_30 (Dropout)         (None, 100, 200)          0
_________________________________________________________________
lstm_7 (LSTM)                (None, 128)               168448
_________________________________________________________________
dense_22 (Dense)             (None, 128)               16512
_________________________________________________________________
dropout_31 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_23 (Dense)             (None, 2)                 258
=================================================================
Total params: 1,785,818
Trainable params: 1,785,818
Non-trainable params: 0
```

Figure 4.17 The Moderna Dataset Single LSTM Model Training Parameters

The Figure 4.17 shows the result of building the single LSTM model for the Moderna dataset, the result consisted of total parameters and trainable parameters.

The total parameters have 1,785,818 parameters and the trainable parameters have 1,785,818 parameters.

4.1.3 Training Result: Sinovac Dataset Model

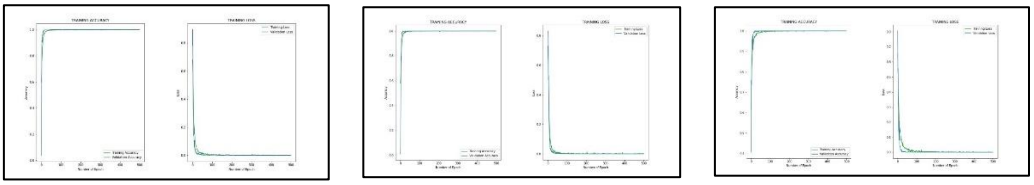


Hybrid CNN-LSTM Model *Single CNN Model* *Single LSTM Model*

Figure 4.18 The Sinovac Dataset Training Result

The Figure 4.18 shows the training results between the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model for Sinovac dataset. Overall, the three model training accuracies and validation accuracies converge has converge well with 500 epochs. However, the hybrid CNN-LSTM model, the single CNN model and the single LSTM model validation loss is beginning to raise after around 30 – 50 epochs. In contrast, the hybrid CNN-LSTM model training is noisier than the single CNN model and the single LSTM model, where the validation loss spike is unstable.

4.1.4 Training Result: AstraZeneca Dataset Model



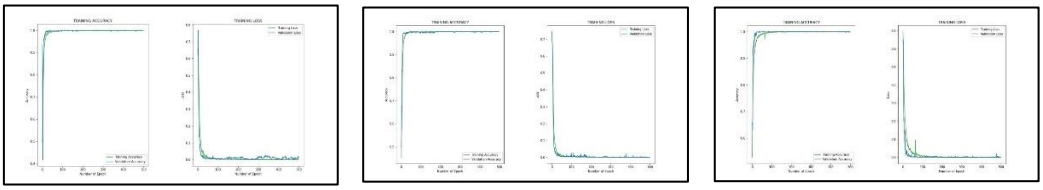
Hybrid CNN-LSTM Model *Single CNN Model* *Single LSTM Model*

Figure 4.19 The AstraZeneca Dataset Training Result

The Figure 4.19 shows the training results between the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model for AstraZeneca dataset. Overall, the three model training accuracies and validation accuracies has converged very well with 500 epochs. In terms of validation loss and training loss, the single CNN model is converged earlier around 60 – 70 epochs, followed by the hybrid CNN-LSTM model that converged around 80 – 100 epochs, and followed by the single LSTM model that converged lately around 120 – 150 epochs. The

hybrid CNN-LSTM model training has the most stable training process other than the single CNN model and the single LSTM model.

4.1.5 Training Result: Pfizer Dataset Model

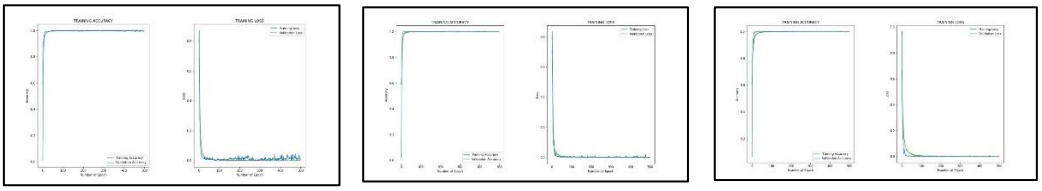


Hybrid CNN-LSTM Model *Single CNN Model* *Single LSTM Model*

Figure 4.20 The Pfizer Dataset Training Result

The Figure 4.20 shows the training results between the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model for Pfizer dataset. Overall, the three model training accuracies and validation accuracies has converged very well with long epochs. In terms of validation loss and training loss, the hybrid CNN-LSTM model is converged earlier around 60 – 100 epochs, however the validation loss is unstable after around 110 – 120 epochs. The single CNN model and the single LSTM model has more stable validation loss, the single LSTM model is converged earlier around 180 – 190 epochs and the single CNN is converged after 200 epochs.

4.1.6 Training Result: Moderna Dataset Model



Hybrid CNN-LSTM Model *Single CNN Model* *Single LSTM Model*

Figure 4.21 The Moderna Dataset Training Result

The Figure 4.21 shows the training results between the hybrid CNN-LSTM model, the single CNN model, and the single LSTM model for Pfizer dataset. Again, the three model training accuracies and validation accuracies has converged very well with long epochs. In terms of validation loss and training loss, the hybrid CNN-LSTM model is converged earlier around 60 – 100 epochs, however the validation loss is unstable after around 120 – 150 epochs. The single CNN model and the single LSTM model has more stable validation loss, the single CNN model is

converged earlier around 80 – 100 epochs and the single LSTM is converged after 120 – 130 epochs.

4.1.7 Testing and Evaluation Result: Sinovac Dataset Model

Table 4.18 The Sinovac Dataset Hybrid CNN-LSTM Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*69*	*20*
Positive	*26*	*52*

The Figure 4.18 shows the confusion matrix of the hybrid CNN-LSTM model for the Sinovac dataset. The hybrid CNN-LSTM model predicted 52 tweets as positive sentiment correctly (TP), predicted 69 tweets as negative sentiment correctly (TN), predicted 26 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 20 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

Table 4.19 The Sinovac Dataset Single CNN Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*72*	*17*
Positive	*23*	*55*

The Figure 4.19 shows the confusion matrix of the single CNN model for the Sinovac dataset. The single CNN model predicted 55 tweets as positive sentiment correctly (TP), predicted 72 tweets as positive sentiment correctly (TN), predicted 23 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 17 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

Table 4 20 The Sinovac Dataset Single LSTM Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*65*	*24*
Positive	*25*	*53*

The Figure 4.20 shows the confusion matrix of the single LSTM model for the Sinovac dataset. The single LSTM model predicted 53 tweets as positive sentiment correctly (TP), predicted 65 tweets as negative sentiment correctly (TN), predicted 25 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 24 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

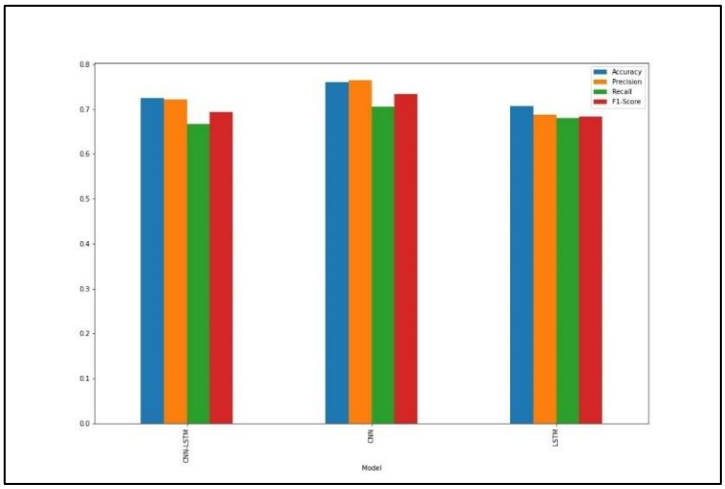


Figure 4.22 The Sinovac Dataset The Hybrid CNN-LSTM Model, The Single CNN, and The Single LSTM Model Evaluation Metrics Comparison

Table 4.21 The Sinovac Dataset The Hybrid CNN-LSTM Model, The Single CNN, and The Single LSTM Model Evaluation Metrics Result

Model	***Label***	***Metrics***		
		Recall	***Precision***	***F1-Score***
Hybrid CNN-LSTM	*Negative*	***78%***	*73%*	*75%*
	Positive	*67%*	***72%***	*69%*
Average		*72%*	*72%*	*72%*
Single CNN	*Negative*	***81%***	*76%*	*78%*
	Positive	*71%*	***76%***	*73%*

Average		***76%***	***76%***	***76%***
Single LSTM	*Negative*	***73%***	*72%*	*73%*
	Positive	*68%*	***69%***	*68%*
Average		*70%*	*71%*	*71%*

The Table 4.21 shows the evaluation metrics recall, precision, and f1-score of the hybrid CNN-LSTM mode, the single CNN model, and the single LSTM model for the Sinovac dataset. The single CNN model has the best average of recall, precision, and f1-score for the Sinovac dataset with value 76%, 76%, and 76% respectively. All the three models have the highest recall value on negative sentiment and the highest precision value on positive sentiment: the hybrid CNN-LSTM model has 78% recall on the negative sentiment and 72% precision on the positive sentiment; the single CNN model has 81% recall on the negative sentiment and 76% precision on the positive sentiment; and the single LSTM model has 73% recall on the negative sentiment and 69% precision on the positive sentiment.

Table 4.22 The Sinovac Dataset The Hybrid CNN-LSTM Model, The Single CNN, and The Single LSTM Model Testing Result

Model	***Model Testing***	
	Accuracy	***Loss***
Hybrid CNN-LSTM	*72,46%*	***3,4***
Single CNN	***76,05%***	*10,3*
Single LSTM	*70,66%*	*3,86*

The Table 4.22 shows the test results of the hybrid CNN-LSTM mode, the single CNN model, and the single LSTM model for the Sinovac dataset. Hence, the single CNN model has the highest accuracy on the test with 76,05% accuracy, in contrast, the hybrid CNN-LSTM has the lowest loss on the test with 3,4 loss.

4.1.8 Testing and Evaluation Result: AstraZeneca Dataset Model

Table 4.23 The AstraZeneca Dataset Hybrid CNN-LSTM Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*32*	*10*
Positive	*8*	*42*

The Figure 4.23 shows the confusion matrix of the hybrid CNN-LSTM model for the AstraZeneca dataset. The hybrid CNN-LSTM model predicted 42 tweets as positive sentiment correctly (TP), predicted 32 tweets as negative sentiment correctly (TN), predicted 8 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 10 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

Table 4.24 The AstraZeneca Dataset Single CNN Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*31*	*11*
Positive	*7*	*43*

The Figure 4.24 shows the confusion matrix of the single CNN model for the AstraZeneca dataset. The single CNN model predicted 43 tweets as positive sentiment correctly (TP), predicted 31 tweets as negative sentiment correctly (TN), predicted 7 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 11 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

Table 4 25 The AstraZeneca Dataset Single LSTM Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*30*	*12*
Positive	*12*	*38*

The Figure 4.25 shows the confusion matrix of the single LSTM model for the AstraZeneca dataset. The single LSTM model predicted 38 tweets as positive sentiment correctly (TP), predicted 30 tweets as negative sentiment correctly (TN), predicted 12 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 12 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

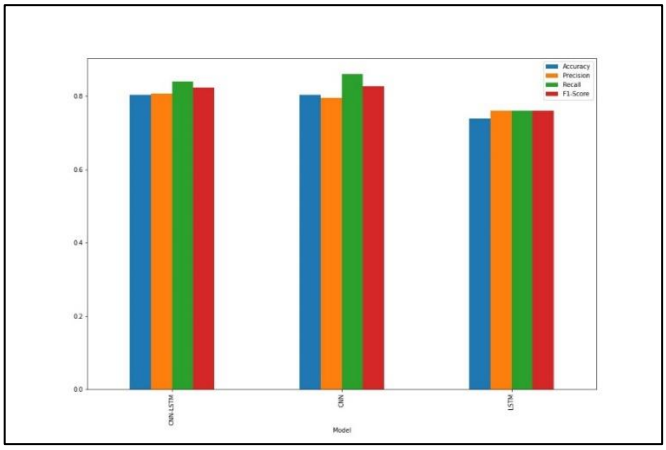


Figure 4.23 The AstraZeneca Dataset The Hybrid CNN-LSTM Model, The Single CNN, and The Single LSTM Model Evaluation Metrics Comparison

Table 4.26 The AstraZeneca Dataset The Hybrid CNN-LSTM Model, The Single CNN, and The Single LSTM Model Evaluation Metrics Result

Model	***Label***	***Metrics***		
		Recall	***Precision***	***F1-Score***
Hybrid CNN-LSTM	*Negative*	*76%*	***80%***	*78%*
	Positive	***84%***	*81%*	*82%*
Average		*80%*	*80%*	*80%*
Single CNN	*Negative*	*74%*	***82%***	*78%*
	Positive	***86%***	*80%*	*80%*
Average		***80%***	***81%***	***81%***
Single LSTM	*Negative*	*71%*	*71%*	*71%*
	Positive	*76%*	*76%*	*76%*
Average		*74%*	*74%*	*74%*

The Table 4.26 shows the evaluation metrics recall, precision, and f1-score of the hybrid CNN-LSTM mode, the single CNN model, and the single LSTM

model for the AstraZeneca dataset. The single CNN model has the best average of recall, precision, and f1-score for the AstraZeneca dataset with value 80%, 81%, and 81% respectively. The hybrid CNN-LSTM model and the single CNN model has the highest precision value on the negative sentiment and the highest recall on the positive sentiment: the hybrid CNN-LSTM model has 80% precision on the negative sentiment and 84% recall on the positive sentiment and the single CNN model has 82% precision on the negative sentiment and 86% recall on the positive sentiment; and the single LSTM model has 73% on the negative sentiment and 69% on the positive sentiment.

Table 4.27 The AstraZeneca Dataset Three Models Testing Result

Model	***Model Testing***	
	Accuracy	***Loss***
Hybrid CNN-LSTM	***80,43%***	***2,17***
Single CNN	*80,43%*	*3,08*
Single LSTM	*73,91%*	*2,9*

The Table 4.27 shows the test results of the hybrid CNN-LSTM mode, the single CNN model, and the single LSTM model for the AstraZeneca dataset. Hence, the hybrid CNN-LSTM model has the highest accuracy on the test with 80,43% accuracy and the lowest loss on the test with 2,17 loss among the two single models, although the precision, recall, and f1-score lower than the single CNN model.

4.1.8 Testing and Evaluation Result: Pfizer Dataset Model

Table 4.28 The Pfizer Dataset Hybrid CNN-LSTM Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*43*	*8*
Positive	*15*	*26*

The Figure 4.28 shows the confusion matrix of the hybrid CNN-LSTM model for the Pfizer dataset. The hybrid CNN-LSTM model predicted 26 tweets as positive sentiment correctly (TP), predicted 43 tweets as negative sentiment

correctly (TN), predicted 15 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 8 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

Table 4.29 The Pfizer Dataset Single CNN Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*40*	*5*
Positive	*11*	*26*

The Figure 4.29 shows the confusion matrix of the single CNN model for the Pfizer dataset. The single CNN model predicted 26 tweets as positive sentiment correctly (TP), predicted 40 tweets as negative sentiment correctly (TN), predicted 11 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 5 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

Table 4 30 The Pfizer Dataset Single LSTM Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*42*	*9*
Positive	*14*	*27*

The Figure 4.30 shows the confusion matrix of the single LSTM model for the Pfizer dataset. The single LSTM model predicted 27 tweets as positive sentiment correctly (TP), predicted 42 tweets as negative sentiment correctly (TN), predicted 14 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 9 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

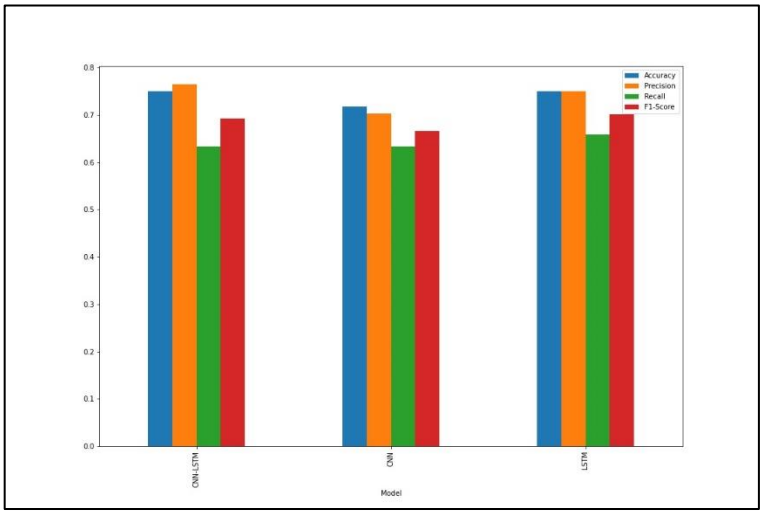


Figure 4.24 The Pfizer Dataset The Hybrid CNN-LSTM Model, The Single CNN, and The Single LSTM Model Evaluation Metrics Comparison

Table 4.31 The Pfizer Dataset The Hybrid CNN-LSTM Model, The Single CNN, and The Single LSTM Model Evaluation Metrics Result

Model	***Label***	***Metrics***		
		Recall	***Precision***	***F1-Score***
Hybrid CNN-LSTM	*Negative*	***84%***	*74%*	*79%*
	Positive	*63%*	***76%***	*69%*
Average		***74%***	***75%***	***74%***
Single CNN	*Negative*	***78%***	*73%*	*75%*
	Positive	*63%*	***70%***	*67%*
Average		*71%*	*71%*	*71%*
Single LSTM	*Negative*	***82%***	*75%*	*79%*
	Positive	*66%*	***75%***	*70%*
Average		***74%***	***75%***	***74%***

The Table 4.31 shows the evaluation metrics recall, precision, and f1-score of the hybrid CNN-LSTM mode, the single CNN model, and the single LSTM model for the Pfizer dataset. The hybrid CNN-LSTM and the single LSTM model has the equal average of recall, precision, and f1-score for the AstraZeneca dataset with value 74%, 75%, and 74% respectively. All the three models have the highest recall value on negative sentiment and the highest precision value on positive sentiment: the hybrid CNN-LSTM model has 84% recall on the negative sentiment

and 76% precision on the positive sentiment; the single CNN model has 78% recall on the negative sentiment and 70% precision on the positive sentiment; and the single LSTM model has 82% recall on the negative sentiment and 75% precision on the positive sentiment.

Table 4.32 The Pfizer Dataset Three Models Testing Result

Model	***Model Testing***	
	Accuracy	***Loss***
Hybrid CNN-LSTM	***75,00%***	***3,71***
Single CNN	*71,74%*	*6,65*
Single LSTM	*75,00%*	*4,41*

The Table 4.32 shows the test results of the hybrid CNN-LSTM mode, the single CNN model, and the single LSTM model for the Pfizer dataset. Hence, the hybrid CNN-LSTM model has the highest accuracy on the test with 75,00% accuracy and the lowest loss on the test with 3,71 loss among the two single models.

4.1.9 Testing and Evaluation Result: Moderna Dataset Model

Table 4.33 The Moderna Dataset Hybrid CNN-LSTM Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*34*	*25*
Positive	*12*	*62*

The Figure 4.33 shows the confusion matrix of the hybrid CNN-LSTM model for the Moderna dataset. The hybrid CNN-LSTM model predicted 62 tweets as positive sentiment correctly (TP), predicted 34 tweets as negative sentiment correctly (TN), predicted 12 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 25 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

Table 4.34 The Moderna Dataset Single CNN Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*33*	*26*
Positive	*15*	*59*

The Figure 4.34 shows the confusion matrix of the single CNN model for the Pfizer dataset. The single CNN model predicted 59 tweets as positive sentiment correctly (TP), predicted 33 tweets as negative sentiment correctly (TN), predicted 15 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 26 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

Table 4.35 The Moderna Dataset Single LSTM Model Confusion Matrix Result

Predicted	***Actual***	
	Negative	***Positive***
Negative	*35*	*24*
Positive	*14*	*60*

The Figure 4.35 shows the confusion matrix of the single LSTM model for the Moderna dataset. The single LSTM model predicted 60 tweets as positive sentiment correctly (TP), predicted 35 tweets as negative sentiment correctly (TN), predicted 14 tweets as positive sentiment incorrectly that the actual label is negative sentiment (FN), and predicted 24 tweets as negative sentiment incorrectly that the actual label is positive sentiment (FP).

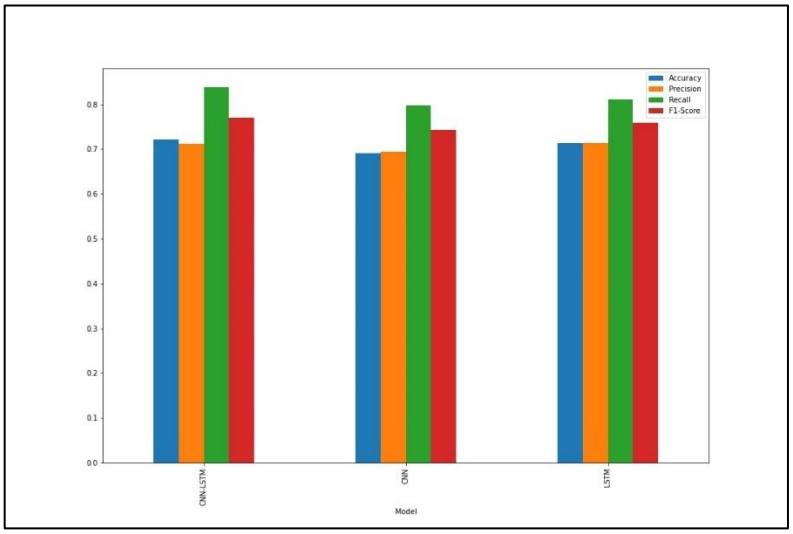


Figure 4.25 The Moderna Dataset The Hybrid CNN-LSTM Model, The Single CNN, and The Single LSTM Model Evaluation Metrics Comparison

Table 4.36 The Moderna Dataset The Hybrid CNN-LSTM Model, The Single CNN, and The Single LSTM Model Evaluation Metrics Result

Model	***Label***	***Metrics***		
		Recall	***Precision***	***F1-Score***
Hybrid CNN-LSTM	*Negative*	*58%*	***74%***	*65%*
	Positive	***84%***	*71%*	*77%*
Average		***71%***	***73%***	***71%***
Single CNN	*Negative*	*56%*	***69%***	*62%*
	Positive	***80%***	*69%*	*74%*
Average		*68%*	*69%*	*68%*
Single LSTM	*Negative*	*59%*	***71%***	*65%*
	Positive	***81%***	*71%*	*76%*
Average		*70%*	*71%*	*70%*

The Table 4.36 shows the evaluation metrics recall, precision, and f1-score of the hybrid CNN-LSTM mode, the single CNN model, and the single LSTM model for the Moderna dataset. The hybrid CNN-LSTM model has the best average of recall, precision, and f1-score for the Moderna dataset with value 71%, 73%, and 71% respectively. All the three models have the highest precision value on negative sentiment and the highest recall value on positive sentiment: the hybrid CNN-LSTM model has 74% precision on the negative sentiment and 84% recall on the

positive sentiment; the single CNN model has 69% precision on the negative sentiment and 80% recall on the positive sentiment; and the single LSTM model has 71% precision on the negative sentiment and 81% recall on the positive sentiment.

Table 4.37 The Moderna Dataset Three Models Testing Result

Model	***Model Testing***	
	Accuracy	***Loss***
Hybrid CNN-LSTM	***72,18%***	***3,88***
Single CNN	*69,17%*	*6,66*
Single LSTM	*71,43%*	*3,95*

The Table 4.37 shows the test results of the hybrid CNN-LSTM mode, the single CNN model, and the single LSTM model for the Pfizer dataset. Hence, the hybrid CNN-LSTM model has the highest accuracy on the test with 72,18% accuracy and the lowest loss on the test with 3,88 loss among the two single models.

4.1.10 Discussion

Based on the results, the sentiment analysis model performances are varied on the different dataset topics. On the Sinovac dataset topic, the single CNN model has the highest average of the recall, precision, and f1-score with 76%, 76%, and 76% respectively. In contrast, the single CNN model has the highest accuracy on the test data with 76,05% and the loss is quite high with 10,3 loss compared to the hybrid CNN-LSTM model with 3,4 loss. On the AstraZeneca dataset topic, the single CNN model has the highest average of the recall, precision, and f1-score with 80%, 81%, and 81% respectively. Interestingly, the hybrid CNN-LSTM model has better at the test data because of the lower loss with 80,43% accuracy and 2,17 loss compared the single CNN model with 80,43% accuracy and 3,08 loss. On the Pfizer dataset topic, the hybrid LSTM-CNN model has the highest average of the recall, precision, and f1-score with 74%, 75%, and 74% respectively. The hybrid CNN-LSTM model also has the highest performance on the test data with 75,00% accuracy and 3,71 loss. On the Moderna dataset topic, the hybrid LSTM-CNN

model has the highest average of the recall, precision, and f1-score with 74%, 75%, and 74% respectively. The hybrid CNN-LSTM model also has the highest performance on the test data with 72,18% accuracy and 3,88 loss.

Unfortunately, all the sentiment analysis models have high loss on the test data despite the accuracies are mostly reach 70%. There are several main causes that can be suspected in order to explain the high loss on the test data. First, some tweet texts are truncated, which, reduced the whole information; this may affect the lexicon-based labeling process that is possible to label the wrong sentiment because of the reduced information on the truncated tweet texts. Thus, there is some bias on the labeling process where several most words in positive sentiment also exist in the negative sentiment and made the sentiment analysis models "confuse" to generalize the unseen data. Second, the lack of the training data also can be the cause because there is not enough data to represent the unseen data with bias on the data label. Third, some words do not available in the pre-trained Glove Twitter word embedding because the pre-trained Glove Twitter word embedding may not contain words that are exist in COVID-19 trend. This also cause the word embedding ignored some important words.

4.2 Incremental 2

This section covered result from the second increment processes that are the sentiment analysis web application about COVID-19 vaccine types such as Sinovac vaccine, AstraZeneca vaccine, Pfizer vaccine, and Moderna vaccine. This section showed the user interface that built with *Flask* framework for *Python* web development and the black-box testing for each function.

4.2.1 Sentiment Analysis Web Application User Interface

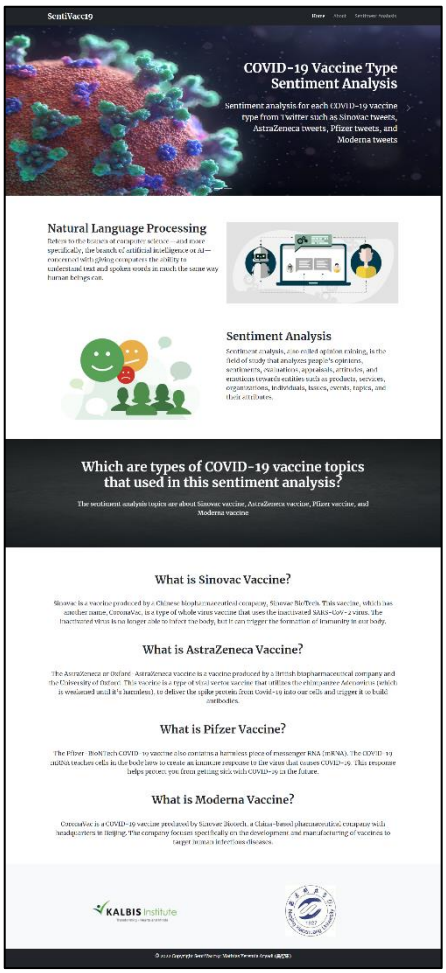


Figure 4.26 Sentiment Analysis Web Application: Home Page

The Figure 4.34 shows the home page for the sentiment analysis web application as the first page or landing page when user access the website. The home page simply shows a brief topic introduction that contained such as NLP (Natural Language Processing), sentiment analysis, Sinovac vaccine, AstraZeneca vaccine, Pfizer vaccine, and Moderna vaccine.

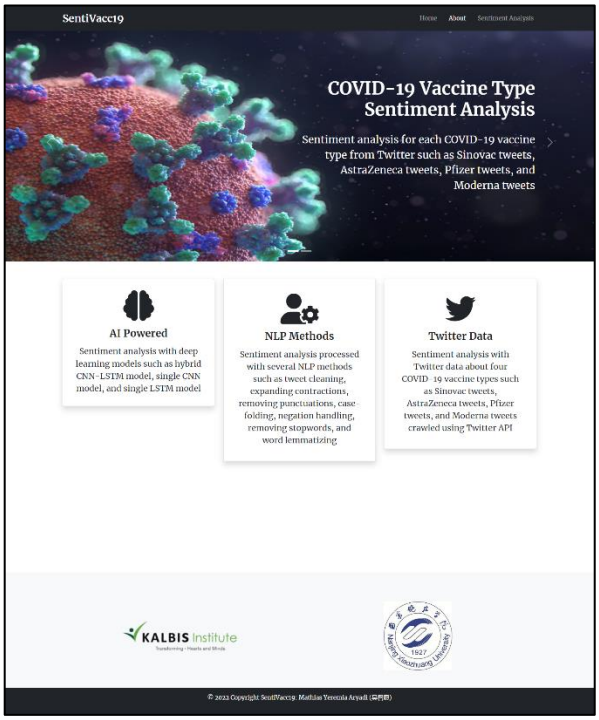


Figure 4.27 Sentiment Analysis Web Application: About Page

The Figure 4.35 shows the about page for the sentiment analysis web application as the second page of the website. The about page tells the user about what are technologies and methods behind the sentiment analysis for COVID-19 vaccine types.

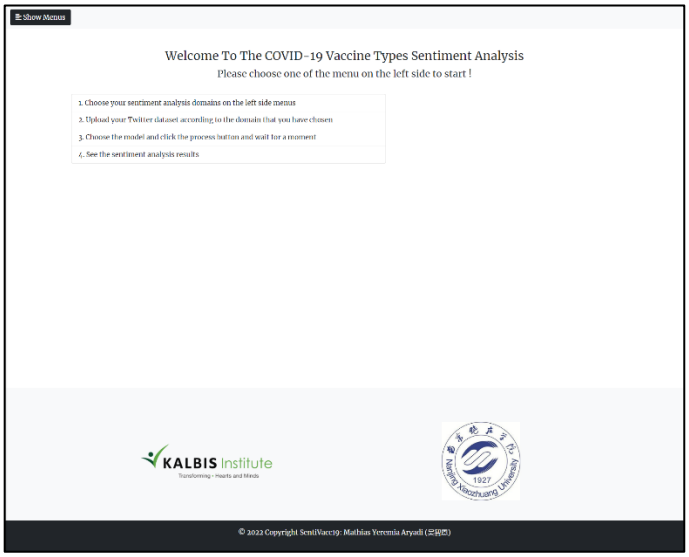


Figure 4.28 Sentiment Analysis Web Application: Sentiment Analysis Page

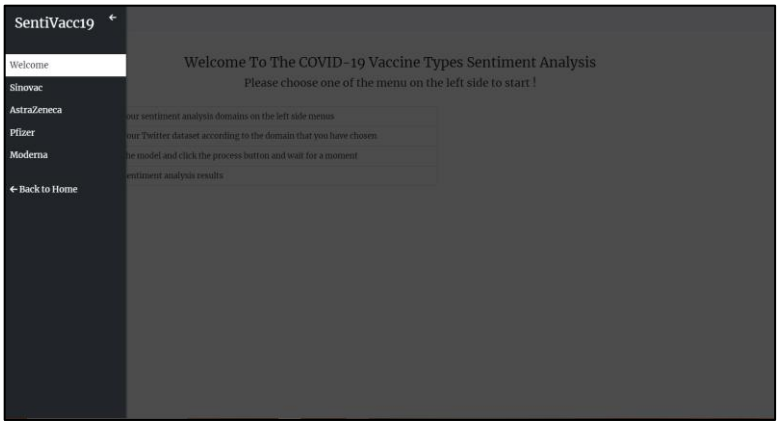


Figure 4.29 Sentiment Analysis Web Application: Sidebar Menu Triggered at Sentiment Analysis Page

The Figure 4.36 shows the sentiment analysis welcome page for the sentiment analysis web application as the third page of the website. The sentiment analysis welcome page above simply tells user the instruction to use the sentiment analysis web application. The Figure 4.37 shows the sub pages in the sentiment analysis page for each COVID-19 vaccine type topics and one menu to go back to the home page.

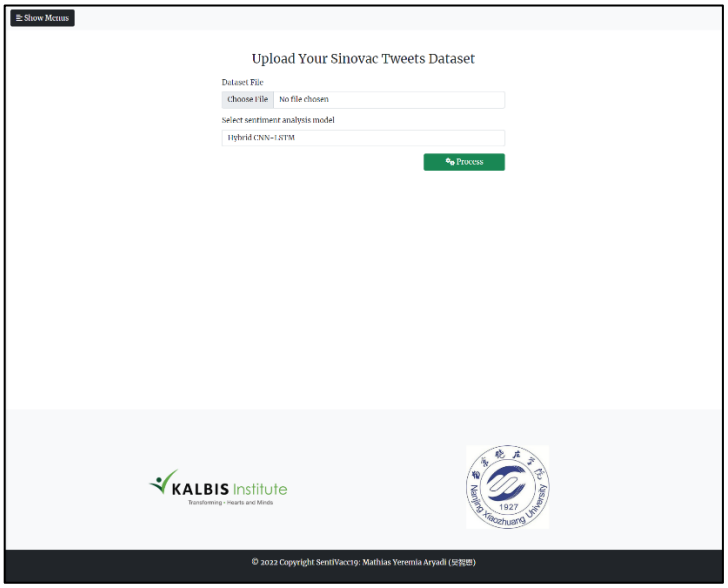


Figure 4.30 Sentiment Analysis Web Application: Sinovac Sentiment Analysis Page

The Figure 4.38 shows the Sinovac sentiment analysis page for the sentiment analysis web application as the first subpage of the sentiment analysis page. The Sinovac sentiment analysis page has a file input field and select input field. The user can upload the Sinovac implementation dataset, choose the sentiment analysis model to perform the sentiment classification, and click the process button so the system can upload, pre-process, and classify the Sinovac implementation dataset.

Figure 4.31 Sentiment Analysis Web Application: Sinovac Sentiment Analysis Result Page

The Figure 4.42 shows the Sinovac sentiment analysis result page for the sentiment analysis web application as the first subpage of the sentiment analysis page. The Sinovac sentiment analysis result page shows the sentiment analysis summary result to the user based on the chosen sentiment analysis model (the hybrid CNN-LSTM is used as example), where the result can be different when using different sentiment analysis model. The page shows the three panels about the number of tweets, the number of classified positive sentiments, and the number of classified negative sentiments; then, shows the top ten example classified tweets with the original tweets and its sentiment; finally, shows the sentiment analysis visualization in bar plot and pie plot.

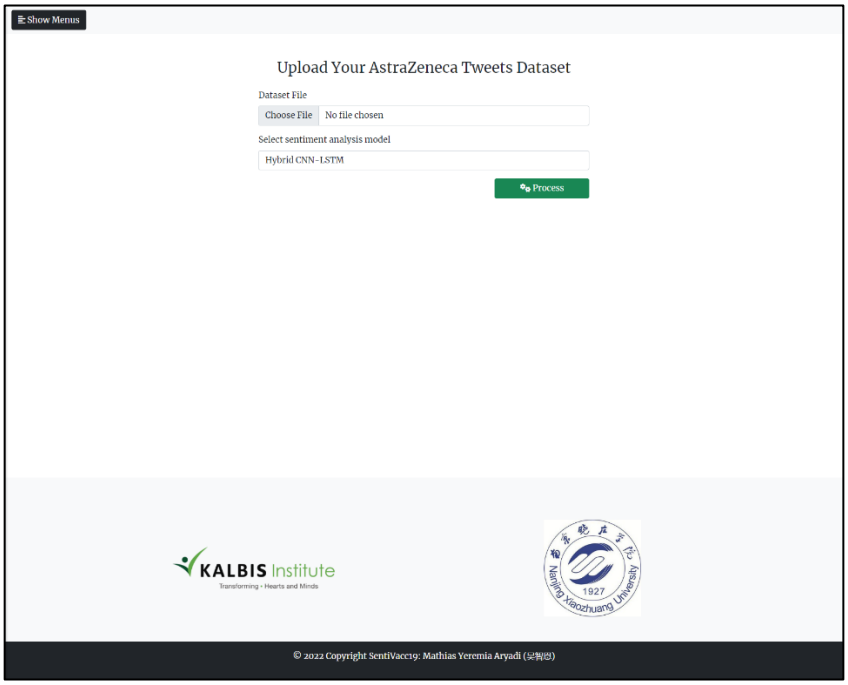


Figure 4.32 Sentiment Analysis Web Application: AstraZeneca Sentiment Analysis Page

The Figure 4.39 shows the AstraZeneca sentiment analysis page for the sentiment analysis web application as the second subpage of the sentiment analysis page. The AstraZeneca sentiment analysis page has a file input field and select input field. The user can upload the AstraZeneca implementation dataset, choose a sentiment analysis model to perform the sentiment classification, and click the process button so the system can upload, pre-process, and classify the AstraZeneca implementation dataset.

Figure 4.33 Sentiment Analysis Web Application: AstraZeneca Sentiment Analysis Result Page

The Figure 4.43 shows the AstraZeneca sentiment analysis result page for the sentiment analysis web application as the first subpage of the sentiment analysis page. The AstraZeneca sentiment analysis result page shows the sentiment analysis summary result to the user based on the chosen sentiment analysis model (the hybrid CNN-LSTM is used as example), where the result can be different when using different sentiment analysis model. The page shows the three panels about the number of tweets, the number of classified positive sentiments, and the number of classified negative sentiments; then, shows the top ten example classified tweets with the original tweets and its sentiment; finally, shows the sentiment analysis visualization in bar plot and pie plot.

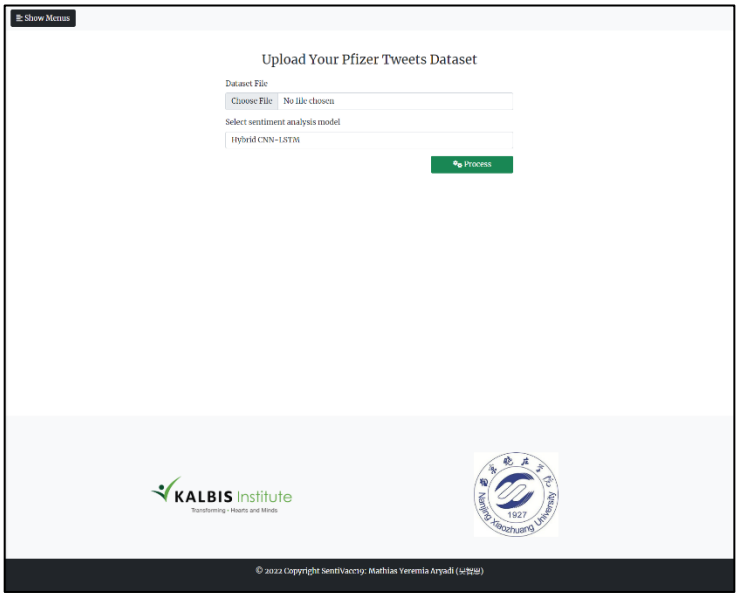


Figure 4.34 Sentiment Analysis Web Application: Pfizer Sentiment Analysis Page

The Figure 4.40 shows the Pfizer sentiment analysis page for the sentiment analysis web application as the second subpage of the sentiment analysis page. The Pfizer sentiment analysis page has a file input field and select input field. The user can upload the Pfizer implementation dataset, choose a sentiment analysis model to perform the sentiment classification, and click the process button so the system can upload, pre-process, and classify the Pfizer implementation dataset.

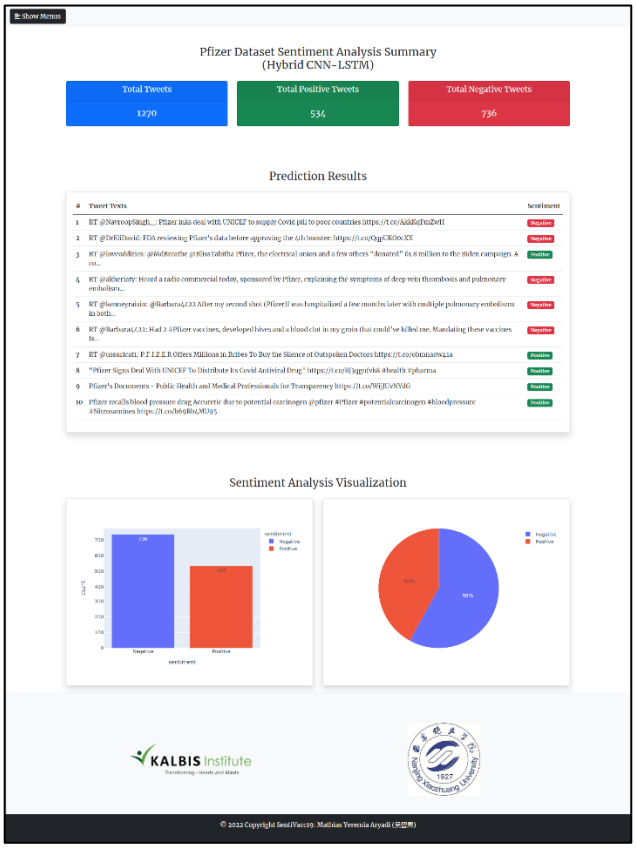


Figure 4.35 Sentiment Analysis Web Application: Pfizer Sentiment Analysis Result Page

The Figure 4.44 shows the Pfizer sentiment analysis result page for the sentiment analysis web application as the first subpage of the sentiment analysis page. The Pfizer sentiment analysis result page shows the sentiment analysis summary result to the user based on the chosen sentiment analysis model (the hybrid CNN-LSTM is used as example), where the result can be different when using different sentiment analysis model. The page shows the three panels about the number of tweets, the number of classified positive sentiments, and the number of classified negative sentiments; then, shows the top ten example classified tweets with the original tweets and its sentiment; finally, shows the sentiment analysis visualization in bar plot and pie plot.

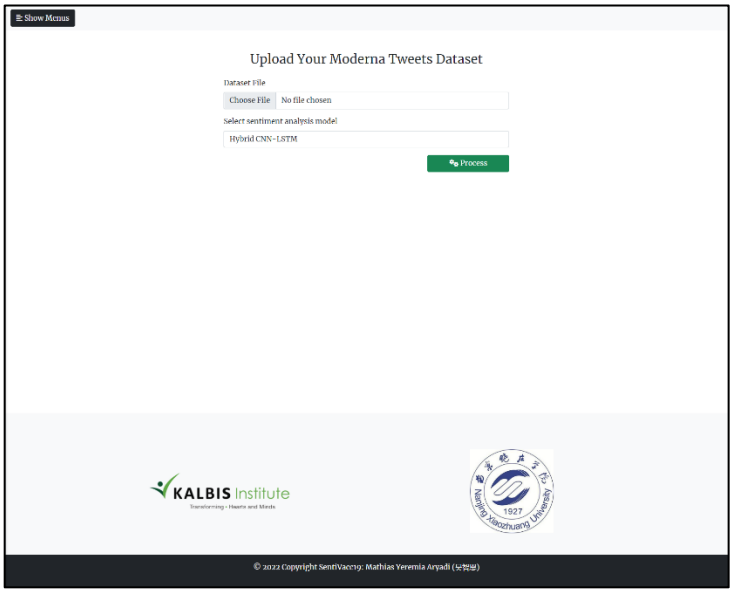


Figure 4.36 Sentiment Analysis Web Application: Moderna Sentiment Analysis Page

The Figure 4.41 shows the Moderna sentiment analysis page for the sentiment analysis web application as the second subpage of the sentiment analysis page. The Moderna sentiment analysis page has a file input field and select input field. The user can upload the Moderna implementation dataset, choose a sentiment analysis model to perform the sentiment classification, and click the process button so the system can upload, pre-process, and classify the Moderna implementation dataset.

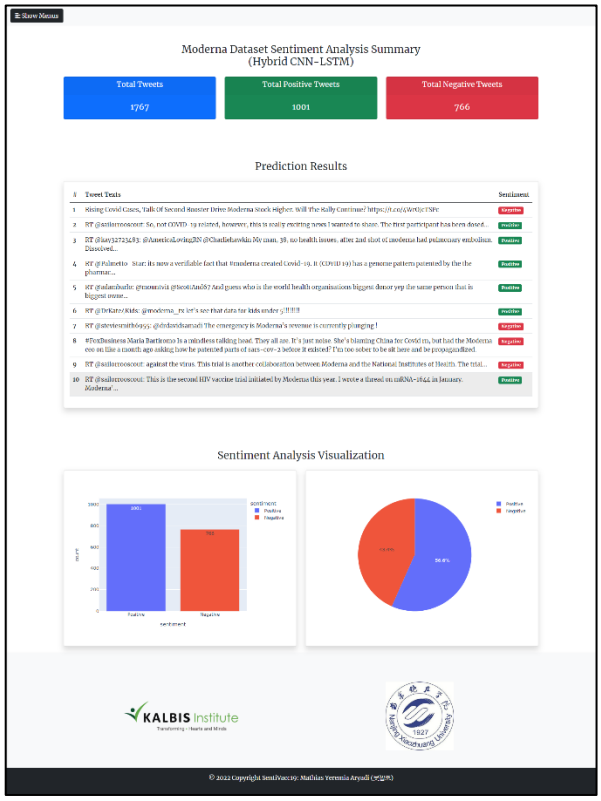


Figure 4.37 Sentiment Analysis Web Application: Moderna Sentiment Analysis Result Page

The Figure 4.45 shows the Moderna sentiment analysis result page for the sentiment analysis web application as the first subpage of the sentiment analysis page. The Moderna sentiment analysis result page shows the sentiment analysis summary result to the user based on the chosen sentiment analysis model (the hybrid CNN-LSTM is used as example), where the result can be different when using different sentiment analysis model. The page shows the three panels about the number of tweets, the number of classified positive sentiments, and the number of classified negative sentiments; then, shows the top ten example classified tweets with the original tweets and its sentiment; finally, shows the sentiment analysis visualization in bar plot and pie plot.

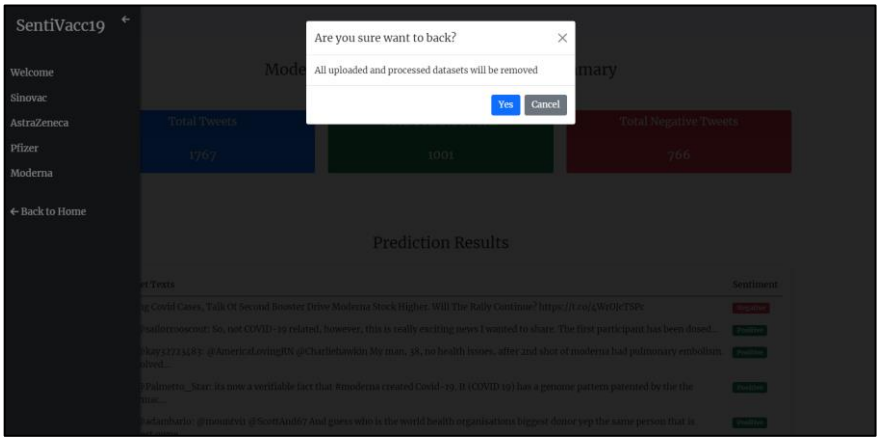


Figure 4.38 Sentiment Analysis Web Application: Reset All Sentiment Analysis Results

The Figure 4.46 shows the reset all sentiment analysis results for the sentiment analysis web application as the six subpages of the sentiment analysis page. The back to home button shows the confirmation modal to the users before they quit the sentiment analysis page and back to the home page, that the system will reset all the sentiment analysis results by removing all the uploaded, pre-processed, and classified file dataset.

4.2.2 Back-Box Testing

Test Scenario	***Test Case***	***Expected Output***	***Test Result***	***Summary***
Home Page	*Open the home page by clicking the home menu in navigation bar*	*Show the home page*	*Success*	*Normal*
About Page	*Open the about page by clicking the about menu in navigation bar*	*Show the about page*	*Success*	*Normal*
Sentiment Analysis Page	*Open the about page by clicking the sentiment analysis menu in navigation bar*	*Show the sentiment analysis welcome page*	*Success*	*Normal*

Sinovac Sentiment Analysis Page	*Open the Sinovac sentiment analysis page by clicking the Sinovac menu in sidebar*	*Show the Sinovac sentiment analysis page*	*Success*	*Normal*
Sinovac Sentiment Analysis Form Validation	*Make the file input field empty and upload a non-CSV format*	*Show the form validation error message*	*Success*	*Normal*
Sinovac Sentiment Analysis Result Page	*Upload the proper Sinovac dataset file, choose the sentiment analysis model, and clicked the process button*	*Show the Sinovac sentiment analysis result page with overview panels, example of classified sentiment, and the visualization*	*Success*	*Normal, only the bar plot and the pie plot little bit unresponsive on small screen*
AstraZeneca Sentiment Analysis Page	*Open the AstraZeneca sentiment analysis page by clicking the AstraZeneca menu in sidebar*	*Show the AstraZeneca sentiment analysis page*	*Success*	*Normal*
AstraZeneca Sentiment Analysis Form Validation	*Make the file input field empty and upload a non-CSV format*	*Show the form validation error message*	*Success*	*Normal*
AstraZeneca Sentiment Analysis Result Page	*Upload the proper AstraZeneca dataset file, choose the*	*Show the AstraZeneca sentiment analysis result page with*	*Success*	*Normal, only the bar plot and the pie plot little bit unresponsive*

	sentiment analysis model, and clicked the process button	*overview panels, example of classified sentiment, and the visualization*		*on small screen*
Pfizer Sentiment Analysis Page	*Open the Pfizer sentiment analysis page by clicking the Pfizer menu in sidebar*	*Show the Pfizer sentiment analysis page*	*Success*	*Normal*
Pfizer Sentiment Analysis Form Validation	*Make the file input field empty and upload a non-CSV format*	*Show the form validation error message*	*Success*	*Normal*
Pfizer Sentiment Analysis Result Page	*Upload the proper Pfizer dataset file, choose the sentiment analysis model, and clicked the process button*	*Show the Pfizer sentiment analysis result page with overview panels, example of classified sentiment, and the visualization*	*Success*	*Normal, only the bar plot and the pie plot little bit unresponsive on small screen*
Moderna Sentiment Analysis Page	*Open the Moderna sentiment analysis page by clicking the Moderna menu in sidebar*	*Show the Moderna sentiment analysis page*	*Success*	*Normal*
Moderna Sentiment Analysis Form Validation	*Make the file input field empty and upload a non-CSV format*	*Show the form validation error message*	*Success*	*Normal*
Moderna Sentiment	*Upload the proper Moderna dataset file,*	*Show the Moderna sentiment*	*Success*	*Normal, only the bar plot and the pie*

Analysis Result Page	*choose the sentiment analysis model, and clicked the process button*	*analysis result page with overview panels, example of classified sentiment, and the visualization*		*plot little bit unresponsive on small screen*
Reset Sentiment Analysis Result and Back to the Home Page	*Reset all the sentiment analysis result by clicking the back to home menu in sidebar*	*Delete all the uploaded, pre-processed, and classified data and file, then open the home page*	*Success*	*Normal*

4.2.3 Discussion

Based on the black-box testing, the sentiment analysis web application built using *flask* framework is working as expected and fulfilled the requirement. All the functions in the sentiment analysis web application have worked normally. The *flask* framework is used to build the sentiment analysis web application because it is easier to implement the trained sentiment analysis models with same programming language, which is, *Python*.

The sentiment analysis web application has successfully performed sentiment analysis on the four dataset topics. The sentiment analysis web application user interface also has fulfilled the purpose

CHAPTER 5
SUMMARY

5.1 Summary

In this work, I experimented with three deep learning models such as the hybrid CNN-LSTM, the single CNN model, and the single LSTM model with EDA (Easy Data Augmentation) for four different sentiment analysis domains or topics. The domains or topics are about COVID-19 vaccine types such as Sinovac vaccine, AstraZeneca vaccine, Pfizer vaccine, and Moderna vaccine. The Twitter data is used to retrieve the topics tweet texts. The aim of this work is to discover the performance difference between the hybrid CNN-LSTM, the single CNN model, and the single LSTM model with EDA (Easy Data Augmentation) for each sentiment analysis domains or topics; and build sentiment analysis web applcation. Therefore, this work can be summarized into several key notes:

1. This work has succeeded implemented the data augmentation on the sentiment analysis models using EDA (Easy Data Augmentation) method.
2. This work has succeeded built the hybrid CNN-LSTM, the single CNN model, and the single LSTM model with data augmentation for COVID-19 vaccine types sentiment analysis through model training, testing, and evaluation based on the first increment requirement and design.
3. This work has succeeded compose the sentiment analysis model performances comparison between the hybrid CNN-LSTM, the single CNN model, and the single LSTM model with data augmentation for COVID-19 vaccine types sentiment analysis.
4. This work has succeeded developing the sentiment analysis web application using *flask* framework for *Python* web development based on the second increment requirement and design.
5. On the Sinovac dataset topic, the single CNN model has the highest average of the recall, precision, and f1-score with 76%, 76%, and 76% respectively. In contrast, the single CNN model has the highest accuracy

on the test data with 76,05% and the loss is quite high with 10,3 loss compared to the hybrid CNN-LSTM model with 3,4 loss.

6. On the AstraZeneca dataset topic, the single CNN model has the highest average of the recall, precision, and f1-score with 80%, 81%, and 81% respectively. Interestingly, the hybrid CNN-LSTM model has better at the test data because of the lower loss with 80,43% accuracy and 2,17 loss compared the single CNN model with 80,43% accuracy and 3,08 loss.
7. On the Pfizer dataset topic, the hybrid LSTM-CNN model has the highest average of the recall, precision, and f1-score with 74%, 75%, and 74% respectively. The hybrid CNN-LSTM model also has the highest performance on the test data with 75,00% accuracy and 3,71 loss.
8. On the Moderna dataset topic, the hybrid LSTM-CNN model has the highest average of the recall, precision, and f1-score with 74%, 75%, and 74% respectively. The hybrid CNN-LSTM model also has the highest performance on the test data with 72,18% accuracy and 3,88 loss.
9. Based on the result, this work has main drawbacks that the sentiment analysis models has not achieved the optimal testing and evaluation yet because of the high losses. This drawback is cause by the limitation of the datasets: some tweet texts are truncated, biases in automated lexicon-based labeling, lack of the training data that represent the unseen data, and limitation of the pre-trained Glove Twitter word embedding.

5.2 Suggestion

Based on the summaries, there are several suggestions for further work to improve the drawbacks that existed in this work:

1. Improve the Twitter API search query so it can retrieve the non-truncated tweet texts.
2. It is highly suggested to label the text data manually with many time and resources, because human generalization more accurate than automated

lexicon-based data labeling in order to reduce the biases on the labeling process, even on the truncated text.

3. Increase more datasets for the real-world Twitter data in order to increase the sentiment analysis model generalization.
4. Implement and compare with other data augmentations such as BERT (Bidirectional Encoder Representations from Transformers), GAN (Generative Adversarial Network), or ELMo (Embedding from Language Model) methods.
5. Implement other machine learning or deep learning algorithms to enrich the model comparison for sentiment analysis

LIST OF REFERENCES

[1] Research Department, Statista, "Most popular social networks worldwide as of October 2021, ranked by number of active users," 26 November 2021. [Online]. Available: https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/. [Accessed 2021 December 26].

[2] Research Development, Statista, "Number of monetizable daily active Twitter users (mDAU) worldwide from 1st quarter 2017 to 3rd quarter 2021," 2021 November 01 . [Online]. Available: https://www.statista.com/statistics/970920/monetizable-daily-active-twitter-users-worldwide/. [Accessed 27 December 2021].

[3] B. Lui, "Sentiment Analysis: A Fascinating Problem," in *Sentiment Analysis and Opinion Mining*, California, Morgan & Claypool, 2012, p. 7.

[4] V. A. Fitri, R. Andreswari and M. A. Hasibuan, "Sentiment analysis of social media Twitter with case of Anti-LGBT campaign in Indonesia using Naïve Bayes, decision tree, and random forest algorithm," in *Procedia Computer Science*, Bandung, 2019.

[5] R. Novendri, A. S. Callista, D. N. Pratama and C. E. Puspita, "Sentiment Analysis of YouTube Movie Trailer Comments Using Naïve Bayes," *Bulletin of Computer Science and Electrical Engineering,* vol. 1, no. 1, pp. 26-32, 2020.

[6] Pristiyono, M. Ritonga, M. A. A. Ihsan, A. Anjar and F. H. Rambe, "Sentiment analysis of COVID-19 vaccine in Indonesia using Naïve Bayes Algorithm," *IOP Conference Series: Materials Science and Engineering,* vol. 1088, no. 1, pp. 12-45, 2021.

[7] M. Birjali, M. Kasri and A. Beni-Hssane, "A comprehensive survey on sentiment analysis: Approaches, challenges and trends," *Knowledge-Based Systems,* vol. 226, 2021.

[8] U. D. Gandhi, P. Malarvizhi Kumar, G. Chandra Babu and G. Karthick, "Sentiment Analysis on Twitter Data by Using Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM)," *Wireless Personal Communications,* 2021.

[9] P. F. Muhammad, R. Kusumaningrum and A. Wibowo, "Sentiment Analysis Using Word2vec and Long Short-Term Memory (LSTM) for Indonesian Hotel Reviews," in *Procedia Computer Science*, Semarang, 2021.

[10] S. prabha.K.S and P. N. Karthikayan, "For Movie Reviews, A Sentiment Analysis using Long Short Term Memory Networks," *Turkish Journal of Computer and Mathematics Education,* vol. 12, no. 9, pp. 1758-1766, 2021.

[11] A. U. Rehman, A. K. Malik, B. Raza and W. Ali, "A Hybrid CNN-LSTM Model for Improving Accuracy of Movie Reviews Sentiment Analysis," *Multimedia Tools and Applications,* vol. 78, no. 18, pp. 26597-26613, 2019.

[12] A. H. Ombabi, W. Ouarda and A. M. Alimi, "Deep learning CNN–LSTM framework for Arabic sentiment analysis using textual information shared in social networks," *Social Network Analysis and Mining,* vol. 10, no. 1, 2020.

[13] W. Yue and L. Li, "Sentiment analysis using word2vec-cnn-bilstm classification," in *2020 7th International Conference on Social Network Analysis, Management and Security, SNAMS 2020*, 2020.

[14] A. Giachanou and F. Crestani, "Like it or not: A survey of Twitter sentiment analysis methods," *ACM Computing Surveys,* vol. 49, no. 2, 2016.

[15] Twitter, "Twitter API," 21 November 2021. [Online]. Available: https://developer.twitter.com/en/docs/twitter-api. [Accessed 07 March 2022].

[16] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data,* vol. 6, no. 1, 2019.

[17] C. Shorten, T. M. Khoshgoftaar and B. Furht, "Text Data Augmentation for Deep Learning," *Journal of Big Data,* vol. 8, no. 1, 2021.

[18] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie and L. Farhan, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data,* vol. 8, no. 1, 2021.

[19] S. Khan, H. Rahmani, S. A. A. Shah and M. Bennamoun, "A Guide to Convolutional Neural Networks for Computer Vision," *Synthesis Lectures on Computer Vision,* vol. 8, no. 1, pp. 1-207, 2018.

[20] M. Bernico, "Deep Learning Quick Reference Useful hacks for training and optimizing deep neural networks with TensorFlow and Keras," in *Long Short Term Memory Networks*, Birmingham, Packt, 2018, pp. 290-293.

[21] P. K. Jain, V. Saravanan and R. Pamula, "A Hybrid CNN-LSTM: A Deep Learning Approach for Consumer Sentiment Analysis Using Qualitative User-Generated Contents," *ACM Transactions on Asian and Low-Resource Language Information Processing,* vol. 20, no. 5, pp. 1-15, 2021.

[22] S. Liu, K. Lee and I. Lee, "Document-level multi-topic sentiment classification of Email data with BiLSTM and data augmentation," *Knowledge-Based Systems,* vol. 197, 2020.

[23] T. Ho Huong and V. Truong Hoang, "A data augmentation technique based on text for Vietnamese sentiment analysis," *Association for Computing Machinery,* 2020.

[24] B. Acharya and K. Sahu, "Software Development Life Cycle Models: A Review Paper," *International Journal of Advanced Research in Engineering and Technology,* vol. 11, no. 12, pp. 169-176, 2020.

[25] Deepanshi, "In-depth understanding of Confusion Matrix," Analytics Vidhya, 18 May 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/in-depth-understanding-of-confusion-matrix/. [Accessed 03 May 2022].

[26] A. C. Müller and S. Guido, "Model Evaluation and Improvement," in *Introduction to Machine Learning with Python*, The United States of America, O'Reilly Media, Inc., 2017, pp. 282-284.

[27] ReqTest, "Black Box Testing – Understanding the Basics," 08 August 2019. [Online]. Available: https://reqtest.com/testing-blog/black-box-testing/#:~:text=Black%20box%20testing%20checks%20scenarios,on%20entering%20an%20incorrect%20password.. [Accessed 15 March 2022].

[28] H. Bhasin, Python Basics: A Self-Teaching Introduction, Dulles: Mercury Learning and Information LLC., 2019.

[29] M. Griberg, Flask Web Development: Developing Web Applications With Python, United States of America: O'Reilly Media Inc., 2018.

[30] V. Bonta, N. Kumaresh and N. Janardhan, "A Comprehensive Study on Lexicon Based Approaches for Sentiment Analysis," *Asian Journal of Computer Science and Technology,* vol. 8, no. S2, pp. 1-6, 2019.

[31] F. Illia, M. P. Eugenia and S. A. Rutba, "Sentiment Analysis on PeduliLindungi Application Using TextBlob and VADER Library," *ICDCSOS,* pp. 278-288, 2021.

[32] S. Sanyal and M. Kumar Barai, "Comparative Study on Lexicon-based sentiment analysers over Negative sentiment," *International Journal of Electrical, Electronics and Computers,* vol. 6, no. 6, pp. 1-13, 2021.

[33] J. Wei and K. Zou, "EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks," *arXiv preprint arXiv:1901.11196,* 2019.