Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

STATISTICS AND PROBABILITY MIDTERM REPORT

STATISTICS AND PROBABILITY MIDTERM ESSAY

Instructor: **Mr. Nguyen Quoc Binh**
Student: **Le Gia Phu – 520H0401**
Class **: 20H50202**
Year **: 24**

HO CHI MINH CITY, 2022

Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

STATISTICS AND PROBABILITY MIDTERM REPORT

STATISTICS AND PROBABILITY

MIDTERM ESSAY

Instructor: **Mr. Nguyen Quoc Binh**
Student: **Lê Gia Phu – 520H0401**
Class **: 20H50202**
Year **: 24**

HO CHI MINH CITY, 2022

ACKNOWLEDGEMENT

The first sincere thanks I want to give to Mr. Nguyen Quoc Binh, who enthusiastically taught and worked tirelessly to give me enough tools and skills to complete this report. She played an important role in improving my mathematical logic and knowledge. The second thanks I would like to give to the teachers of the Department of Information Technology of Ton Duc Thang University for giving me the opportunity to do this report, because it is not only a report but also a very important experience for me in the next 2 years.

Because the impact of the epidemic is too great, my report will have some errors, I am very open to receiving feedback from teachers so that I can improve my report writing skills.

Finally, I wish you good health and success in your noble career.

Ho Chi Minh city, 15th April, 2022

Author

(Sign and write full name)

Le Gia Phu

THIS PROJECT WAS COMPLETED AT
TON DUC THANG UNIVERSIY

I fully declare that this is my own project and is guided by Mr. Nguyen Quoc Binh. The research contents and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

Besides that, the project also uses a number of comments, assessments as well as data from other authors, other agencies and organizations, with citations and source annotations.

Should any frauds were found, I will take full responsibility for the content of my report. Ton Duc Thang University is not related to copyright and copyright violations caused by me during the implementation process (if any).

Ho Chi Minh city, 15th April, 2022

Author

(Sign and write full name)

Lê Gia Phu

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh city, April, 2022

(Sign and write full name)

Evaluation section for grading instructor

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh city, April, 2022

(Sign and write full name)

SUMMARY

This is a report on Statistic and Probability by Faculty of Information Technology of Ton Duc Thang University. The report has 4 CHAPTERS:

CHAPTER 1: INTRODCUTION

CHAPTER 2: MONOALPHABETIC SUBTITUTION CIPHER

CHAPTER 3: FREQUNCY ANALYSIS

CHAPTER 4: EXPERIMENT

I spent my own time research on the Internet and reading different materials from Vietnamese to English to find out the answers, and by the help of Mr. Nguyen Quoc Binh. This report was finish at 8:30AM on 15th April, 2022. If it weren't for Mr. Binh lessons, I could not perfectly finish this report. Sometimes this article of mine has many errors, I am very open to receive the constructive contributions of teachers and will use it as a lesson for the final articles.

INDEX

CHAPTER 1 – INTRODUCTION

1. Concepts of encryption and decryption:

a. Encryption:

Encryption is a way of scrambling data so that only authorized parties can understand the information. In technical terms, it is the process of converting human-readable plaintext to incomprehensible text, also known as ***ciphertext***. In simpler terms, encryption takes readable data and alters it so that it appears random. Encryption requires the use of a ***cryptographic key***: a set of mathematical values that both the sender and the recipient of an encrypted message agree on.

Although encrypted data appears random, encryption proceeds in a logical, predictable way, allowing a party that receives the encrypted data and possesses the right key to decrypt the data, turning it back into plaintext. Truly secure encryption will use keys complex enough that a third party is highly unlikely to decrypt or break the ciphertext by brute force — in other words, by guessing the key. Data can be encrypted "at rest," when it is stored, or "in transit," while it is being transmitted somewhere else.

Encryption used something called a cryptographic key and/or an encryption algorithm. **A cryptographic key** is a string of characters used within an encryption algorithm for altering data so that it appears random. Like a physical key, it locks (encrypts) data so that only someone with the right key can unlock (decrypt) it. **An encryption algorithm** is the method used to transform data into ciphertext. An algorithm will use the encryption key in order to alter the data in a predictable way, so that even though the encrypted data will appear random, it can be turned back into plaintext by using the decryption key.

Encryption provides *PRIVACY (ensures that no one can read communications or data at rest except the intended recipient or the rightful data owner),* SECURITY *(prevent data breaches, whether the data is in transit or at rest),* DATA INTEGRITY *(helps prevent malicious behavior such as on-path attacks),* AUTHENTICATION *(allows users of the website to be sure that they are connected to the real website)*

The primary purpose of encryption is to protect the confidentiality of digital data stored on computer systems or transmitted over the internet or any other computer network. In addition to security, the adoption of encryption is often driven by the need to meet compliance regulations. A number of organizations and standards bodies either recommend or require sensitive data to be encrypted in order to prevent unauthorized third parties or threat actors from accessing the data. For example, the Payment Card Industry Data Security Standard (PCI DSS) requires merchants to encrypt customers' payment card data when it is both stored at rest and transmitted across public networks.

A disadvantage of encryption, while encryption is designed to keep unauthorized entities from being able to understand the data they have acquired, in some situations, encryption can keep the data's owner from being able to access the data as well. **Key management is one of the biggest challenges of building an enterprise encryption strategy** because the keys to decrypt the cipher text have to be living somewhere in the environment, and attackers often have a pretty good idea of where to look.

Having a key management system in place isn't enough. Administrators must come up with a comprehensive plan for protecting the key management system. Typically, this means backing it up separately from everything else and storing those

backups in a way that makes it easy to retrieve the keys in the event of a large-scale disaster.

b. Decryption:

Decryption is a process that transforms encrypted information into its original format. The process of encryption transforms information from its original format — called plaintext — into an unreadable format — called ciphertext — while it is being shared or transmitted. To do this, parties to a private conversation use an encryption scheme, called an algorithm, and the keys to encrypt and decrypt messages. (Encrypted messages are called ciphertext, as algorithms are also called ciphers.) Message recipients decrypt the information back into its original, readable format. Future messages when passed through the system are encrypted, and vice versa.

Although encryption protects the data, recipients must have the right Decryption or decoding tools to access the original details. What Decryption does is unencrypt the data, which can be done manually, automatically, using the best Decryption software, unique keys, passwords, or codes. This translates unreadable or indecipherable data into original text files, e-mail messages, images, user data, and directories that users and computer systems can read and interpret.

Decryption exists something called a brute force attack. A brute force attack is when an attacker who does not know the decryption key attempts to determine the key by making millions or billions of guesses. Brute force attacks are much faster with modern computers, which is why encryption has to be extremely strong and complex. Most modern encryption methods, coupled with high-quality passwords, are resistant to brute force attacks, although they may become vulnerable to such

attacks in the future as computers become more and more powerful. Weak passwords are still susceptible to brute force attacks.

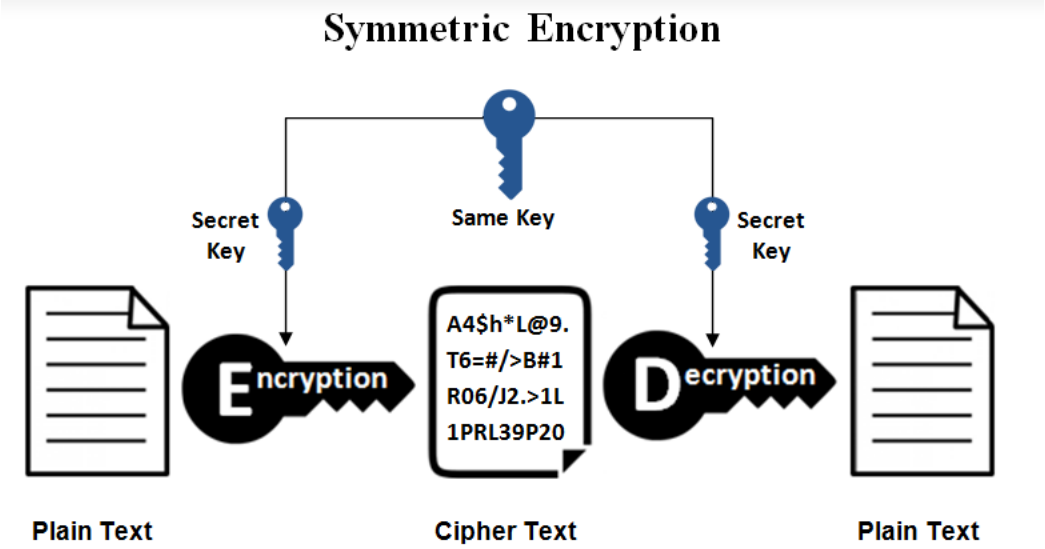
2. Symmetric and Asymmetric cryptosystem:

At the beginning of the encryption process, the sender must decide what cipher will best disguise the meaning of the message and what variable to use as a key to make the encoded message unique. The most widely used types of ciphers fall into two categories: ***symmetric and asymmetric***.

The fundamental difference between them is that the ***symmetric cryptosystem only involves one key*** that the system uses for encryption and decryption. ***The asymmetric cryptosystem uses two keys (private and public)*** for encryption and decryption.

a. Symmetric cryptosystem:

Symmetric ciphers, also referred to as *secret key encryption*, use a single key. The key is sometimes referred to as a *shared secret* because the sender or computing system doing the encryption must share the secret key with all entities authorized to decrypt the message. Symmetric key encryption is usually much faster than asymmetric encryption. The most widely used symmetric key cipher is the Advanced Encryption Standard (AES), which was designed to protect government-classified information.



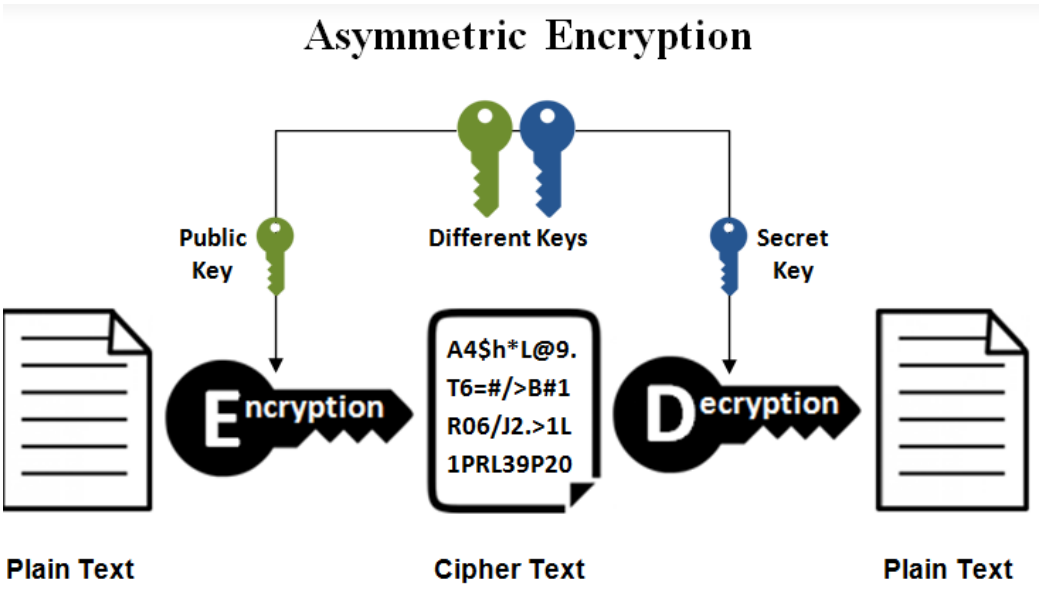
Since the symmetric cryptosystem uses the same key for encryption and decryption, the sender and receiver need to know the key before exchanging messages. For this, the system employs key exchange protocols (like the Diffie-Hellman Protocol) to ensure that the sender and receiver agree on the secret key before starting the communication.

For the encryption algorithms, the symmetric cryptosystem uses AES-128, AES-192, and AES-256. The encryption process is generally faster in symmetric cryptosystems due to the smaller key lengths. The drawback of this system is that the senders and receivers need to exchange keys before decrypting the message. If the keys are not changed regularly, the system will be prone to attacks since an attacker can use the leaked key to disrupt the communication. The symmetric cryptosystem uses MACs to provide integrity and authentication.

b. Asymmetric cryptosystem:

Asymmetric ciphers, also known as *public key encryption*, use two different -- but logically linked -- keys. This type of cryptography often uses prime numbers to create keys since it is computationally difficult to factor large prime numbers and reverse-

engineer the encryption. The Rivest-Shamir-Adleman (RSA) encryption algorithm is currently the most widely used public key algorithm. With RSA, the public or the private key can be used to encrypt a message; whichever key is not used for encryption becomes the decryption key.



The asymmetric cryptosystem overcomes the symmetric cryptosystem challenges by eliminating the need to pre-sharing the key before communication. The asymmetric cryptosystem used two keys (private and public) that are mathematically related to each other. The strength of security lies in these keys' properties since it is computationally infeasible to calculate one key using the other. Each sender and receiver will have their private-public key pair in this system. Suppose A wants to send a message to B, A will need to use B's public key to encrypt the message, and B will decrypt the message using their private key.

For the encryption algorithms, the asymmetric cryptosystem uses El Gamal Encryption or RSA Encryption schemes. The encryption process is slower than symmetric encryption due to larger key sizes. The asymmetric cryptosystem is more

secure since the private and public key pairs are less likely to get leaked and predicted. The asymmetric cryptosystem uses RSA signatures to provide integrity and authentication.

c. Symmetric vs Asymmetric cryptosystem:

Symmetric algorithms encrypt and decrypt with the same key. Main advantages of symmetric algorithms are its security and performance. Asymmetric algorithms encrypt and decrypt with different keys. Data is encrypted with a public key and decrypted with a private key. Asymmetric algorithms (also known as public-key algorithms) need at least a 3,000-bit key to achieve the same level of security of a 128-bit symmetric algorithm. Asymmetric algorithms are also incredibly slow and it is impractical to use them to encrypt large amounts of data. Symmetric algorithms are about 1,000 times faster than asymmetric ones.

Today, many cryptographic processes use a symmetric algorithm to encrypt data and an asymmetric algorithm to securely exchange the secret key.

CHAPTER 2 – MONOALPHABETIC SUBTITUTION CIPHER

1. Definition of monoalphabetic subtitution cipher:

A cipher where each symbol is replaced by another symbol, where the replacement does not vary, is called a monoalphabetic substitution cipher. A good monoalphabetic substitution algorithm matches the plain alphabet with a permutation of itself, a permutation determined by a key. *(Notice that this is a substitution algorithm, not a transposition algorithm, because it replaces the plain alphabet, not the encrypted message, with a permutation)* If the alphabet consists of 26 symbols, there are 26! ≈ 4 x 10^{26} monoalphabetic substitution ciphers for the alphabet. There are many different monoalphabetic substitution ciphers, in fact infinitely many, as each letter can be encrypted to any symbol, not just another letter. This type of cipher is a form of symmetric encryption as the same key can be used to both ***encrypt*** and ***decrypt*** a message.

Plain alphabet	ABCDEFGHIJKLMNOPQRSTUVWXYZ␣
Cipher alphabet	LAND␣TREFGHIJKMOPQSUVWXYZBC

In spite of this impressive number, these ciphers are not secure.

2. Some problems and constraints with monoalphabetic subtitution cipher:

Monoalphabetic substitution codes were extensively used throughout the first millennium A.D. and for a long time they were more than adequate for the purposes for which they were needed . By today's standards they are very weak, and incredibly easy to break, but they were a very important step in developing cryptography. They fell victims to the development of statistics and concepts of probability.

Simple Substitution Cipher is a considerable improvement over the Caesar Cipher. The possible number of keys is large (26!) and even the modern computing systems are not yet powerful enough to comfortably launch a brute force attack to break

the system. However, the Simple Substitution Cipher has a simple design and it is prone to design flaws, say choosing obvious permutation, this cryptosystem can be easily broken.

Breaking such a code is easy and does not require checking many alternatives. It is based on the fact that in any language, the various letters appear in texts with different probabilities (we say that the distribution of letters is nonuniform). Some letters are common while others are rare. The most common letters in English, for example, are E, T, and A (if a blank space is also included in the alphabet, then it is the most common), and the least common are Z and Q. If a mono alphabetic substitution code replaces E with, say, D, then D should be the most common letter in the ciphertext.

3. Method of monoalphabetic subtitution cipher:

As I said earlier, monoalphabetic subtitution cipher is a cipher where each symbol is replaced by another symbol (but these symbols are unique, they do not repeat). It is better than a Ceasar Cipher (this cipher shift all the letters of the alphabet by 3), by relying on permutation, 26! permutation to be percise which means 26! possible keys for to encryption.

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Alphabet	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A

Because monoalphabetic subtitution cipher has a simple way to imply and there are 26! possible keys that can be created, so every values that we put in can be called a monoalphabetic subtitution cipher. But there are **2 possible ways** to create a monoalphabetic cipher:

- The first one is random keyword subtitutions. Giving the American alphabet with 26 letters, the first letter A can be subtituted with any of the 25 other

letters, this process continued to the last letter of the alphabet, the letter Z. This will generate a random keyword every time.

- The second one, is via mixing keyword. Giving a keyword (e.g. LAND AND TREE), we then stripped any of the repeated letters in the keyword (e.g. LANDTRE), we then applied it to the alphabbet, replace after the keyword with the remaining letters. Noted that you can mixed keyword with a various of ways, for example, you can mixed it like the example above or you can mixed it via columns or via the length of the keyword it self.

4. Examples of monoalphabetic subtitution cipher:

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Alphabet	K	D	G	F	N	S	L	V	B	W	A	H	E	X	J	M	Q	C	P	Z	R	T	Y	I	U	O

5. Personal comments about monoalphabetic subtitution cipher:

Monoalphabettic subtitution cipher is the most simpliest cipher to applied. It was created as a successor to the Ceasar Cipher, because Ceasar Cipher was a cipher that shift only 3 letters which is not secure, the Monoalphabettic subtitution cipher relies on its 26! keyword permutaion for a more secure encryption, disabling anyone from trying to break the cipher by brute force. It is a based that will lead to the creation of other complex ciphers.

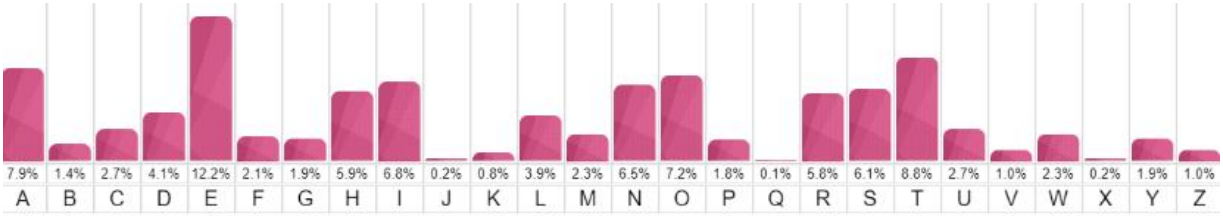
Although the monoalphabetic subtitution cipher was created for a more secure way to encrypt data while also eliminating brute force attack that can break a cipher but it still fall for modern day cipher breaker method, such as one method called frequency analysis, which can based on the letter frequency and brute force break the cipher. Still the monoalphabetic subtitution cipher is one of the more well known and adequte cipher for the purpose for which they were needed.

CHAPTER 3 – FREQUENCY ANALYSIS

1. Definition of frequency analysis:

In cryptography, frequency analysis is the study of the **frequency of letters** or groups of letters in a ciphertext. The method is used as an aid to breaking **substitution ciphers** *(e.g. mono-alphabetic substitution cipher, Caesar shift cipher, Vatsyayana cipher).*

Frequency anaelysis consists of **counting the occurrence of each letter** in a text. Frequency analysis is based on the fact that, in any given piece of text, certain letters and combinations of letters occur with varying frequencies. For instance, given a section of English language, letters **E, T, A and O** are the most common, while letters Z, Q and X are not as frequently used. We can assume that most samples of text written in English would have a similar distribution of letters.



This is possible because of letters frequency distribution. There are a lot of possible cipher keys (26! actually) so in theory a Simple Substitution cipher would be difficult to crack. Even throwing aside modern computing power though, its actually very doable to decipher a message of 100 or more characters by hand, provided you know the original language and a little about frequency distribution. Most languages have certain tells. Some letters and some sequences of letters appear more often than others. In the average English text (over say, 100 characters), ‘e’ will generally appear most frequently, followed by ‘t’ and ‘a’. The letters ‘x’, ‘q’ and ‘z’ appear the least.

However this is only true if the sample of text is long enough. A very short text may lead to a significantly different distribution. When trying to decrypt a cipher text

based on a substitution cipher, we can use a frequency analysis to help identify the most recurring letters in a cipher text and hence make **hypothesis** of what these letters have been encoded as (e.g. E, T, A, O, etc). This will help us decrypt some of the letters in the text. We can then recognise **patterns/words** in the partly decoded text to identify more substitutions.

2. Some problems and constraints with frequency analysis:

The most common problem with frequency analysis is that it is not accurate. As we talked before frequency analysis will based on the frequency of appearing of the letters in the cipher text then it will map it with the correspond letter in the alphabet. There are some letters that are too close to each other and the margin for error is quite high.

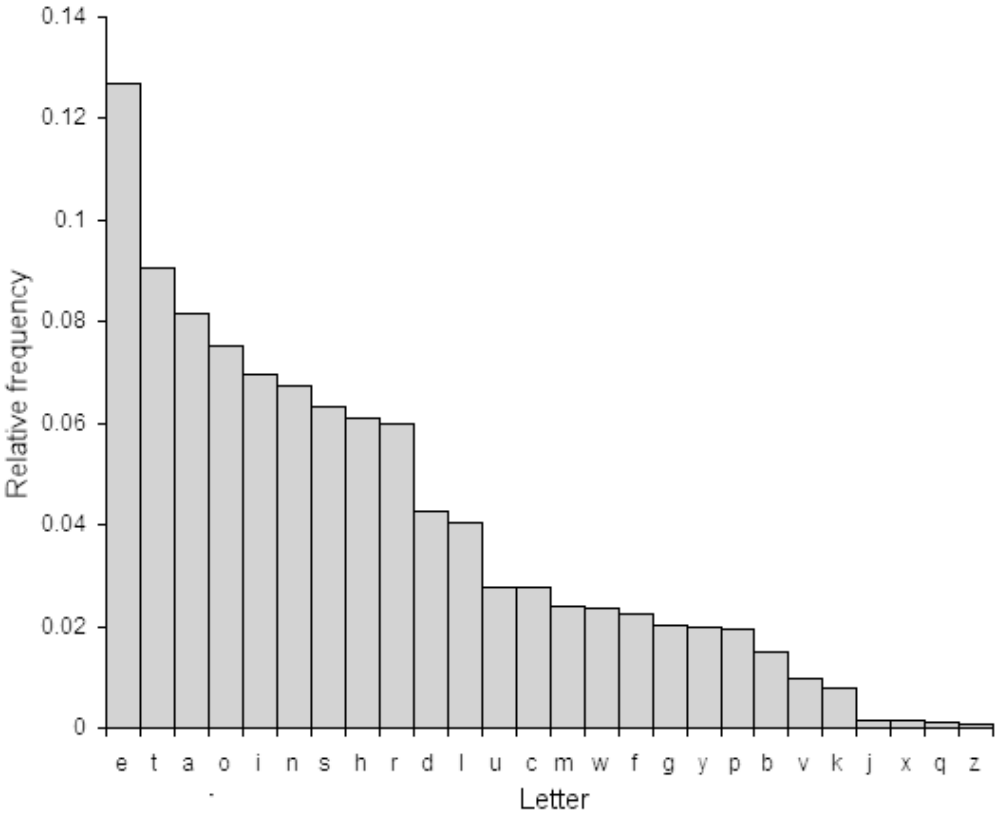
More over, frequency analysis works best with passage that have longer words, cipher text that has 100 words above are more accurate to be deduced than cipher text that has 50 words. Not only that, there are some passages that only used a certain amount letters in the alphabet so we can not fully deduce a cipher. And as we talked before, we need cipher text that have more words will lead to a more accurate decryption but with longer cipher text we will have to use more resources to fuly derypt them.

Furthermore, frequency analysis is a basic way to decrypt a message but it is a brute force way, so not only you need to have a longer cipher text, you would also need to kknow the context or the situation that the cipher text is referencing. For example, British crytographer in the past when trying to decrypt message from the Nazis, they tried to find the common words “Hail Hitler” and then decode the message from there.

3. Method of frequency analysis:

As I said before, with frequency analysis we have 2 basic problem, the first one is with the length of the cipher text, the longer the better, the second one is with the

context of the cipher text. So, giving that we have a cipher text with no context or source that we can reference. How do we do it?



First we need to calculate the frequency of all the letters in the cipher text. After we got our calculation of the frequency, then we tried to map that with a common frequency table (there are a lot of frequency table on the internet with different values but most of them are the same with the first 6 letters, we will use the table on wikipedia). We agreed on that cipher text is in the uppercase while plain text or decrypted text will be in the lowercase.

Secondly, we will tried to map some of the common words that are used in literature.

One-Letter Words	a, I.
Frequent Two-Letter Words	of, to, in, it, is, be, as, at, so, we, he, by, or, on, do, if, me, my, up, an, go, no, us, am
Frequent Three-Letter Words	the, and, for, are, but, not, you, all, any, can, had, her, was, one, our, out, day, get, has, him, his, how, man, new, now, old, see, two, way, who, boy, did, its, let, put, say, she, too, use
Frequent Four-Letter Words	that, with, have, this, will, your, from, they, know, want, been, good, much, some, time

Thirdly, we will tried to map with common digram and trigram (these are common 2 and 3 letters words that usually go together)

Most Frequent Digraphs	th er on an re he in ed nd ha at en es of or nt ea ti to it st io le is ou ar as de rt ve
Most Frequent Trigraphs	the and tha ent ion tio for nde has nce edt tis oft sth men

After that, we will tried to fill out the remaining words by guessing the word and context of the passage.

This is a picture showing the common words that are used in English.

ENGLISH	
Order Of Frequency Of Single Letters	E T A O I N S H R D L U
Order Of Frequency Of Digraphs	th er on an re he in ed nd ha at en es of or nt ea ti to it st io le is ou ar as de rt ve
Order Of Frequency Of Trigraphs	the and tha ent ion tio for nde has nce edt tis oft sth men
Order Of Frequency Of Most Common Doubles	ss ee tt ff ll mm oo
Order Of Frequency Of Initial Letters	T O A W B C D S F M R H I Y E G L N P U J K
Order Of Frequency Of Final Letters	E S T D N R Y F L O G H A K M P U W
One-Letter Words	a, I.
Most Frequent Two-Letter Words	of, to, in, it, is, be, as, at, so, we, he, by, or, on, do, if, me, my, up, an, go, no, us, am
Most Frequent Three-Letter Words	the, and, for, are, but, not, you, all, any, can, had, her, was, one, our, out, day, get, has, him, his, how, man, new, now, old, see, two, way, who, boy, did, its, let, put, say, she, too, use
Most Frequent Four-Letter Words	that, with, have, this, will, your, from, they, know, want, been, good, much, some, time

4. Personal comments about frequency analysis:

Frequency analysis is the most common and easiest way to decode a monoalphabetic subtitution cipher with trying all 26! keyword permutations but it is not the most effective way to decrypt a cipher. It relies on the frequncy of words in the cipher text to break the cipher text, it needs a long cipher text to correctly decode a cipher and also it needs to know the context of the cipher text to accurately decipher the text.

In conclusion, frequency analysis is a simple way for anyone that want to decode a cipher, with simple instruction but it is not the most effective way to decode a cipher.

CHAPTER 4 – EXPERIMENT

I HIGHLY ADVISED TO RUN THE PROGRAM IN GOOGLE COLAB BECAUSE THERE ARE A LOT OF LIBRARIES THAT CAN ONLY BE USED ON THERE WITHOUT THE NEED TO INSTALL OTHERS TO YOUR MACHINE. I WILL HAVE DETAILED INSTRUCTION ON HOW TO RUN THE TEXT FILES AND THE PROGRAM IN GOOGLE COLAB.

1. Instructions on building my program for encrypt and decrypt:

The program consist of **2 main functions called encrypt and decrypt**, there do exists extra functions that assist these 2 main functions but we will talk about them later.

a. Encryption:

As I said before, we will used monoalphabetic subtitution cipher to encrypt our plain text. We will first need to read our plain text from our file.

```
def readFile(links):
  with open(links, "r") as file:
    input_string = file.read()
  file.closed
  return input_string
```

This will read the plain text in a file called ***"input_plain_text.txt"*** and return a string value, which will be pass as a parameter into a function called encryption.

```
res = encryption(readFile("/content/drive/MyDrive/Colab Notebooks/input_plain_text.txt").lower(), keyword)
```

This function will take **2 input as parameters**, one is the plain text as a string input, the second one is the keyword *(if we do not have a keyword, we will replace it with "None")*.

```
def encryption(plain_text, keyword):
  default_alphabet = ['a','b','c','d','e','f','g','h','i',
'j','k','l','m','n','o','p','q','r','s','t','u','v','w','x
','y','z']
  if keyword != None:
    keyword_alphabet = create_keyword_alphabet(keyword, de
fault_alphabet)
  else:
    keyword_alphabet = random_keyword_string(default_alpha
bet)
  result = monoalphabetic_substitution_encryption(keyword_
alphabet, plain_text)
  return result.upper()
```

This is the encryption fucntion, which have a variable called default_alphabet that house all the letters of the alphabet in a list, this is used as the default alphabet for every step of the encryption. **First we need to have a cipher alphabet**, is the alphabet that we used to encrypt our plain text string. We have 2 cases here, one is with a keyword, the other is without a keyword, we will give instruction on creating the **cipher alphabet without the keyword first**:

```
def random_keyword_string (input_alphabet):
  return list_to_string(random.sample(input_alphabet,
26))
```

This is the function that will generate a keyword within the 26! permutations. It takes in the default alphabet list and run the sample 26 times to create a list with 26 differennt values in the alphabet. After which, it used the function **list_to_string** to turn the cipher alphabet list into a cipher alphabet string.

```
def list_to_string (input_string):
  return ''.join(item for item in input_string)
```

The **list_to_string** function will iterate throught the list and join them together, creating a string.

Above is creating the cipher alphabet without create a keyword, we will move to creating the cipher alphabet with a keyword. With a keyword, we need to remove any special symbols, repeated letters and blankspaces between the letters in order to create a unique keyword that we will need to create the cipher alphabet:

```
def create_keyword_alphabet(input_keyword, alphabet):
  li = string_to_list("".join(dict.fromkeys(input_key
word)))
  l3 = [x for x in alphabet if x not in li]
  li.extend(l3)
  return list_to_string(li)
```

We will striped the keyword, remove any blabkspaces and repeated letters from a key word. Next, we use the function **string_to_list** to turn they keyword into a list. After that, we will iterate through the default_alphabet and append the rest of the letters that are not in the keyword. Finally, we use the function **list_to_string** to turn the cipher alphabet list into a string and return it.

```
def string_to_list(input_string):
    li = []
    for i in input_string:
      li.append(i)
    return li
```

This function will iterate through our input_string and turn it into a list.

After we got our cipher alphabet either from a keyword or without a keyword, we moved to our next function with is called monoalphabetic_subtituion_encryption, which will encrypt our plain text based on the cipher alphabet using monoalphabetic subtitution. This fucntion take **2 parameters** as input called keyword_alphabet *(the cipher alphabet)* and plaintext *(our plain text that we read from the input_plain_text.txt).*

```
def monoalphabetic_substitution_encryption(keyword_alphabe
t, plaintext):
  cipher = ''
  for i in plaintext:
```

```
    if i in string.ascii_lowercase:
      index = ord(i) - ord('a')
      cipher = cipher + keyword_alphabet[index]
    else:
      cipher = cipher + i
  return cipher
```

This will create a varialbe called cipher which will be our cipher text. It iterate through the plaintext getting the index of each letter in the plain text then find the correct cipher letter in the cipher alphabet, then append it to the variable cipher.

When we iterate through the end, we will have our completed cipher text in the variable cipher and the function will return it.

After that we will used the function called writeFile to write our cipher text into a file called "**encrypted_text.txt**"

```
writeFile("/content/drive/MyDrive/Colab Notebooks/encrypte
d_text.txt", res)
```

This is will call the function write file, it takes in 2 parameters, one is the link to the file, the other is the data or the string.

```
def writeFile(links, data):
  with open(links, "w") as file:
    file.write(data)
  print("\t\t\t\t\t>>>COMPLETED WRITING TO FILE<<<")
  # print(file.read())
```

This is the function that will write our file. It takes in 2 parameters, one is the link of our file, the other is the data or the string that we just encrypted.

b. Dencryption:

We will use frequency analysis on the decryption process. We will read the cipher text from a file called "input_cipher_text" and have our results in 2 file called "decrypted_text" and "cipher_breaker". We will first need to create 4 gobal variable called ALPHABET *(containing string of the alphabet)*, ETAOIN *(containing string and*

order of the most common letters in the alphabet), common_two_words_list *(containing the common 2 letters words in literature),* common_three_words_list *(containing the common 3 letters words in literature),* common_four_letters_list *(containing the common 4 letters words in literature)*

We will first run the decrypt function. This function will remove first any symbols that do not belong to the alphabet and then run a function called **switch_crack:**

```
def switch_crack(string):
    key = {}
    frequent_letters = collections.Counter(string).most_co
mmon()
    index = 0
    for letter in frequent_letters:
        if letter[0] in ALPHABET:
            key[ETAOIN[index]] = letter[0]
            index += 1
    return key
```

This function will count the frequency of letters in the cipher text and map them with the ETAOIN list which is a list that contains the order of most common word, and from my experiment the 3 letters ETA has the most accurate values when mapping from a cipher text to a plain text using frequency analysis. So the function will replace every words in the cipher text with the corresponded ETA letters.

```
  for i in res3:
    if "’" in i:
      # print(i[-1])
      input_string_2 = input_string_2.replace(i[-1], 's')
      temp = i[-1]
      res3 = [i.replace(temp, 's') for i in res3]
      cipher_breaker[i[-1]] = 's'

  common_1 = [i for i in res3 if len(i) == 1]
  for i in common_1:
    if i.islower() == False:
      temp = i
```

```
      input_string_2 = input_string_2.replace(temp, 'i')
      res3 = [i.replace(temp, 'i') for i in res3]
      cipher_breaker[temp] = 'i'
```

If a word that stand alone in the cipher text it is either the letter A or the letter I, and because we already reaplce the letter A, all the letters that stand alone afterward are the letter I. And we learn in English that words containing the symbol of Possessive Pronouns end with the letter S. So, this is what these lines of code do, they replace the cipher text with the common letter I and S.

We move on the the next function which is called “replace_common_word”, this function will read the cipher text and find the common words that appear in the cipher text, this function will reapeat 3 times for the 2 common words, 3 common words and 4 common words. After that, it will return a list containing the decode cipher text, the cipher breaker and res3 which contains the last remaing un-cipherable text *(this will be handle at the end of the function).*

We move on to the last functions that will fully decrypt the cipher text

```
def getmatches(matchstring, word_list):
  matchstring = matchstring.replace('-', '.')
  return [word for word in word_list if re.fullmatch(match
string, word)]
def replace_uppercase_with_dash(input_string):
  s = '' + input_string
  for i in s:
    if i.isupper() == True:
      s = s.replace(i,'-')
  return s
def create_sub_list_dictionary_words(word, dictionary_list
):
  word_replaced_dash = replace_uppercase_with_dash(word)
  result = getmatches(word_replaced_dash, dictionary_list)
  return result
def formula_1(input_string, input_string_2, cipher_breaker
, res3, dictionary_list):
```

```
  clear_output()
  for word_index in range(len(res3)):
    clear_output()
    # print("the 1st:", res3[word_index],"\n", res3)
    sub_list = create_sub_list_dictionary_words(res3[word_
index], dictionary_list)
    if ((res3[word_index].isupper() == False) and (res3[wo
rd_index].islower() == False)) and (len(sub_list) != 0):
      print(">Cipher text:\n\t", input_string_2)
      print(">This is your word:", res3[word_index], "\n>T
his is the suggested list:", sub_list)
      token_2 = True
      while (token_2 == True):
        uppercase_input = input("Enter letter in UPPERCASE
: ")
        lowercase_input = input("Enter letter in lowercase
: ")
        #action in here
        uppercase_input = uppercase_input.strip().upper()[
0]
        lowercase_input = lowercase_input.strip().lower()[
0]
        if (len(uppercase_input)) > 1 or (len(lowercase_in
put) > 1):
          token_2 = True
        elif ((lowercase_input in cipher_breaker.values())
 == True) or (uppercase_input in cipher_breaker) == True:
          print("\t\t\t>>>EXISTED<<<")
          token_2 == True
        else:
          temp_string = '' + input_string_2
          temp_string = temp_string.replace(uppercase_inpu
t, lowercase_input)
          print("\t>Cipher text and plain text (REPLACED):
\n", "\t\t", input_string,"\n\t\t", temp_string)
          # are you sure?
          user_option = str(input("> Are you sure you want
 to replace? (1 for YES, 0 for NO): "))
          if user_option == '1':
            input_string_2 = input_string_2.replace(upperc
ase_input, lowercase_input)
```

```
            cipher_breaker[uppercase_input] = lowercase_in
put
            res3 = [i.replace(uppercase_input, lowercase_i
nput) for i in res3]
            token_2 = False
            break
          else:
            token_2 = True
  return [input_string_2, cipher_breaker, res3]
```

These function first will read a textfile that contains every words exist in the dictionary to create a dictionary list used later for references.

The **getmatches**, **replace_uppercase_with_dash** and **create_sub_list_dictionary_words** functions are a combo functions. The **replace_uppercase_with_dash** funtion will iterate through the rest of the cipher text and replace any words containing uppercase with the ‘-‘ symbol, after that the **getmatches** function will find all the words that have dahes, run through the dictionary list and find all the words in the dictionary that can be a possible result of each of the words. Finally, **create_sub_list_dictionary_words** will generate and return all of the possible outcomes of that words.

This process will be used in the **formula_1** function which will be the last function to decrypt the cipher text.

```
def formula_1(input_string, input_string_2, cipher_breaker
, res3, dictionary_list):
  clear_output()
  for word_index in range(len(res3)):
    clear_output()
    # print("the 1st:", res3[word_index],"\n", res3)
    sub_list = create_sub_list_dictionary_words(res3[word_
index], dictionary_list)
    if ((res3[word_index].isupper() == False) and (res3[wo
rd_index].islower() == False)) and (len(sub_list) != 0):
      print(">Cipher text:\n\t", input_string_2)
```

```
      print(">This is your word:", res3[word_index], "\n>T
his is the suggested list:", sub_list)
      token_2 = True
      while (token_2 == True):
        uppercase_input = input("Enter letter in UPPERCASE
: ")
        lowercase_input = input("Enter letter in lowercase
: ")
        #action in here
        uppercase_input = uppercase_input.strip().upper()[
0]
        lowercase_input = lowercase_input.strip().lower()[
0]
        if (len(uppercase_input)) > 1 or (len(lowercase_in
put) > 1):
          token_2 = True
        elif ((lowercase_input in cipher_breaker.values())
 == True) or (uppercase_input in cipher_breaker) == True:
          print("\t\t\t>>>EXISTED<<<")
          token_2 == True
        else:
          temp_string = '' + input_string_2
          temp_string = temp_string.replace(uppercase_inpu
t, lowercase_input)
          print("\t>Cipher text and plain text (REPLACED):
\n", "\t\t", input_string,"\n\t\t", temp_string)
          # are you sure?
          user_option = str(input("> Are you sure you want
 to replace? (1 for YES, 0 for NO): "))
          if user_option == '1':
            input_string_2 = input_string_2.replace(upperc
ase_input, lowercase_input)
            cipher_breaker[uppercase_input] = lowercase_in
put
            res3 = [i.replace(uppercase_input, lowercase_i
nput) for i in res3]
            token_2 = False
            break
          else:
            token_2 = True
  return [input_string_2, cipher_breaker, res3]
```

This function contain a loop that will loop through the rest of the cipher text, it will not process words that are fully uppercase, lowercase or do not exist in the dictionary.

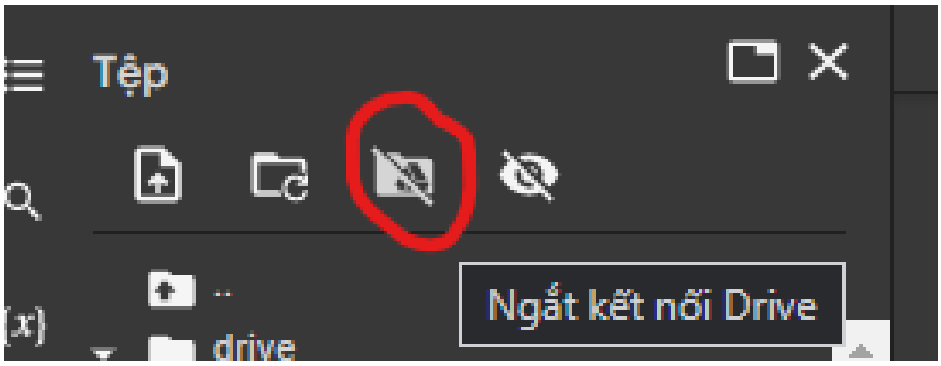
After the process is done, it will write to a file called decrypted_text

2. Instructions on running my program for encrypt and decrypt:

I HIGHLY ADVISED TO RUN THE PROGRAM IN GOOGLE COLAB BECAUSE THERE ARE A LOT OF LIBRARIES THAT CAN ONLY BE USED ON THERE WITHOUT THE NEED TO INSTALL OTHERS TO YOUR MACHINE. I WILL HAVE DETAILED INSTRUCTION ON HOW TO RUN THE TEXT FILES AND THE PROGRAM IN GOOGLE COLAB.

When we download the file it will contains 8 files: 520H0401.py, 520H0401.ipynb, english3.txt, input_plain_text, input_cipher_text, encrypted_text, decrypted_text and cipher_breaker.

Firstly, we connect with google colab and then connect with google drive by clicking this button.



After that, we locate to the folder that contains our 8 files, and then we copy the path of each files and place them in the correct variable:

input_plain_text_link for the plain text

encrypted_text_link for the result after we encrypt it

input_cipher_text_link for the cipher text

english3_text_link for the dictionary

decrypted_text_link for the decrypted text

cipher_breaker_link for the cipher breaker file after decrypt

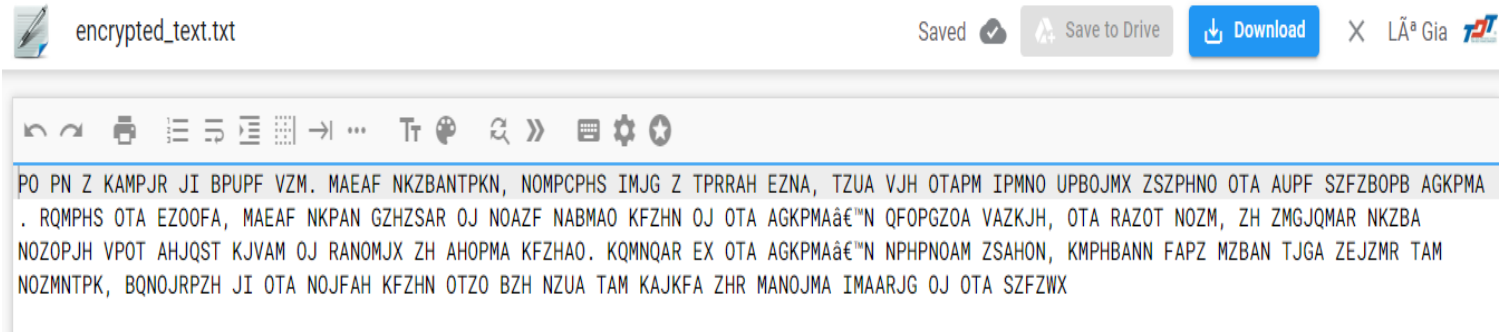
a. Encryption:

To run the encryption, we will need to fill in the links for **input_plain_text_link** and **encrypted_text_link.** The program will read the plain text in our file **automatically** and write the answer in the other file.

b. Decryption:

To run the decryption, we will need to fill in the links for **input_cipher_text_link, english3_text_link, decrypted_text_link** and **cipher_breaker_link.** The program will run automatically and write the answer in the correspond files. We will demonstrate how to run the program later in the next section.

3. <u>Demo on running the program to encrypt a plain text and decrypt cipher text:</u>



This is after we run the encryption

```
REPLACE COMMON (2) LETTERS WORDS:
Cipher Text:
        Pt Ps a KeMPJR JI BPUPF VaM. MeEeF sKaBesTPK, stMPCPHS IMJG a TPRReH Ease, TaUe VJH tTePM IPMst UPBtJMX aSaPHst tT
What do you want to do? (1 for break cipher text/continue, 0 for cancel):
```

After we run the dedcrypt program, it will lead us to the common 2 letter, we press 1 if we want to continue and 0 if we want to end.

```
REPLACE COMMON (2) LETTERS WORDS:
Cipher Text:
         Pt Ps a KeMPJR JI BPUPF VaM. MeEeF sKaBesTPK, stMPCPHS IMJG a TPRReH Ease, TaUe VJH tTePM IPM
What do you want to do? (1 for break cipher text/continue, 0 for cancel): 1
        >Common (%d) words:
                ['of', 'to', 'in', 'it', 'is', 'be', 'as', 'at', 'so', 'we', 'he', 'by', 'or', 'on',
        >Cipher text and plain text:
                PO PN Z KAMPJR JI BPUPF VZM. MAEAF NKZBANTPK, NOMPCPHS IMJG Z TPRRAH EZNA, TZUA VJH O
                Pt Ps a KeMPJR JI BPUPF VaM. MeEeF sKaBesTPK, stMPCPHS IMJG a TPRReH Ease, TaUe VJH t
        >Cipher breaker:
                {'A': 'e', 'O': 't', 'Z': 'a', 'N': 's'}
Enter letter in UPPERCASE:
```

This section will replace common two letter words, this process will repeat until we have fully replaced all of the common two letter words.

```
REPLACE COMMON (3) LETTERS WORDS:
Cipher Text:
        it is a KeMioR of BiUiF VaM. MeEeF sKaBesTiK, stMiCinS fMoG a TiRRen Ease, TaUe Von tTeiM fi
What do you want to do? (1 for break cipher text/continue, 0 for cancel):
```

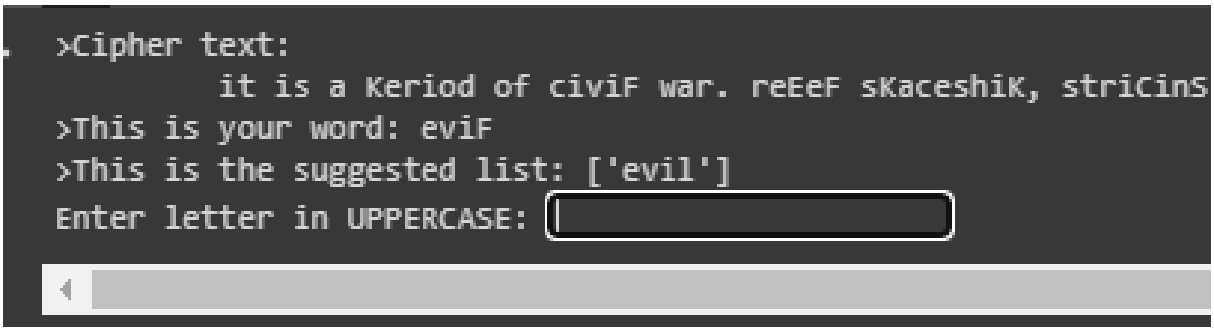
```
...  REPLACE COMMON (3) LETTERS WORDS:
    Cipher Text:
             it is a KeMioR of BiUiF VaM. MeEeF sKaBesTiK, stMiCinS fMoG a TiRRen Ease, TaUe Von tTeiM fiM
    What do you want to do? (1 for break cipher text/continue, 0 for cancel): 1
            >Common (%d) words:
                    ['the', 'and', 'for', 'are', 'but', 'not', 'you', 'all', 'any', 'can', 'had', 'her',
            >Cipher text and plain text:
                    PO PN Z KAMPJR JI BPUPF VZM. MAEAF NKZBANTPK, NOMPCPHS IMJG Z TPRRAH EZNA, TZUA VJH O
                    it is a KeMioR of BiUiF VaM. MeEeF sKaBesTiK, stMiCinS fMoG a TiRRen Ease, TaUe Von t
            >Cipher breaker:
                    {'A': 'e', 'O': 't', 'Z': 'a', 'N': 's', 'P': 'i', 'J': 'o', 'I': 'f', 'H': 'n'}
    Enter letter in UPPERCASE: T
    Enter letter in lowercase: h
            >Cipher text and plain text (REPLACED):
                    PO PN Z KAMPJR JI BPUPF VZM. MAEAF NKZBANTPK, NOMPCPHS IMJG Z TPRRAH EZNA, TZUA VJH O
                    it is a KeMioR of BiUiF VaM. MeEeF sKaBeshiK, stMiCinS fMoG a hiRRen Ease, haUe Von t
    > Are you sure you want to replace? (1 for YES, 0 for NO): [redacted]
```

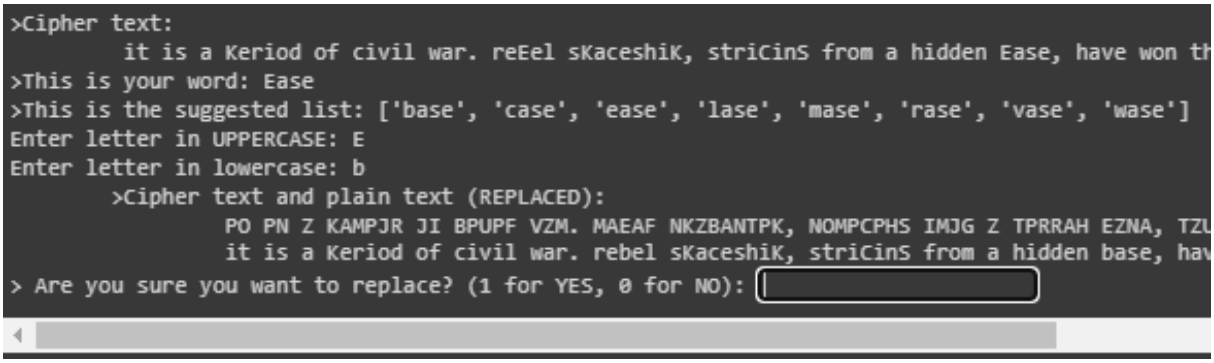
These two picures are the same process as the replace common two letter words. It will repeat until we have replaced all three letters.

```
REPLACE COMMON (4) LETTERS WORDS:
Cipher Text:
        it is a Keriod of ciUiF Var. reEeF sKaceshiK, striCinS froG a hidden Ease, haUe Von their first Uicto
What do you want to do? (1 for break cipher text/continue, 0 for cancel): [redacted]
```

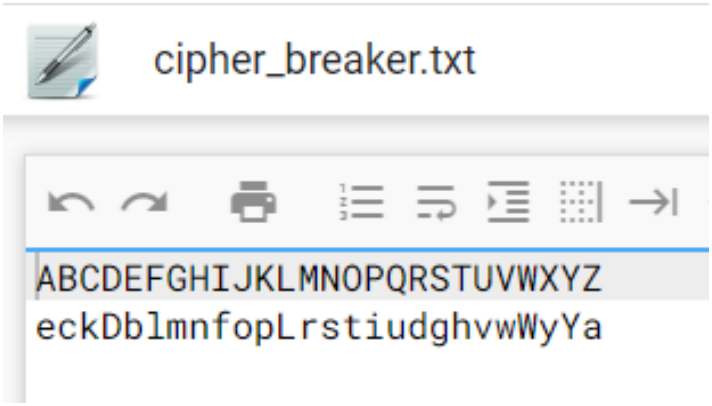
```
REPLACE COMMON (4) LETTERS WORDS:
Cipher Text:
         it is a Keriod of ciUiF Var. reEeF sKaceshiK, striCinS froG a hidden Ease, ha
What do you want to do? (1 for break cipher text/continue, 0 for cancel): 1
        >Common (%d) words:
                ['that', 'with', 'have', 'this', 'will', 'your', 'from', 'they', 'kno
        >Cipher text and plain text:
                PO PN Z KAMPJR JI BPUPF VZM. MAEAF NKZBANTPK, NOMPCPHS IMJG Z TPRRAH
                it is a Keriod of ciUiF Var. reEeF sKaceshiK, striCinS froG a hidden
        >Cipher breaker:
                {'A': 'e', 'O': 't', 'Z': 'a', 'N': 's', 'P': 'i', 'J': 'o', 'I': 'f'
Enter letter in UPPERCASE: G
Enter letter in lowercase: m
        >Cipher text and plain text (REPLACED):
                PO PN Z KAMPJR JI BPUPF VZM. MAEAF NKZBANTPK, NOMPCPHS IMJG Z TPRRAH
                it is a Keriod of ciUiF Var. reEeF sKaceshiK, striCinS from a hidden
> Are you sure you want to replace? (1 for YES, 0 for NO): [redacted]
```

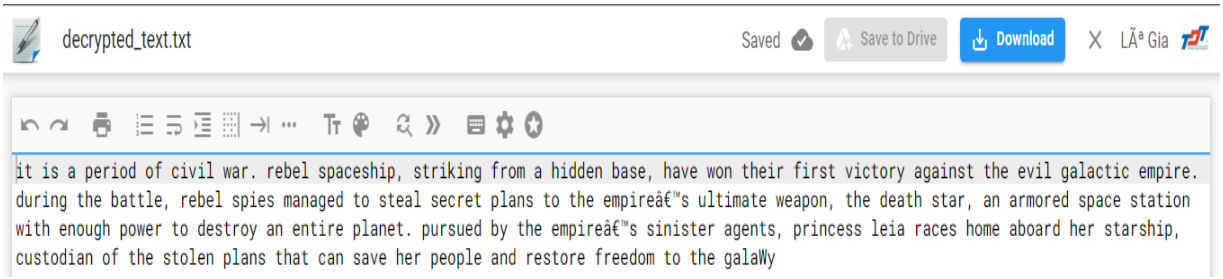
This process will reapeat until we have fully replaced all the common 2, 3 and 4 letter words. After this our cipher text is almost ready, we just need to replace a few words. So the last function will do this job, it will iterate through the whole text and find the most suitable letter to replace





After we finished replacing all of our text we will have 2 files called decrypted_text and cipher_breaker which contain our results





REFERENCES DOCUMENT

[1]. Tutorialspoint, Traditional Ciphers, accessed 10/4/2022, https://www.tutorialspoint.com/cryptography/traditional_ciphers.htm

[2]. 101Computing, Frequency Analysis, accessed 10/4/2022, https://www.101computing.net/frequency-analysis/

[3]. 101Computing, Monoalphabetic Subtitution Cipher, accessed 10/4/2022, https://www.101computing.net/mono-alphabetic-substitution-cipher/

[4]. CS Cornell, Introduction to Cryptography, accessed 10/4/2022, http://www.cs.cornell.edu/courses/cs513/2002fa/L23.html

[5]. Data Privacy and Security, Monoaphabetic Subtitution Ciphers, accessed 10/4/2022, https://link.springer.com/chapter/10.1007/978-0-387-21707-9_2?noAccess=true

[6]. Practical cryptography, Simple Subtitution Cipher, accessed 10/4/2022, http://www.practicalcryptography.com/ciphers/simple-substitution-cipher/

[7]. Notre Dame University, Cryptanalysis Hints, accessed 10/4/2022, https://www3.nd.edu/~busiforc/handouts/cryptography/cryptography%20hints.html

[8]. Stackoverflow, Python frequency analysis, accessed 10/4/2022, https://stackoverflow.com/questions/29797475/python-frequency-analysis

[9]. Educative, What are symmetric and asymetric cryptosystems, accessed 10/4/2022, https://www.educative.io/edpresso/what-are-symmetric-and-asymmetric-cryptosystems

[10]. Red hat, Basic concepts of encryption in cryptography, accessed 10/4/2022, https://www.redhat.com/sysadmin/basic-concepts-encryption-cryptography