

# **GRAMMAR CORRECTION USING RULE BASED SYSTEM**

Project Report was submitted in partial fulfilment of the Bachelor of  
Technology Degree requirement

in

**Computer Science and Engineering**

By

Chakori Chaturvedi (181420)

Under the supervision of

Ananay Bakshi

to



Department of Computer Science & Engineering  
**Jaypee University of Information Technology Waknaghat, Solan-  
173234, Himachal Pradesh**

## CANDIDATE'S DECLARATION

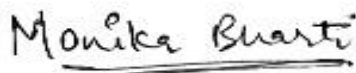
I hereby declare that the work presented in this report titled "GRAMMAR CORRECTION USING RULE BASED SYSTEM" in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering/Information Technology submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from January 2022 to May 2022 under the supervision of Ananay Bakshi(Specialist Programmer).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.



Chakori Chaturvedi, 181420

This is to certify that the above statement made by the candidate is true to the best of my knowledge.



Supervisor Name: Dr. Monika Bharti

Designation: Associate Professor

Department name: Computer Science

Dated: 27/05/2022



Supervisor Name: Ananay Bakshi

Designation: Specialist Programmer

Department name: STG(Infosys)

Dated: 27/05/2022

## **ACKNOWLEDGEMENT**

Firstly, I express my heartiest thanks and gratefulness to almighty God for his divine blessing makes it possible to complete the project work successfully.

I am really grateful to my supervisor, Ananay Bakshi (Specialist Programmer), for his extensive knowledge and genuine interest in the field of Artificial Intelligence, which enabled me to complete my project. His never-ending patience, scholarly direction, constant encouragement, constant and energetic supervision, constructive criticism, helpful suggestions, and reading numerous poor versions and editing them at all stages allowed this project to be completed.

I'd like to thank Ananay Bakshi (Specialist Programmer), for his invaluable assistance in completing my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Chakori Chaturvedi (181420)

# TABLE OF CONTENTS

<b>TITLE</b>	i
<b>CANDIDATE’S DECLARATION</b>	ii
<b>ACKNOWLEDGEMENTS</b>	iii
<b>TABLE OF CONTENTS</b>	iv
<b>LIST OF FIGURES</b>	vii
<b>LIST OF GRAPHS</b>	viii
<b>LIST OF TABLES</b>	ix
<b>LIST OF ABBREVIATIONS</b>	x
<b>ABSTRACT</b>	xi
<b>CHAPTER 1 INTRODUCTION</b>	
1.1 Introduction	12
1.2 Problem Statement	13
1.3 Objectives	13
1.4 Methodology	14
1.4.1 Spelling Checker	14
1.4.2 Word Tokenization	14
1.4.3 Part Of speech Tagging	14
1.4.4 Chunking Phrases	15
1.4.5 Rule Matching	15
1.4.6 Output	15
1.4.7 Packaging the project	15
1.5 Organization	15
<b>CHAPTER 2 LITERATURE REVIEW</b>	
2.1 Rule Based	18
2.2 Minimize the Error	18
2.3 Rule Based	18
2.4 Grammar Checker	19

2.5	Chunking Phrases	19
2.6	Rule Based system	19
2.7	A Rule Based style and Grammar checker	20
2.8	Sentence Correctness	21
2.9	Sentence Correctness	21

## **CHAPTER 3 SYSTEM DEVELOPMENT**

3.1	Input Output Parameters	22
3.2	Article Error	22
3.3	Because Error	23
3.4	Tense Error	25
3.5	But Error	26
3.6	Capitalization Error	27
3.7	Subject Verb Agreement Error	29
3.8	Reflex Pronoun Error	30
3.9	And Error	31
3.10	Spelling Error	33
3.11	Apostrophe Error	34
3.12	Pluralization Error	35
3.13	Although Though Error	36
3.14	Either Neither Error	38
3.15	Sentence Formatter	39

## **CHAPTER 4 PERFORMANCE ANALYSIS**

4.1	Word Tokenization	40
4.2	POS Tagging	41
4.3	Chunking Phrases	41
4.4	Rule Based Matching	41
4.4.1	Article Error	42
4.4.2	Because Error	44
4.4.3	Tense Error	45
4.4.4	But Error	47

## **CHAPTER 5 CONCLUSIONS**

5.1	Conclusion	48
5.2	Future Scope	49
5.3	Applications	49

<b>REFERENCES</b>	<b>50</b>
-------------------	-----------

## LIST OF FIGURES

<b>Figure 4.1</b>	Word tokenization.....	40
<b>Figure 4.2</b>	Word Tokenization.....	40
<b>Figure 4.3</b>	POS Tagging.....	41
<b>Figure 4.4</b>	POS Tagging.....	41
<b>Figure 4.5</b>	Rule Matching.....	42
<b>Figure 4.6</b>	Article Error.....	43
<b>Figure 4.7</b>	Article Error.....	43
<b>Figure 4.8</b>	Because Error.....	44
<b>Figure 4.9</b>	Because Error.....	45
<b>Figure 4.10</b>	Tense Error.....	46
<b>Figure 4.11</b>	Tense Error.....	46
<b>Figure 4.12</b>	But Error.....	47

## LIST OF GRAPHS

<b>Graph 3.1</b>	Article Error.....	23
<b>Graph 3.2</b>	Because Error.....	24
<b>Graph 3.3</b>	Tense Error.....	26
<b>Graph 3.4</b>	But Error.....	27
<b>Graph 3.5</b>	Capitalization Error.....	29
<b>Graph 3.6</b>	Subject Verb Agreement Error.....	30
<b>Graph 3.7</b>	Reflex Pronoun Error.....	31
<b>Graph 3.8</b>	And Error.....	32
<b>Graph 3.9</b>	Spelling Error.....	34
<b>Graph 3.10</b>	Apostrophe Error.....	35
<b>Graph 3.11</b>	Pluralization error.....	36
<b>Graph 3.12</b>	Although Though Error.....	37
<b>Graph 3.13</b>	Either neither.....	38



## LIST OF TABLES

<b>Table 4.1</b>	I/O Results .....	40
------------------	-------------------	----

## **LIST OF ABBREVIATIONS**

<i>POS</i>	Part Of Speech
<i>NLP</i>	Natural Language Processing
<i>NLTK</i>	Natural language Toolkit

## **ABSTRACT**

Spelling and grammar checkers are frequently used programmes that are designed to aid in the detection and correction of a variety of writing problems. There are currently no proofreading algorithms that can detect both spelling and grammar problems in English text. Currently, the rule matcher module employs 14 rules to detect, correct, and explain frequent punctuation, word choice, and spelling problems.

This paper describes a new way for checking "English" grammar. For morphological analysis and rule-based systems, this system uses a full-form lexicon. In this method, we offer a system that matches a set of rules against an input English sentence that has been POS tagged at the very least. This method is similar to the statistics-based method, except in our method, all of the rules are created manually.

# CHAPTER -1

## INTRODUCTION

### 1.1 Introduction

Grammar checking is a common problem in natural language processing. Grammar checkers are useful in a variety of contexts, such as text generation and language learning. Such programmes aim to detect grammatical errors in the input text, such as incorrect use of person, number, case, or gender, wrong verb government, and incorrect word order, among other things. A grammar checker is typically used in tandem with a spellchecker, which detects spelling errors in specific words. Spell checkers, in most situations, are unable to correct even simple grammatical problems, such as wrong article selection (as in the term "an box").

Despite the fact that a spellchecker is now a fundamental part of any current text production system, grammar checking is still only found in large commercial programmes like Microsoft Office or WordPerfect Office. Some grammar checkers are included in larger software packages or are provided as online services from independent companies.

In today's world, this condition is steadily changing. More natural language processing technologies should become available for greater use as open-source software grows in popularity.

The purpose of this thesis is to develop an open-source English language style and grammar checker. Despite the fact that all major Open-Source word processors feature spell checking, none have a style and grammar checker. Furthermore, such a feature is not available as a free standalone programme. As a result of this thesis, a free programme will be created that may be used as a standalone style and grammar checker or integrated into a word processor.

Grammar checking is a more difficult process, and most open projects are still much beyond the capabilities of well-known proofing tools like those found in Microsoft Word.

## 1.2 Problem Statement

This project is based on NLP (Natural Language Processing), an area of linguistics, computer science, and artificial intelligence concerned with how to train computers to process and evaluate huge amounts of natural language data.

This project's purpose is to develop an open-source English style and grammar checker. Despite the fact that all major Open-Source word processors feature spell checking, none have a style and grammar checker. Furthermore, such a feature is not available as a free standalone programme. As a result, the project's final product will be a free style and grammar checker that may be used standalone or as part of a word processor.

## 1.3 Objectives

The following is a list of the project's overall objectives:

- The goal is to create a model that correctly removes all grammatical faults from a sentence entered by the user.
- To correct punctuation and sentence structure mistakes while maintaining overall accuracy.
- It improves sentence processing, resulting in a better dependency tree and, as a result, better information extraction.
- Improving the accuracy of the grammar correction project by using rules.
- Developing new grammatical correction rules that aren't yet included in open-source grammar correction datasets.
- It should be quick, in the sense that it should be able to be used interactively.

## 1.4 Methodology

### 1.4.1 Spelling Checker

In any text processing or analysis, checking for spelling is a must. This feature is provided by the python module spellchecker, which finds words that may have been misspelt and suggests plausible repairs.

INPUT: Amazing spiderman is out in theaters.

OUTPUT: Amazing spiderman is out in theatres.

### 1.4.2 Word Tokenization

Tokenization is the process of splitting or tokenizing a string of text into a list of tokens. Tokens are components; for example, a token is a word in a sentence, and a phrase is a token in a paragraph.

INPUT: I ate a apple

OUTPUT: 'I', 'ate', 'a', 'apple'

### 1.4.3 Part of Speech Tagging

- In NLTK, POS Tagging is a method of marking up words in text format for a specific segment of a speech based on their definition and context.
- CC, CD, EX, JJ, MD, NNP, PDT, PRP\$, TO, and other NLTK POS tagging examples include: CC, CD, EX, JJ, MD, NNP, PDT, PRP\$, TO, and so on.
- The POS tagger assigns grammatical information to each word in a phrase.

INPUT: 'I', 'ate', 'a', 'apple'

OUTPUT: ('I', 'PRP'), ('ate', 'VBP'), ('a', 'DT'), ('apple', 'NN')

#### 1.4.4 Chunking Phrases

One of the primary purposes of chunking is to organise information into "noun phrases." These are phrases made up of one or more words that include a noun, possibly some descriptive words, a verb, and possibly an adverb. The objective is to put nouns together with the words that are related to them.

#### 1.4.5 Rule Matching

- Rule of replacing a to an or vice versa according to the structure of sentence.  
INPUT: ('I', 'PRP'), ('ate', 'VBP'), ('a', 'DT'), ('apple', 'NN')  
OUTPUT: 'I', 'ate', 'an', 'apple'.
- Rule of replacing small letters to capital letters according to structure of sentence.  
INPUT: ('shyam', 'NN'), ('is', 'CC'), ('my', 'PRP\$'), ('friend', 'NN')  
OUTPUT: 'Shyam', 'is', 'my', 'friend'.
- Rule of replacing incorrect words with correct words.  
INPUT: ('Amazing', 'JJ'), ('spiderman', 'NN'), ('is', 'VBZ'), ('out', 'RP'), ('in', 'IN'), ('theaters', 'NNS')  
OUTPUT: Amazing spiderman is out in theatres.

#### 1.4.6 Output

Integrating the output of all the rules to create the final grammatically correct query

FINAL OUTPUT: This is an apple.

#### 1.4.7 Packaging the project

A function that user could integrate in their project and call that function to directly get the grammar corrected sentence. Making the project available as a pip installable package

### 1.5 Organization

It turns out that a grammar checker can be implemented in one of three ways. I'll use the following terms to describe them:

**Syntax-based checking:**

[Jensen et al, 1993] describes syntax-based checking. This method parses a text entirely, analysing each sentence and assigning a tree structure to each sentence. If the parsing fails, the text is regarded as wrong.

**Statistics-based checking:**

[Attwell, 1987] describes statistics-based checking. A POS-annotated corpus is used to generate a list of POS tag sequences in this method. Some sequences will be extremely common (for example, determiner, adjective, and noun, as in the old man), while others will almost certainly not be seen at all (for example determiner, determiner, adjective). Sequences that appear frequently in the corpus can be assumed to be correct in other texts as well; nevertheless, rare sequences may constitute errors.

**Rule-based checking:**

As used in this project, rule-based checking. A set of rules is matched against a text that has at least been POS tagged in this method. This method is similar to the statistics-based method, but all of the rules are written by hand.

The syntax-based technique has the advantage that grammar checking is always complete if the grammar is full, i.e., the checker will discover any incorrect sentence, no matter how subtle the error is. Unfortunately, the checker will only be able to recognise that the sentence is erroneous; it will not be able to tell the user what the issue is. This necessitates the use of additional rules that parse ill-formed sentences as well. It is wrong if a sentence can only be parsed using this extra rule. Constraint relaxation is the name of this approach.

The syntax-based approach, on the other hand, has a key flaw: it necessitates a comprehensive grammar that covers all forms of writings. Despite the existence of numerous grammar theories, there is still no publicly available strong broad-coverage parser. Parsers also suffer from natural language ambiguity; thus, even accurate phrases frequently produce many results.



Statistics-based parsers, on the other hand, run the danger of making their results difficult to understand: if the system makes a false alarm, the user will be confused as to why his input was deemed erroneous because there is no specific error message. In order to understand the system's judgement, even developers would require access to the corpus on which the system was trained.

Another issue is that a threshold must be established to distinguish the uncommon but correct constructs from the uncommon but wrong ones. Surely, this work might be delegated to the user, who would be required to enter a value between 0 and 100. The concept of a threshold, on the other hand, does not fully fit with the assumption that sentences are usually either correct or erroneous, except from concerns of style and manufactured corner situations.

Due to the limitations with the other approaches, this thesis will design a strictly rule-based system. A rule-based checker, unlike a syntax-based checker, will never be complete, meaning it will always miss errors. It does, however, have a number of advantages:

The software can check the text while it is being entered and provide rapid response, so a sentence does not have to be full to be checked.

It's simple to set up because each rule includes a detailed description and can be switched on and off separately.

It can provide thorough error warnings with useful comments, as well as grammar rules explanations.

Its users can simply extend it because the rule system is straightforward to understand, at least for many common yet simple error instances.

It can be developed in stages, beginning with a single rule and gradually expanding it rule by rule.

## **CHAPTER 2**

### **LITERATURE REVIEW**

We referred to many research papers, journals and websites. They are listed here ordered year wise.

#### **2.1 Rule Based Approach**

By Asanilta Fahda

Asanilta Fahda et al. provided a rule-based strategy for creating a prototype for an Indonesian spelling and grammar checker using a combination of rules and statistical methodologies. The rule matcher module employs 38 rules to identify, rectify, and locate typical punctuation, word choice, and spelling problems. They employ the trigram language model for grammar checking from POS tags, phrase chunks, or tokens to recognise sentences with improper structures. Based on document analysis, the system's complete accuracy is 83.18 percent.

#### **2.2 Minimize the Errors**

By Shashi pal Singh

According to Shashi Pal Singh et al., one should strive to minimise errors when using the language. The fewer the errors, the better the communication will be. They are developing a frequency-based spell checker and a rule-based grammar checker for the English language to aid in this goal. The grammar checker concentrates on detecting and correcting tense errors.

#### **2.3 Rule Based**

By Nivedita S. Bhirud

Nivedita S. Bhirud et.al., described that review look at past, present and the future in the present context for development of various Natural Language grammar on the till date. The grammar checkers of a few Indian languages as well as foreign Languages discussed with the different characteristics are conclude in this survey. They are covering the grammar checkers for various languages and approaches, methodologies and performance evaluation and also common grammatical errors which would be introduced to new tool and system as a whole along with the key concept and grammar checker internals. They observed that professionally grammar checker is available for English language, while for most other languages, the work is in progress. The grammar checker for Marathi language could not have been reproduced.

## **2.4 Grammar Checker**

By Lata Bopche and Gauri Dhopavakar

Lata Bopche and Gauri Dhopavakar describes a method for grammar checker. This system made up to rule-based system and morphological analysis for lexicon. The basic process like POS tagging, tokenization and morphological analysis by passed to the input text. The result is only for simple sentence in their system. The input sentence has the same number of words only checks by the system.

## **2.5 Chunking Phrases**

By D. Naber

D. Naber described a possible text error and list return. Each sentence is divided into chunks, for example, noun phrases are assigned to their part-of-speech tags, and errors are detected. The checker's predefined error rules match the content. In the circumstance of the match, the rule matches contain an error. Rules explain errors as patterns of words, part-of-speech tags, and chunks. Each rule provides a description of the error, which is displayed to the user. It is adequate to allow users to define their own rules while still allowing the rule-based approach system to identify many problems.

## **2.6 Rule Based System**

By Mandeep Singh

Mandeep Singh et.al., implemented the system executes morphological analysis using the rule-based system for lexicon and POS tagging and phrase chunk. The suggestion provided for detecting the grammatical errors in texts used the grammar checking software. This system's prime attraction is that the detected errors are presented in detailed form. It comes with the suggestion as well. The compound sentences as well as complex sentences are dealt with.

Specially designed error detection rules are supported by system for texts. Agreements and order of word in phrases generate several grammatical errors.

## **2.7 A Rule Based Style and Grammar Checker**

By Daniel Naber

The goal of this thesis is to create an English language style and grammar checker that is open-source. Despite the fact that all major Open-Source word processors have spell checking, none of them include a style and grammar checker. Also, such a function is not available as a stand-alone free programme. As a result, this thesis will produce a free tool that may be used as a standalone style and grammar checker or as part of a word processor.

This thesis' style and grammar checker takes a document and generates a list of likely errors. Each word in the text is given a part-of-speech tag, and each sentence is broken down into chunks, such as noun phrases, to discover problems. The text is then compared to all of the checker's previously set error rules. If a rule matches, the text should contain an error at the match's location. The rules define mistakes as word patterns, part-of-speech tags, and chunks. Each rule also includes an error explanation that is displayed to the user.

The software will be built on a technology that I previously designed [Naber]. The present style and grammar checker, as well as the required part-of-speech tagger, will be re-implemented in Python. The rule system will be enhanced so that it may express rules that characterise errors at the phrase level rather than only at the word level. The integration with word processors will be improved so that errors can be noticed in real time, i.e. when text is being entered. The software will provide a solution for numerous problems, which may be used to replace the incorrect text with a single mouse click.

## **2.8 Sentence Correctness**

By Jahangir

Jahangir Md et. al., has done research on correctness of a sentence. Analysis of words based on n gram and POS tags are taken into consideration to check correctness of a sentence. With the help of POS tagger, every single word of sentence is allotted tag by the system. Tag sequence's probability is determined with the help of gram analysis. For sequence to be correct, the probability must be one or more. Bangla as well as English languages are compatible with the system. POS tagging is very vital as system is dependent on it. Manually tagged and automatically tagged sentences are checked by the author. After execution, the results looked very promising for languages like Bangla when compared with English. This is due to brown corpus method as it consists huge number of compound sentences.

## **2.9 Sentence Correctness**

By H. Kabir

Research study has been done on two pass parsing approach by H. Kabir et. al. It is mainly employed in analysing input text. Redundancy is one of the critical obstacles in the phrase structure. Above approach tries to reduce that redundancy effectively. Redundancy in grammar rules which are utilized in sentence analysis is also rectified. When a case result in failure, reparsing of tree is done using movement rules. POS guesser and Morphological disambiguation's module may result in system not working properly. Rest of the time, system runs properly. Structural mistakes and grammatical mistakes are checked by the grammar checker. Mostly these mistakes occur in declarative sentences. The grammar checker also try to suggest error corrections.

# CHAPTER 3

## SYSTEM DEVELOPMENT

### 3.1 Input Output Parameters

#### Input Parameters

nlp: list of sentence tokenizers

#### Output Parameters

error\_count: integer

correct\_text: string

### 3.2 Article Error

This method deals with the wrong use of articles in a sentence. Additionally, it performs required modifications to make the text free from article errors and returns the error count along with accurate text.

#### Usage:

```
from util.article import check_articleError
import nltk
text = input ()
print(check_articleError([nltk.pos_tag(text)]))
```

#### Example 1:

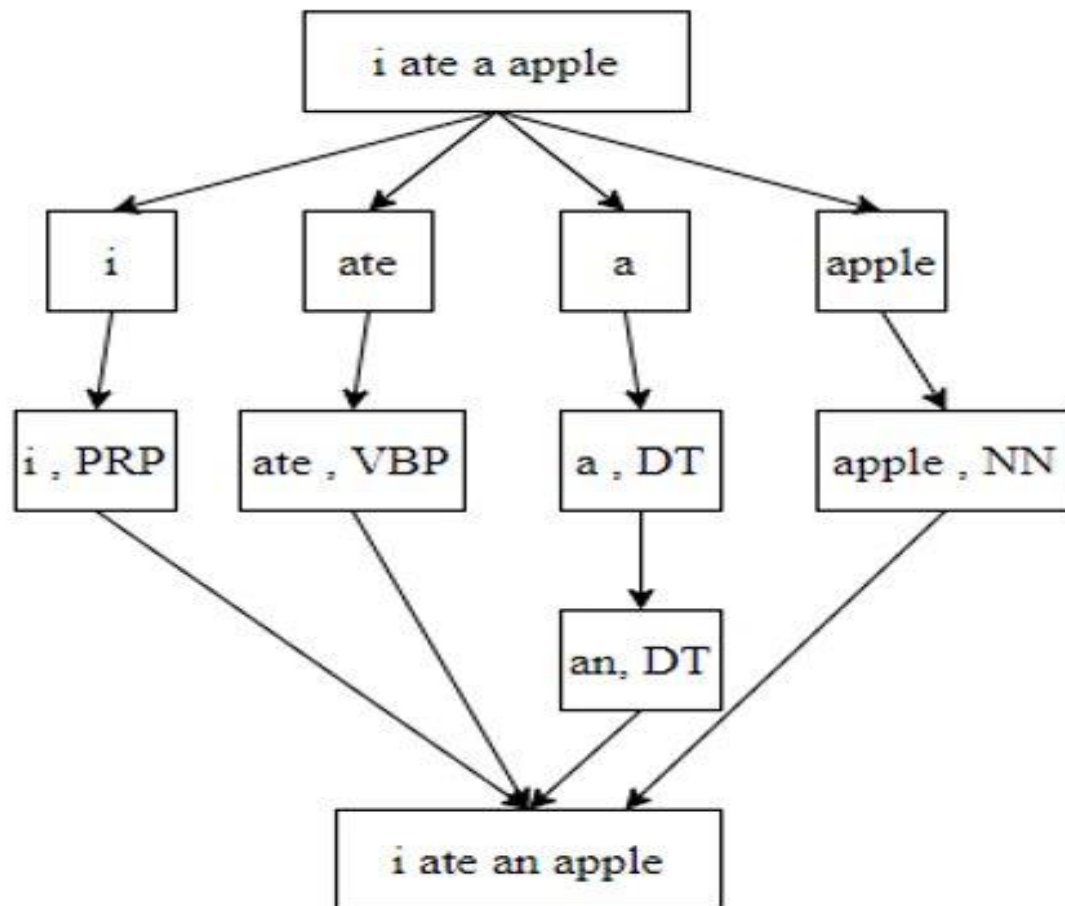
I/P: I ate a apple.

O/P: I ate an apple.

### Example 2:

I/P: Buy a book in a hour.

O/P: Buy a book in an hour.



Graph 3.1

### 3.3 Because Error

This error checks if text after using word 'because' incomplete sentence.

#### Usage:

```
from util.Because import check_becauseError
import nltk
text = input ()
print (check_becauseError ([nltk.pos_tag(text)]))
```

**Example 1:**

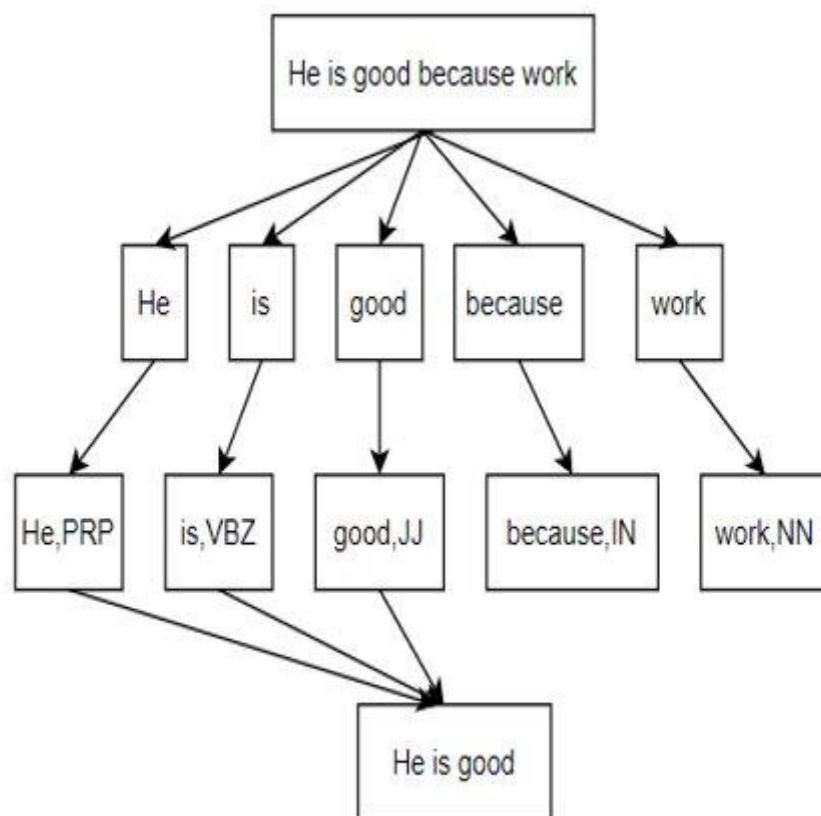
I/O: He is good because work.

O/P: He is good.

**Example 2:**

I/P: He is late because work.

O/P: He is late.



Graph 3.2



### 3.4 Tense Error

This error checks if the sentence has tense related errors.

#### Usage:

```
from util.tense import check_TenseError
import nltk
text = input ()
print (check_TenseError ([nltk.pos_tag(text)]))
```

#### Example 1:

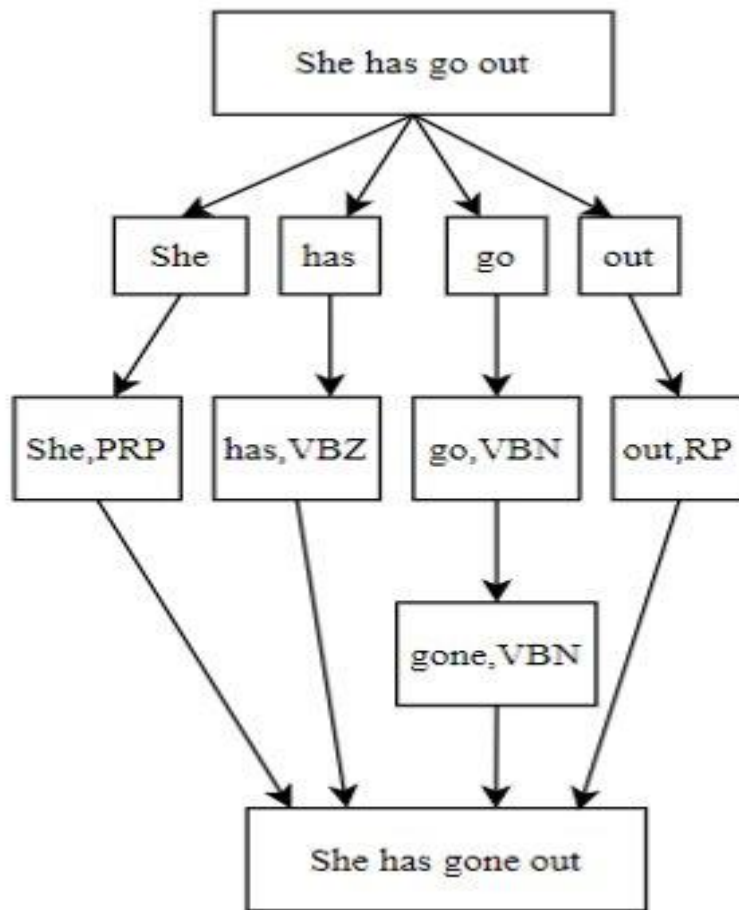
I/P: She has go out.

O/P: She has gone out.

#### Example 2:

I/P: She has know him for long time.

O/P: She has known him for long time.



Graph 3.3

### 3.5 But Error

This method checks for the usage of 'but' conjunction and 'but' can also be used as preposition followed by noun.

#### Usage:

```

from util.tense import check_ButError
import nltk
text = input()
print(check_ButError ([nltk.pos_tag(text)]))
  
```

**Example 1:**

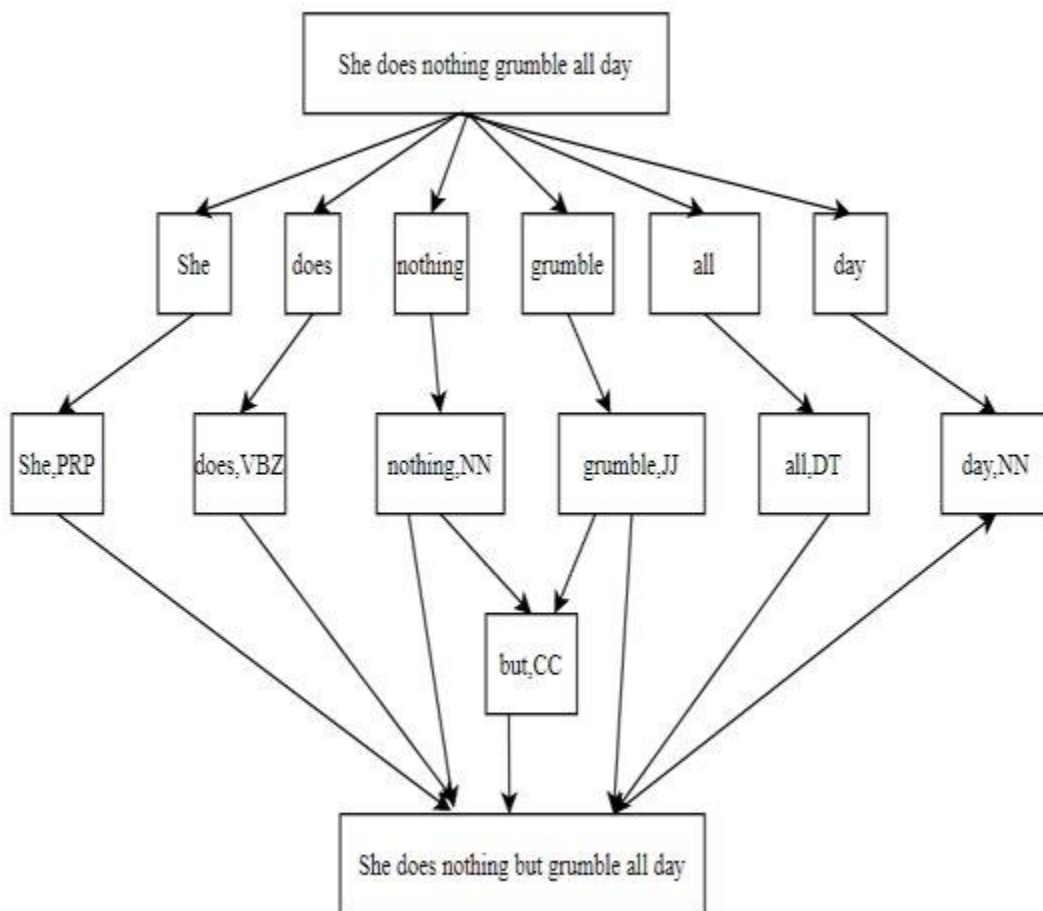
I/O: She does nothing grumble all day

O/P: She does nothing but grumble all day

**Example 2:**

I/P: I want to go to the party, I am so tired.

O/P: I want to go to the party, but I am so tired.



Graph 3.4

**3.6 Capitalization Error**

This method deals with the case sensitivity of words. Additionally, it returns a sentence with correct case sensitivity and the error count.

**Usage:**

```
from util.capitalization import check_capitalization
import nltk
text = input()
Print(check_capitalization([nltk.pos_tag(text)]))
```

**Example 1:**

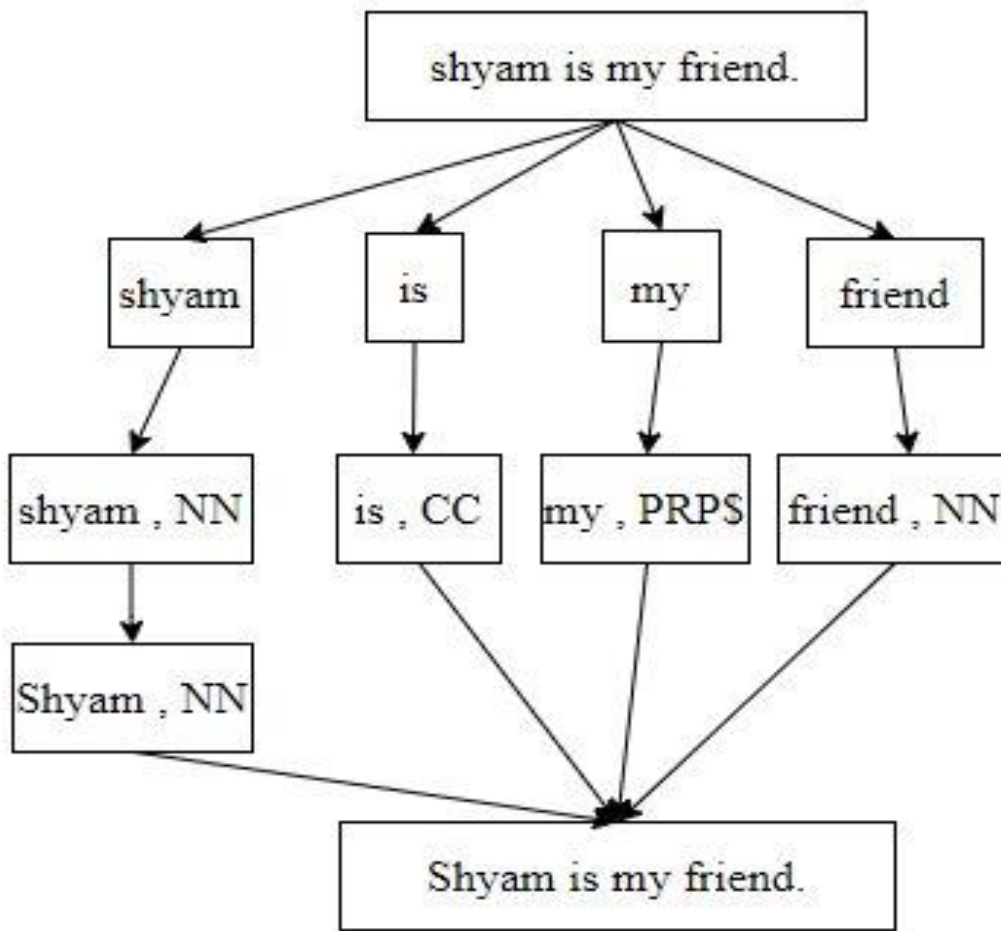
I/P: shyam is my friend.

O/P: Shyam is my friend.

**Example 2:**

I/P: i have to go to the mall.

O/P:I have to go to the mall.



Graph 3.5

### 3.7 Subject Verb Agreement Error

This method checks and modifies the subject verb agreement related errors provided in a sentence. Additionally, it returns a sentence which follows subject verb agreement and the error count.

**Usage:**

```

from util.subVerb import check_SubVerbAgreement
import nltk
text = input()
print(check_SubVerbAgreement ([nltk.pos_tag(text)]))
  
```

**Example 1:**

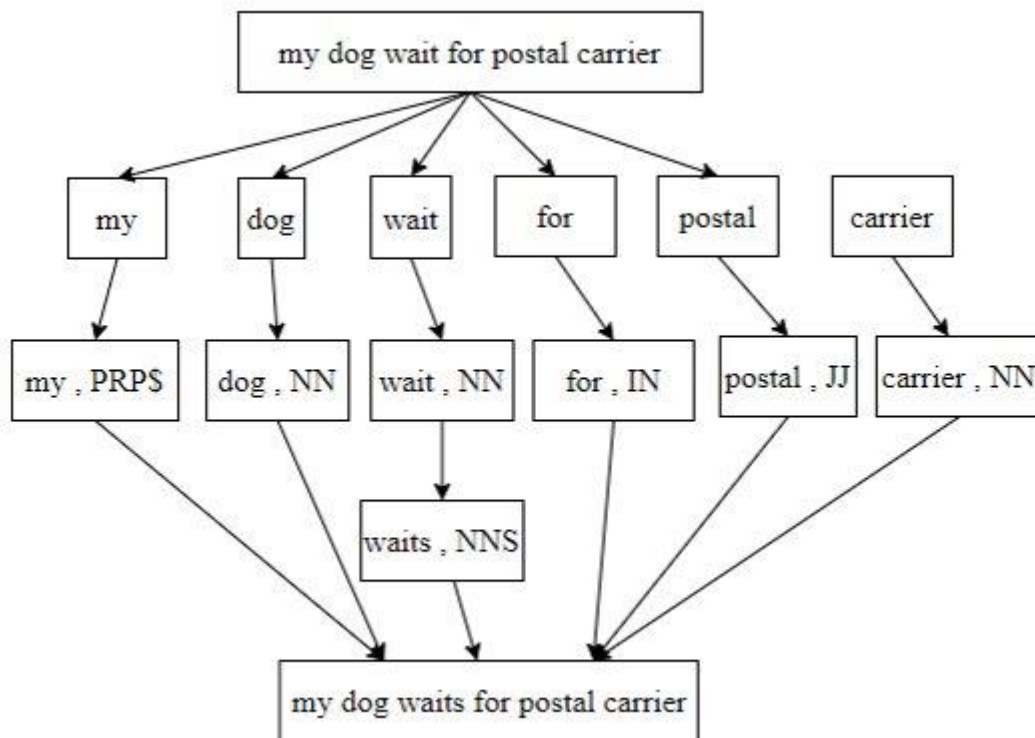
I/P: My dog wait for the postal carrier.

O/P: My dog waits for the postal carrier.

**Example 2:**

I/P: The group meet every day.

O/P: The group meets every day.



Graph 3.6

### 3.8 Reflex Pronoun Error

This method checks for the irregular usage of reflex pronouns (himself, herself etc.).

It eliminates, modifies the misused reflex pronouns and returns the error count along with the correct sentence.

**Usage:**

```
from util.reflexError import check_reflexError
```

```
Text = input ()  
print(check_reflexError(text))
```

**Example 1:**

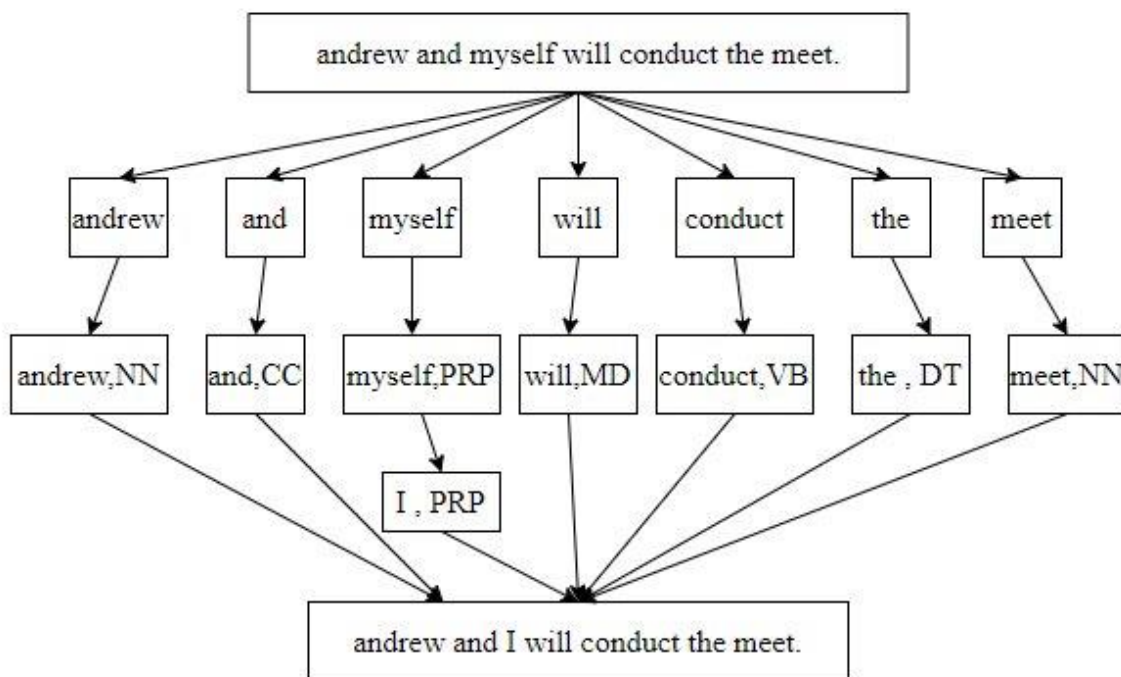
I/P: andrew and myself will conduct the meet.

O/P: andrew and I will conduct the meet.

**Example 2:**

I/P: Teachers and myself will be singing this song.

O/P: Teachers and I will be singing this song.



Graph 3.7

### 3.9 And Error

This method checks for the usage of ‘and’ conjunction used in the sentence provided. Additionally, it modifies the incorrect usage and return the error count.

**Usage:**

```
import nltk
from nltk import pos_tag,word_tokenize
from util.andError import and_check
print(and_check([nltk.pos_tag(word_tokenize(input ())))))
```

**Example 1:**

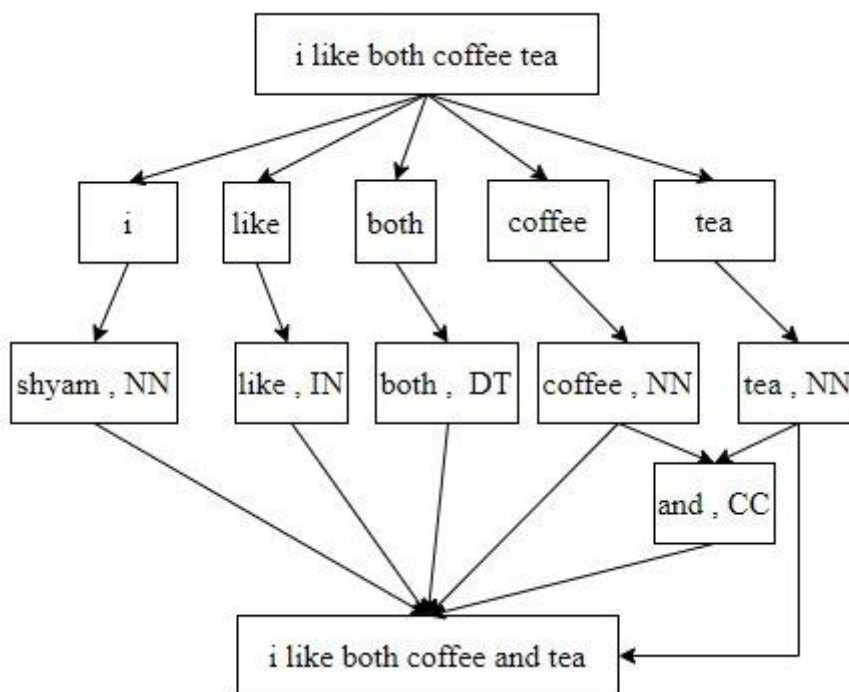
I/P: i like both coffee tea

O/P: i like both coffee and tea

**Example 2:**

I/P: She is loving caring.

O/P: She is loving and caring.



Graph 3.8



### 3.10 Spelling Error

This error checks the spelling errors in a sentence.

#### Usage:

```
from util.spell import spell_checker
import nltk
data = input()
print(spell_checker (data))
```

#### Example 1:

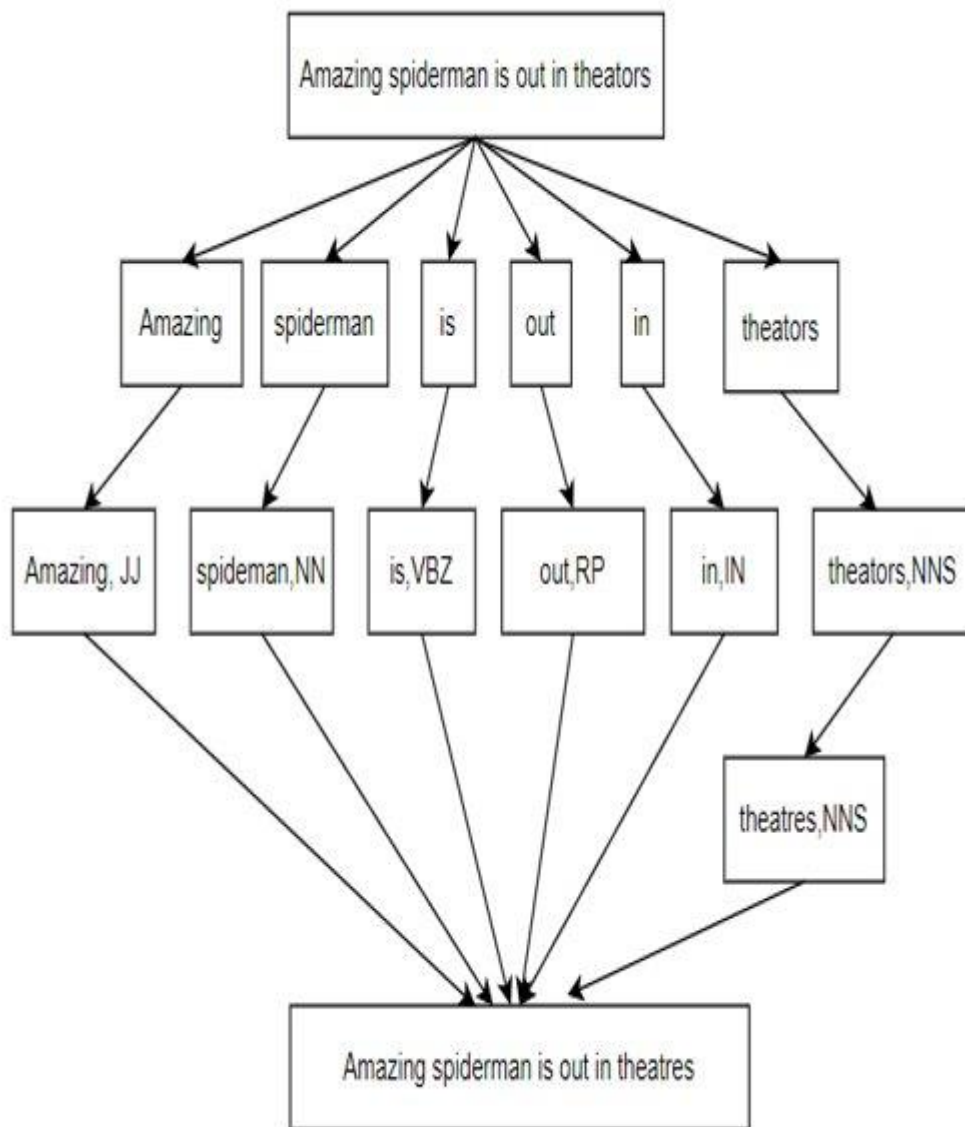
I/P: Amazing spiderman is out in theaters.

O/P: Amazing spiderman is out in theatres.

#### Example 2:

I/P: He thew out the garbge.

O/P: He threw out the garbage.



Graph 3.9

### 3.11 Apostrophe Error

This method deals with the incorrect placement of the apostrophe symbol in a sentence. It also returns the error count and returns the correct text which satisfies the use of apostrophe.

**Usage:**

```
from util.apostrophe import apostropheError
```

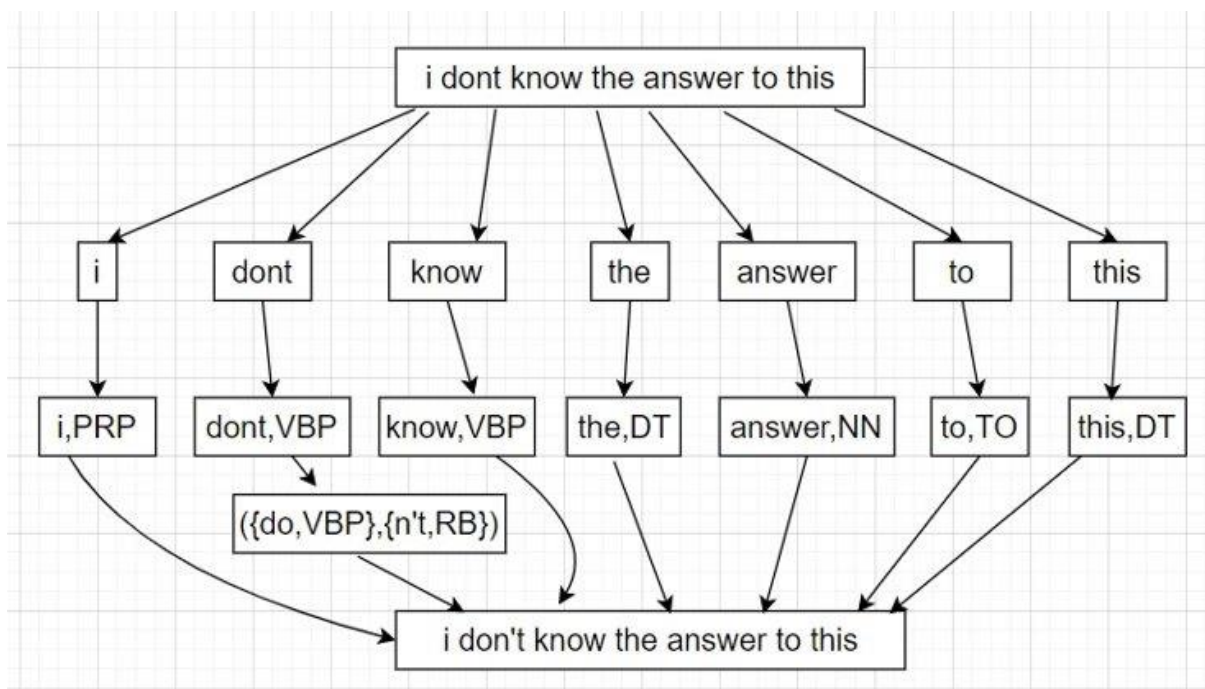
```
import nltk
text = input()
print(apostropheError([nltk.pos_tag(text)]))
```

**Example 1:**

I/P: mum and dads house  
 O/P: mum and dad's house

**Example 2:**

I/P: I dont like tea.  
 O/P: I don't like tea.



Graph 3.10

### 3.12 Pluralization Error

This error deals with the wrong usage of singular or plural words in a sentence.

**Usage:**

```
from util.pluralization import check_pluralization
import nltk
text = input()
print(check_pluralization ([nltk.pos_tag(text)])
```

**Example 1:**

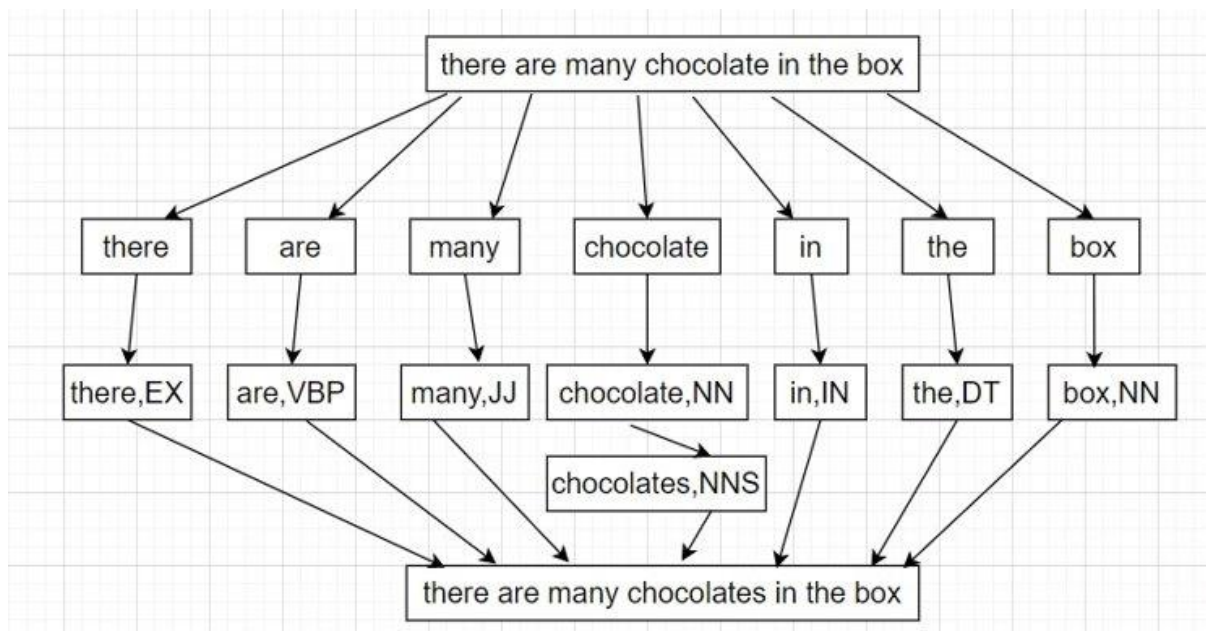
I/P: there are many chocolate in the box.

O/P: there are many chocolates in the box.

**Example 2:**

I/P: one kids is playing in the park.

O/P: one kid is playing in the park.



Graph 3.11

### 3.13 Although Though Error

This method checks for the correct grammatical usage Though/yet and although/comma in sentences.

**Usage:**

```

from util.although import althoughError
import nltk
text = input ()
print (althoughError ([nltk.pos_tag(text)]))

```

**Example 1:**

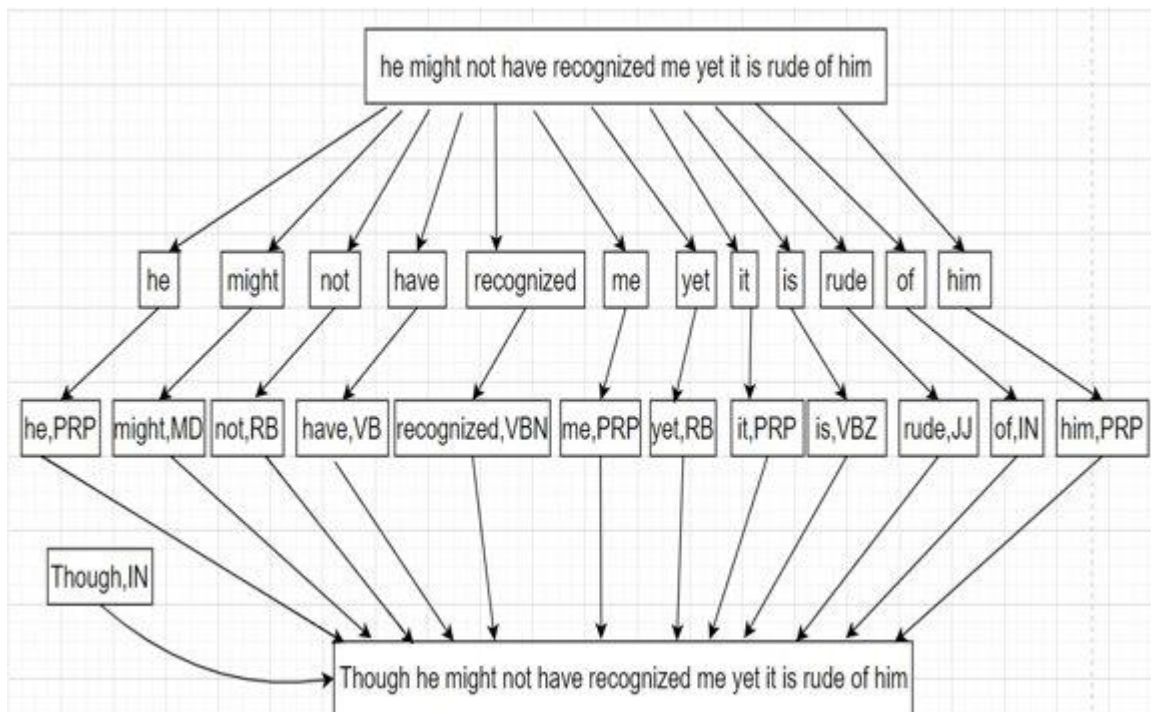
I/O: Though the attack happened, the army survived

O/P: Though the attack happened yet the army survived

**Example 2:**

I/P: Although we ran fast, we did not come first

O/P: Although we ran fast yet we did not come first



Graph 3.12

### 3.14 Either Neither Error

This method checks for the correct grammatical usage of Either/or and neither/nor in sentences.

#### Usage:

```
from util.eitherneither import eitherneitherError
import nltk
text = input ()
print (eitherneitherError ([nltk.pos_tag(text)]))
```

#### Example 1:

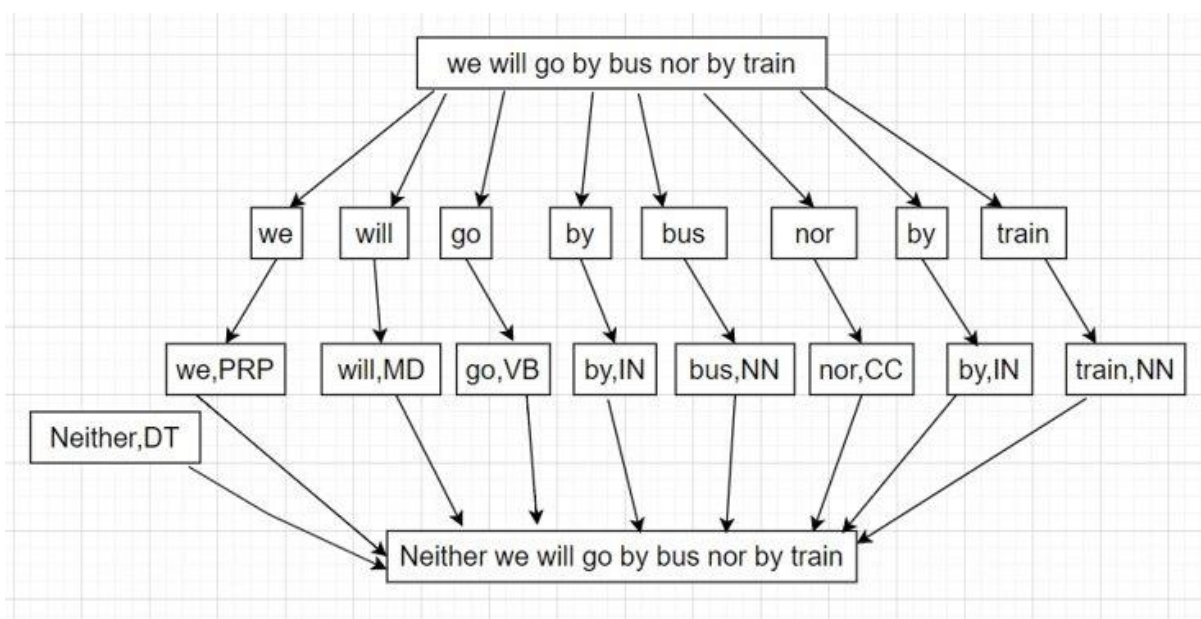
I/O: Either we dance else we sing.

O/P: Either we dance or we sing.

#### Example 2:

I/P: We will go by bus nor by train

O/P: Neither we will go by bus nor by train



Graph 3.13

### 3.15 Sentence Formatter

This module helps to format the given sentence into the readable format which doesn't have any indentation issues.

Extra spaces at the Head and Tail of the sentences and irregular spaces in between the sentence are eliminated.

#### Usage:

```
from util.sentFormatter import textFormatter  
Text = input ()  
print(textFormatter(text))
```

#### Example 1:

I/P: I 'm trying to make this task done on time.

O/P: I'm trying to make this task done on time.

#### Example 2:

I/P: He works hard.

O/P: He works hard.

# CHAPTER 4

## PERFORMANCE ANALYSIS

The style and grammar checker in its current state consists of 14 grammar rules, spell checker and sentence formatter.

INPUT	I ate a apple
WORD TOKENIZATION	['I', 'ate', 'a', 'apple']
PART OF SPEECH TAGGING	[('I', 'PRP'), ('ate', 'VBP'), ('a', 'DT'), ('apple', 'NN')]
CHUNKING PHRASES	[('I', 'PRP'), ('ate', 'VBP'), ('a', 'DT'), ('apple', 'NN')]
RULE MATCHING	[('I', 'PRP'), ('ate', 'VBP'), ('an', 'DT'), ('apple', 'NN')]
OUTPUT	I ate an apple

Table 4.1

### 4.1 Word Tokenization

Tokenization is the process of breaking down a big amount of text into smaller pieces called tokens. These tokens are extremely important for pattern recognition and are used as a starting point for stemming and lemmatization. Tokenization can also be used to replace sensitive data pieces with non-sensitive ones.

```
from nltk.tokenize import word_tokenize
s = "I ate a apple"
print(word_tokenize(s))
```

Figure 4.1

```
['I', 'ate', 'a', 'apple']
```

Figure 4.2



## 4.2 POS Tagging

Part-of-speech tagging, sometimes known as POS-tagging or just tagging, is the act of classifying words into their parts of speech and labelling them accordingly. Word classes or lexical categories are other terms for parts of speech. A tag set is a collection of tags that are used for a specific job.

```
from nltk import pos_tag, word_tokenize
print(pos_tag(word_tokenize("I ate a apple|")))
```

Figure 4.3

```
[('I', 'PRP'), ('ate', 'VBP'), ('a', 'DT'), ('apple', 'NN')]
```

Figure 4.4

## 4.3 Chunking Phrases

Chunking is a method of grouping similar tokens into a single chunk. The outcome will be determined by the grammar that has been chosen. NLTK is also used to categorise patterns and analyse text corpora using Chunking.

## 4.4 Rule Matching

```
from grammarChecker import checker
enter sentence : I ate a apple.
spell : I ate a apple.
cap : I ate a apple.
apostrophe : I ate a apple.
pluralization : I ate a apple.
article : I ate an apple.
sub-verb : I ate an apple.
becoz : I ate an apple.
but : I ate an apple.
either-neither : I ate an apple.
reflex pronoun : I ate an apple.
and : I ate an apple.
although-though : I ate an apple.
tense : I ate an apple.
errors : 1
modified text :
    I ate an apple.
```

Figure 4.5

#### 4.4.1 Article Error

```

from nltk.tokenize import word_tokenize
from pattern.en import referenced
from grammarChecker.util import *

def check_articleError(text):
    nlp = tagger(text)
    error_count = 0
    correct_text = ""
    for sent in nlp:
        for index in range(len(sent)):
            if sent[index][0] in ['a', 'an']:
                # to check whether the (i)th word is in text file or tag of (i+1)th word is plural noun(eg cats,
                # pencils) or proper noun,plural(eg Indians,Americans)
                if sent[index + 1][1] in ["NNS", "NNPS"]:
                    error_count += 1
                # to check whether the tag of(i+1)th word is adjective(large, fast, honest) and the tag of (i+2)th
                # word is plural noun
                elif index < len(sent)-2 and sent[index+1][1] in ["JJ", "JJR"] and sent[index+2][1] in ['NNP', 'NN']:
                    """
                    referenced provides appropriate article before the word
                    eg:
                    I/P: referenced(hour)
                    O/P: an hour
                    """
                    # to check whether (i)th word equals 'a' and reference of next word equals 'an'+(i+1)th word
                    if sent[index][0] == 'a' and referenced(sent[index + 1][0]) == ('an ' + sent[index + 1][0]):
                        correct_text += 'an'
                        error_count += 1
                    # to check whether (i)th word equals 'an' and reference of next word equals 'a'+(i+1)th word
                    elif sent[index][0] == 'an' and referenced(sent[index + 1][0]) == ('a ' + sent[index + 1][0]):
                        correct_text += 'a'
                        error_count += 1

```

Figure 4.6

```

        else:
            correct_text += sent[index][0]
        elif sent[index + 1][1] not in ["NNP", "NN"]:
            error_count += 1
            # to check whether (i)th word equals 'a' and reference of next word equals 'an'+(i+1)th word
            elif sent[index][0] == 'a' and referenced(sent[index + 1][0]) == ('an ' + sent[index + 1][0]):
                correct_text += 'an'
                error_count += 1
            # to check whether (i)th word equals 'an' and reference of next word equals 'a'+(i+1)th word
            elif sent[index][0] == 'an' and referenced(sent[index + 1][0]) == ('a ' + sent[index + 1][0]):
                correct_text += 'a'
                error_count += 1
        else:
            correct_text += sent[index][0]
        correct_text += " "
    else:
        correct_text += sent[index][0]
        correct_text += " "
    return error_count, textFormatter(correct_text)

```

Figure 4.7

## 4.4.2 Because Error

```
from grammarChecker.util import *

def check_becauseError(text):
    nlp = tagger(text)
    error_count = 0
    correct_text = ""
    for sent in nlp:
        for index in range(len(sent)):
            if sent[index][0] == 'because':
                # if condition for checking if the token encountered is punctuation
                if sent[index+1][1] == 'PUNCT' or index == len(sent)-1:
                    error_count += 1
                    correct_text += "."
                    break
                # if condition for checking if the token encountered is preposition
                if sent[index+1][1] == 'IN':
                    if index == len(sent)-2:
                        error_count += 1
                    # if condition for checking if the token encountered is not singular noun,plural noun,
                    # proper singular noun,proper plural noun,personal pronoun,possessive pronoun,determiner
                    elif sent[index + 2][1] not in ['NN', 'NNS', 'NNP', 'NNPS', 'PRP', 'PRP$', 'DT']:
                        error_count += 1
                        correct_text += "."
                        break
                    else:
                        correct_text += sent[index][0]
                        correct_text += " "
                # if condition for checking if the token encountered is singular noun,plural noun,proper singular
                # noun,proper plural noun,personal pronoun,possessive pronoun
                elif sent[index + 1][1] in ['NN', 'NNS', 'NNP', 'NNPS', 'PRP', 'PRP$']:
                    if index == len(sent)-2:
                        error_count += 1
                    flag = 0
```

Figure 4.8

```

for j in range(index + 2, len(sent)):
    # if condition for checking if the token encountered is verb, modal
    if sent[j][1] in ['VB', 'VBP', 'VBZ', 'VBG', 'VBN', 'VBD', 'MD']:
        flag += 1
        break
if flag == 0:
    error_count += 1
    correct_text += "."
    break
else:
    correct_text += sent[index][0]
    correct_text += " "
else:
    correct_text += sent[index][0]
    correct_text += " "
else:
    correct_text += sent[index][0]
    correct_text += " "

return error_count, textFormatter(correct_text)

```

Figure 4.9

### 4.4.3 Tense Error

```

from pattern.en import conjugate
from grammarChecker.util import *

def check_TenseError(text):
    nlp = tagger(text)
    error_count = 0
    correct_text = ""
    try:
        conjugate('go', 'part')
    except:
        pass
    for sent in nlp:
        dic = {"VB": 0, "VBP": 0, "VBD": 0, "VBZ": 0, "VBN": 0, "VBG": 0, "MD": 0}
        for index in range(len(sent)):
            try:
                # if condition for checking if the token encountered is modal(could,will)
                if sent[index][1] == "MD":
                    if index < len(sent) - 1 and sent[index + 1][1] in ["VBZ", 'VBP', 'VBN', 'VBG', 'VBD']:
                        # change verb to its base form
                        word_lemma = lemma(wordnet_tagged[index + 1])
                        if sent[index + 1][0] != word_lemma:
                            sent[index + 1][0] = word_lemma
                            error_count += 1
                        dic["VB"] += 1
                        dic["MD"] += 1
                    # if condition for checking if the word encountered is "be" and token encountered is verb
                    if index < len(sent) - 2 and sent[index + 1][0] == "be" and sent[index + 2][1] in ["VB", "VBP", "VBD", "VBZ"]:
                        # change verb to its participle
                        participle = conjugate(sent[index + 2][0], "part")
                        if participle != sent[index + 2][0]:
                            correct_word = (participle, sent[index + 2][1])
                            sent[index + 2] = correct_word
                            error_count += 1
                        dic["VBG"] += 1
            else:
                correct_text += sent[index][0] + " "
            elif sent[index][1] in ["VB", "VBP", "VBZ"]:
                # if condition for checking if the token encountered is verb and the word encountered is "have","has"
                if index < (len(sent) - 1) and sent[index][0] in ["has", "have"] and sent[index + 1][1] in ["VBZ", "VBD", "VB", "VBP", "VBG", 'VBN']:
                    # change verb to its past participle
                    participle = conjugate(sent[index + 1][0], "ppart")
                    if participle != sent[index + 1][0]:
                        correct_word = (participle, sent[index + 1][0])
                        sent[index + 1] = correct_word
                        error_count += 1
                    dic["VBN"] += 1
                    correct_text += sent[index][0] + " "
                # if condition for checking if the token encountered is verb and the word encountered is "is","am","are"
                elif index < (len(sent) - 1) and (sent[index][0] in ["is", "am", "are"]) and (sent[index + 1][1] in ["VBZ", "VBD", "VB", "VBP"]):
                    # change verb to its past participle
                    participle = conjugate(sent[index + 1][0], "part")
                    if participle != sent[index + 1][0]:
                        correct_word = (participle, sent[index + 1][0])
                        sent[index + 1] = correct_word
                        error_count += 1
                    dic["VBG"] += 1
                    correct_text += sent[index][0] + " "
            else:
                correct_text += sent[index][0] + " "
            else:
                correct_text += sent[index][0] + " "
        except:
            correct_text += sent[index][0] + " "
    return error_count, textFormatter(correct_text)

```

Figure 4.10

```

else:
    correct_text += sent[index][0] + " "
elif sent[index][1] in ["VB", "VBP", "VBZ"]:
    # if condition for checking if the token encountered is verb and the word encountered is "have","has"
    if index < (len(sent) - 1) and sent[index][0] in ["has", "have"] and sent[index + 1][1] in ["VBZ", "VBD", "VB", "VBP", "VBG", 'VBN']:
        # change verb to its past participle
        participle = conjugate(sent[index + 1][0], "ppart")
        if participle != sent[index + 1][0]:
            correct_word = (participle, sent[index + 1][0])
            sent[index + 1] = correct_word
            error_count += 1
        dic["VBN"] += 1
        correct_text += sent[index][0] + " "
    # if condition for checking if the token encountered is verb and the word encountered is "is","am","are"
    elif index < (len(sent) - 1) and (sent[index][0] in ["is", "am", "are"]) and (sent[index + 1][1] in ["VBZ", "VBD", "VB", "VBP"]):
        # change verb to its past participle
        participle = conjugate(sent[index + 1][0], "part")
        if participle != sent[index + 1][0]:
            correct_word = (participle, sent[index + 1][0])
            sent[index + 1] = correct_word
            error_count += 1
        dic["VBG"] += 1
        correct_text += sent[index][0] + " "
    else:
        correct_text += sent[index][0] + " "
else:
    correct_text += sent[index][0] + " "
except:
    correct_text += sent[index][0] + " "
return error_count, textFormatter(correct_text)

```

Figure 4.11

## 4.4.4 But Error

```
from grammarChecker.util import *

def check_butError(text):
    nlp = tagger(text)
    error_count = 0
    correct_text = ''
    for sent in nlp:
        for index in range(len(sent)-1):
            correct_text += sent[index][0] + ' '
            # if the token encountered is ','
            if sent[index][1] in [',']:
                if sent[index-1][1] in ['NN', 'NNP', 'NNS', 'NNPS'] and sent[index+1][1] in ['NN', 'NNS', 'NNP', 'NNPS']:
                    correct_text += ' '
                else:
                    correct_text += 'but '
                    error_count += 1
            # if the token encountered is noun and the next token encountered is also noun, determinat
            elif sent[index][1] in ['nothing', 'everyone', 'anything', 'anyone'] and sent[index+1][1] in ['NN', 'NNS', 'NNP', 'NNPS', 'DT']:
                correct_text += 'but '
                error_count += 1
        correct_text += sent[-1][0]
    return error_count, textFormatter(correct_text)
```

Figure 4.12

# CHAPTER 5

## CONCLUSIONS

### 5.1 Conclusion

A style and grammar checker has been built and tested on real-world errors in this paper. Several of these common errors have been identified using a rule set. By adding new error rules to an XML file, new error rules can be readily added to the checker. The rule can include a helpful discussion of the problem as well as correct and incorrect example sentences for each error. Even faults that the rule system cannot express can be readily added without affecting the checker's source code: a Python file implementing the rule just has to be placed in a dedicated subdirectory to be used automatically.

It's simple to create graphical user interfaces that communicate with the backend because the backend and frontend are clearly separated. The checker has been integrated into KWord and includes a minimal online interface with a limited number of options. A background checker server process is fast enough for interactive use of the checker in KWord, allowing errors to be highlighted while typing. Each error rule can be enabled and disabled independently via a setup dialogue.

The checker's backend and command line tool are simple to set up, requiring only Python and no additional modules. A webserver is also required for the CGI frontend. The KWord integration currently necessitates the installation of KWord from source code in order to apply the necessary changes.

The rule-based method avoids the need for a complex parser and allows for a step-by-step development strategy, which should make adapting the checker to other languages relatively simple. A tagged corpus can be used to train the part-of-speech tagger, and new error criteria can be developed as needed.



## 5.2 Future Scope

Future Perspectives We compared and analysed various grammar checker techniques in this paper, but because each technique has its own set of advantages and disadvantages, it is necessary to develop a technique that can overcome all of these drawbacks with a proper grammar checker while also increasing sentence accuracy.

## 5.3 Applications

The grammar checker is a free tool that may be used as a standalone style and grammar checker or integrated into a word processor. The style and grammar checker in this report analyses a text and generates a list of possible problems. To find difficulties, each word in the text is assigned a part-of-speech tag, and each sentence is broken down into chunks, such as noun phrases. The text is then compared to all of the error rules that the checker has previously specified. If a rule matches, there should be an error in the text where the match occurs. Word patterns, part-of-speech tags, and chunks are all defined as faults in the rules. Each rule also includes an explanation of the error for the user to see.

Major applications:

- 1) Improve Your Writing in Seconds. Improve your grammar, spelling, and clarity in real time, as well as your fluency, style, and tone.
- 2) Stay in Control.

## REFERENCES

- 1) Token · spaCy API Documentation
- 2) style\_and\_grammar\_checker.pdf (danielnaber.de)
- 3) <https://www.nltk.org/>
- 4) <http://digiasset.org/html/pattern-en.html>
- 5) <https://www.nltk.org/api/nltk.tag.html>