Kirubananth Sankar
Introduction to Programming and Python
18 September 2022

Programming Project

Problem Statement:

1. File handling:

   Convert an input text file to a string with only lowercase letters. Remove everything else, including whitespace.

2. Caesar shift Cipher:

   Given a string of lowercase letters, encrypt it using the caeser shift cipher. Then decrypt it.

3. Monoalphabetic Cipher (Random Cipher):

   Given a string of lowercase letters, encrypt it using a Monoalphabetic Cipher. Then decrypt it.

4. Vigenère cipher:

   Given a string of lowercase letters, encrypt it using Vigenere Cipher. Then decrypt it.

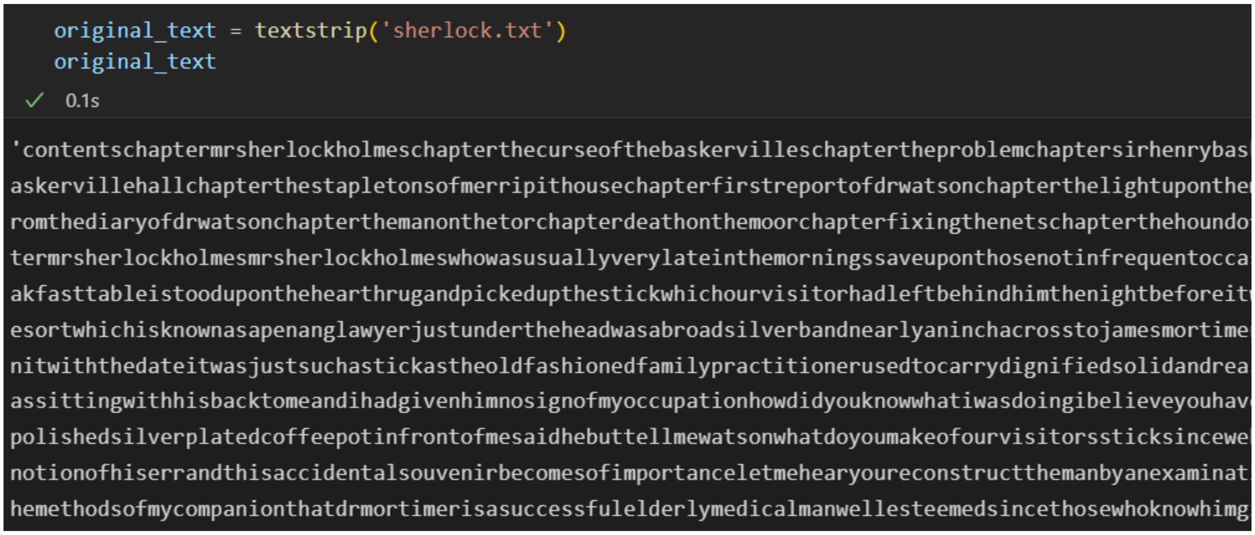
Algorithm and Results:

File Handling:

Algorithm:

- We are given a file named ‘sherlock.txt’ ’
- We are using ‘open’ and file.read, we store it to a variable ‘data’ as a string after converting everything into lowercase.
- Now we create a new string ‘output’. Using string.ascii_lowercase as a filter, we filter out only the lowercase letters from data and store them to output.
- This is all done inside a function named ‘textstrip.’’

Result snippet:



```
original_text = textstrip('sherlock.txt')
original_text
```

✓ 0.1s

```
'contentschaptermrsherlockholmeschapterthecurseofthebaskervilleschaptertheproblemchaptersirhenrybas
askervillehallchapterthestapletonsofmerripithousechapterfirstreportofdrwatsonchapterthelightuponthe
romthediaryofdrwatsonchapterthemanonthetorchapterdeathonthemoorchapterfixingthenetschapterthehoundo
termrsherlockholmesmrsherlockholmeswhowasusuallyverylateinthemorningssaveuponthosenotinfrequentocca
akfasttableistooduponthehearthrugandpickedupthestickwhichourvisitorhadleftbehindhimthenightbeforeit
esortwhichisknownasapenanglawyerjustundertheheadwasabroadsilverbandnearlyaninchacrosstojamesmortime
nitwiththedateitwasjustsuchastickastheoldfashionedfamilypractitionerusedtocarrydignifiedsolidandrea
assittingwithhisbacktomeandihadgivenhimnosignofmyoccupationhowdidyouknowwhatiwasdoingibelieveyouhav
polishedsilverplatedcoffeepotinfrontofmesaidhebuttellmewatsonwhatdoyoumakeofourvisitorssticksincewe
notionofhiserrandthisaccidentalsouvenirbecomesofimportanceletmehearyoureconstructthemanbyanexaminat
hemethodsofmycompanionthatdrmortimerisasuccessfulelderlymedicalmanwellesteemedsincethosewhoknowhimg
```

Caesar shift substitution:

In this method, the text is encoded with a substitution key that is formed by shifting the original alphabets by some number,
E.g., a b c d e f g h i j k . . . . (original letters)
      C D E F G H I J K L M. . . . (cipher substitution)
In the above example, the letters were shifted by ‘’2.”

Algorithm and Funcitons:

- We first create a dictionary ‘d_disp’ that acts as the substitution key.

- Encryption: substitution_encrypt(s,d):
  For this, we create a function called ‘substitution encrypt’ where we take in the stipped string ‘s’ from above and the key dictionary ‘d_disp.’ Now create a new result string, ‘res,’ which is formed by mapping every letter of ‘s’ to the values in ‘d_disp.’

- Frequency distribution: letter_distribution(s):
  For decrypting, we first need to find the frequency distribution of the letters in the encrypted message. The letter with the highest frequency will correspond to ‘e’ since ‘e’ is the letter with the highest frequency in English. This method of decryption is called “Frequency Analysis.”

  To find the frequencies of letters, we create a dictionary containing all letters as keys and 0 as values. Then we loop through our encrypted message, and as each letter appears, we increment the corresponding values in the dictionary.

  Finally, we find each letter's frequency percentage by doing (value/total value)*100

- Cryptanalyse: cryptanalyse_substitution_caser(s):

  We use the above function to find the frequency distribution of letters in the encrypted string, and the letter substituted for 'e.'

  Now we calculate the displacement as value(letter) - 4 (value of an alphabet is its place. E.g. value(a) = 0 , value(b) = 1 …)

  Based on the displacement, now we know how much caser shift happened. So we reverse the process. We create a dictionary now which maps every letter to the new letter, which is a letter at [value(letter) + displacement].

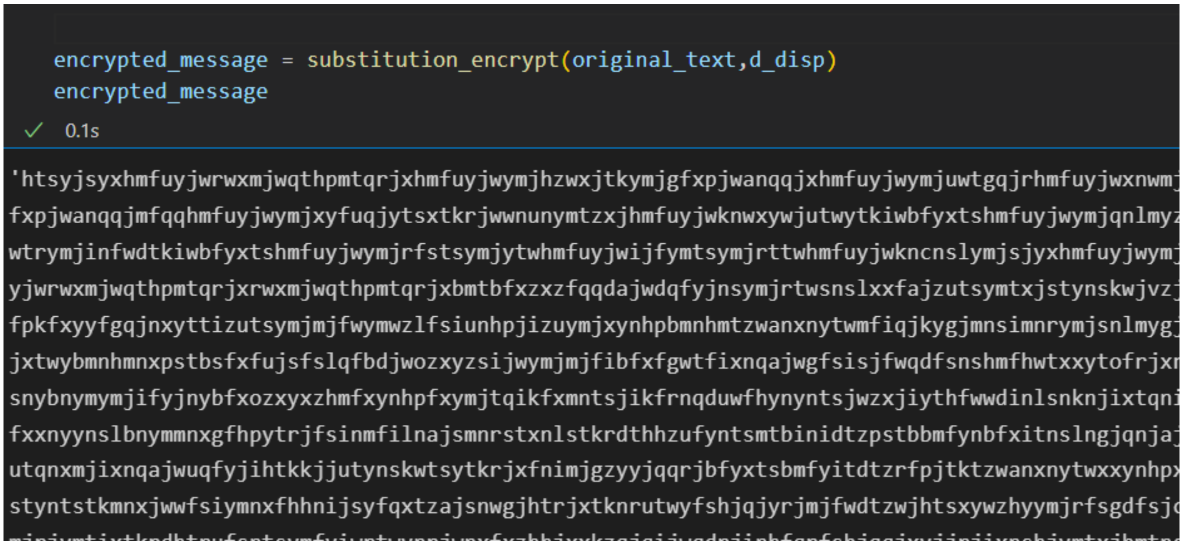
  E.g., displacement is 4. So a is mapped to e, b is mapped to f, and so on…

- Decrypt substitution_decrypt(s,d):

  Using the key dictionary we uncovered in the above step, we map every letter in the encrypted message to the corresponding letter in the dictionary. The result is the decrypted message.

Results:

1. Encrypted message:

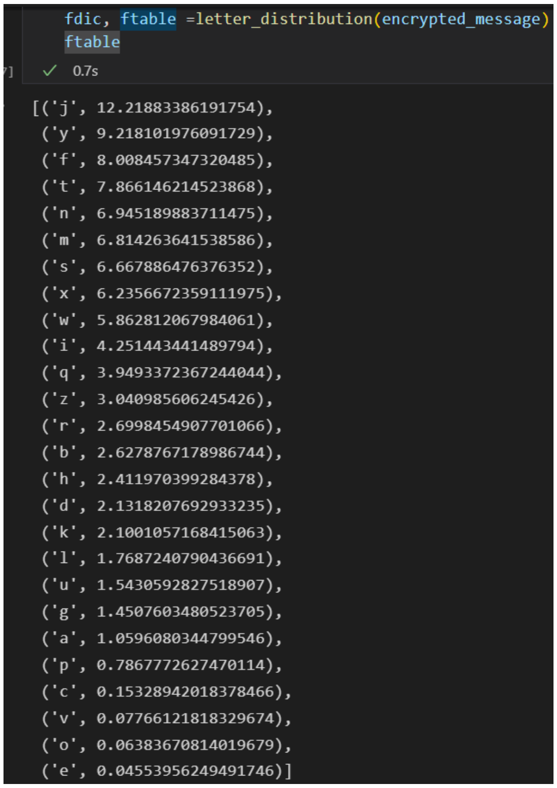


```
encrypted_message = substitution_encrypt(original_text,d_disp)
encrypted_message
```

✓ 0.1s

```
'htsyjsyxhmfuyjwrwxmjwqthpmtqrjxhmfuyjwymjhzwxjtkymjgfxpjwanqqjxhmfuyjwymjuwtgqjrhmfuyjwxnwmj
fxpjwanqqjmfqqhmfuyjwymjxyfuqjytsxtkrjwwnunymtzxjhmfuyjwknwxywjutwytkiwbfyxtshmfuyjwymjqnlmyz
wtrymjinfwdtkiwbfyxtshmfuyjwymjrfstsymjytwhmfuyjwijfymtsymjrttwhmfuyjwkncnslymjsjyxhmfuyjwymj
yjwrwxmjwqthpmtqrjxrwxmjwqthpmtqrjxbmtbfxzxzfqqdajwdqfyjnsymjrtwsnslxxfajzutsymtxjstynskwjvzj
fpkfxyyfgqjnxyttizutsymjmjfwymwzlfsiunhpjizuymjxynhpbmnhmtzwanxnytwmfiqjkygjmnsimnrymjsnlmygj
jxtwybmnhmnxpstbsfxfujsfslqfbdjwozxyzsijwymjmjfibfxfgwtfixnqajwgfsisjfwqdfsnshmfhwtxxytofrjx
snybnymymjifyjnybfxozxyxzhmfxynhpfxymjtqikfxmntsjikfrnqduwfhynyntsjwzxjiythfwwdinlsnknjixtqni
fxxnyynslbnymmnxgfhpytrjfsinmfilnajsmnrstxnlstkrdthhzufyntsmtbinidtzpstbbmfynbfxitnslngjqnjaj
utqnxmjixnqajwuqfyjihtkkjjutynskwtsytkrjxfnimjgzyyjqqrjbfyxtsbmfyitdtzrfpjtktzwanxnytwxxynhpx
styntstkmnxjwwfsiymnxfhhnijsyfqxtzajsnwgjhtrjxtknrutwyfshjqjyrjmjfwdtzwjhtsxywzhyymjrfsgdfsj
```

2. Frequency distribution:

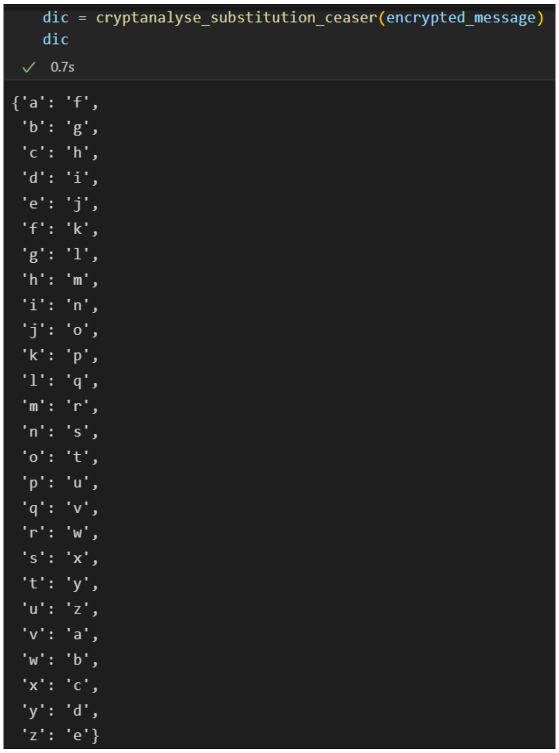


```
fdic, ftable =letter_distribution(encrypted_message)
ftable
```

✓ 0.7s

```
[('j', 12.21883386191754),
 ('y', 9.218101976091729),
 ('f', 8.008457347320485),
 ('t', 7.866146214523868),
 ('n', 6.945189883711475),
 ('m', 6.814263641538586),
 ('s', 6.667886476376352),
 ('x', 6.2356672359111975),
 ('w', 5.862812067984061),
 ('i', 4.251443441489794),
 ('q', 3.9493372367244044),
 ('z', 3.040985606245426),
 ('r', 2.6998454907701066),
 ('b', 2.6278767178986744),
 ('h', 2.411970399284378),
 ('d', 2.1318207692933235),
 ('k', 2.1001057168415063),
 ('l', 1.7687240790436691),
 ('u', 1.5430592827518907),
 ('g', 1.4507603480523705),
 ('a', 1.0596080344799546),
 ('p', 0.7867772627470114),
 ('c', 0.15328942018378466),
 ('v', 0.07766121818329674),
 ('o', 0.06383670814019679),
 ('e', 0.04553956249491746)]
```

3. Decrypted key dictionary:

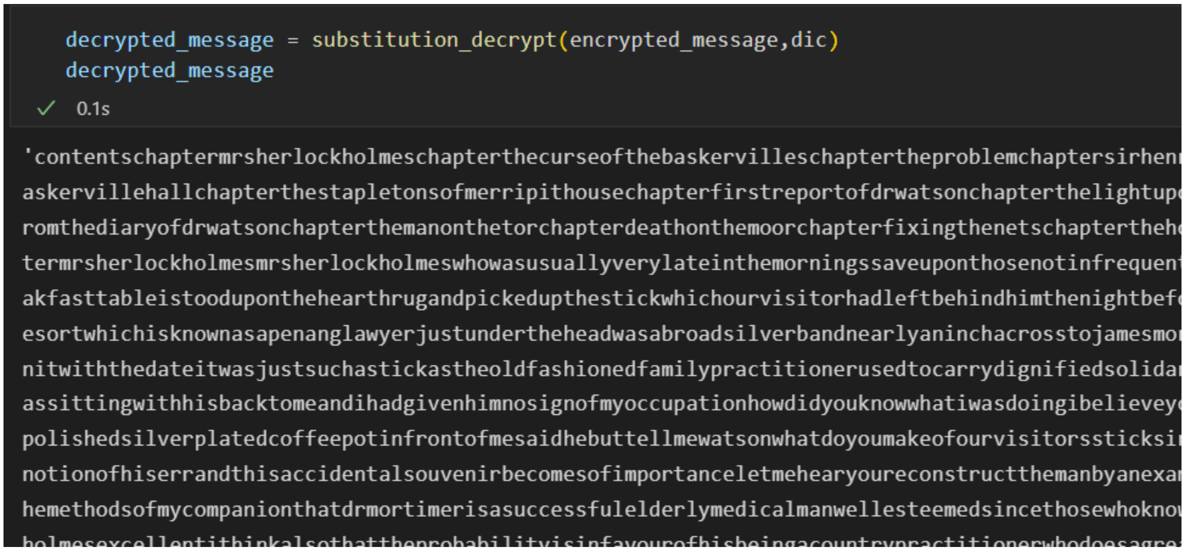


```
dic = cryptanalyse_substitution_ceaser(encrypted_message)
dic
```

✓ 0.7s

```
{'a': 'f',
 'b': 'g',
 'c': 'h',
 'd': 'i',
 'e': 'j',
 'f': 'k',
 'g': 'l',
 'h': 'm',
 'i': 'n',
 'j': 'o',
 'k': 'p',
 'l': 'q',
 'm': 'r',
 'n': 's',
 'o': 't',
 'p': 'u',
 'q': 'v',
 'r': 'w',
 's': 'x',
 't': 'y',
 'u': 'z',
 'v': 'a',
 'w': 'b',
 'x': 'c',
 'y': 'd',
 'z': 'e'}
```

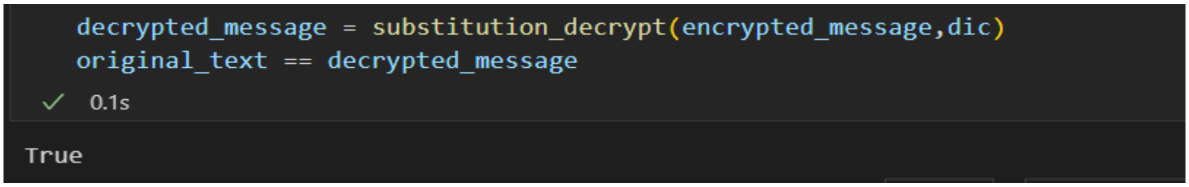
4. Decrypted message:



```
decrypted_message = substitution_decrypt(encrypted_message,dic)
decrypted_message
```

✓ 0.1s

```
'contentschaptermrsherlockholmeschapterthecurseofthebaskervilleschaptertheproblemchaptersirhen
askervillehallchapterthestapletonsofmerripithousechapterfirstreportofdrwatsonchapterthelightup
romthediaryofdrwatsonchapterthemanonthetorchapterdeathonthemoorchapterfixingthenetschaptertheh
termrsherlockholmesmrsherlockholmeswhowasusuallyverylateinthemorningssaveuponthosenotinfrequen
akfasttableistooduponthehearthrugandpickedupthestickwhichourvisitorhadleftbehindhimthenightbef
esortwhichisknownasapenanglawyerjustundertheheadwasabroadsilverbandnearlyaninchacrosstojamesmo
nitwiththedateitwasjustsuchastickastheoldfashionedfamilypractitionerusedtocarrydignifiedsolida
assittingwithhisbacktomeandihadgivenhimnosignofmyoccupationhowdidyouknowwhatiwasdoingibelievey
polishedsilverplatedcoffeepotinfrontofmesaidhebuttellmewatsonwhatdoyoumakeofourvisitorssticksi
notionofhiserrandthisaccidentalsouvenirbecomesofimportanceletmehearyoureconstructthemanbyanexa
hemethodsofmycompanionthatdrmortimerisasuccessfulelderlymedicalmanwellesteemedsincethosewhokno
holmesexcellentithinkalsothattheprobabilityisinfavourofhisbeingacountrypractitionerwhodoesagre
```



```
decrypted_message = substitution_decrypt(encrypted_message,dic)
original_text == decrypted_message
```

✓ 0.1s

```
True
```

Monoalphabetic Substitution:

In this method, the key dictionary can be any random combination of alphabets. There is no particular pattern for mapping.

E.g., a b c e f g h i … (message)
X H I O Y K L P… (cipher).

We could only predict an approximate key dictionary during decryption for this encryption method.

Algorithms for Decryption:

1. *Letters with the top three highest frequencies are generally ‘e,’ ‘t,’ and ‘a.’ :*
   Using this, we can narrow down the substitution for ‘e,’’t’, and ‘a’ by looking at the frequency table and taking the top 3.

2. *The letter with a frequency higher than 12% is probably ‘e.’:*
   If a letter in our frequency table has a value higher than 12, we can map it to ‘e’ with high confidence.

3. *The letter ‘h’ mostly appears before ‘e’ but rarely after.*

   Since we have already uncovered the substitution for ‘e,’ we can use it to make a table for every other letter based on the number of times they come before and after ‘e.’ The letter with the highest difference in this is ‘h.’

4. *The most frequent three-letter words are ‘and’ and ‘the’:*

   We create a dictionary. We loop through the decrypted message, recording every three-letter letter substring. If the substring is already present in the dictionary, we increment its value by 1. Otherwise, we record a new key and value pair with the key as the substring and value as 1. This way, we have noted the frequency of all three letter words.

   The substrings with the top two from the above table must be ‘and’ and ‘the.’ Since we already know the substitutions for ‘h’ and ‘e,’ we can find ‘t’ based on that. Consequently, we can narrow down ‘a’,’n,’ and ‘d’.

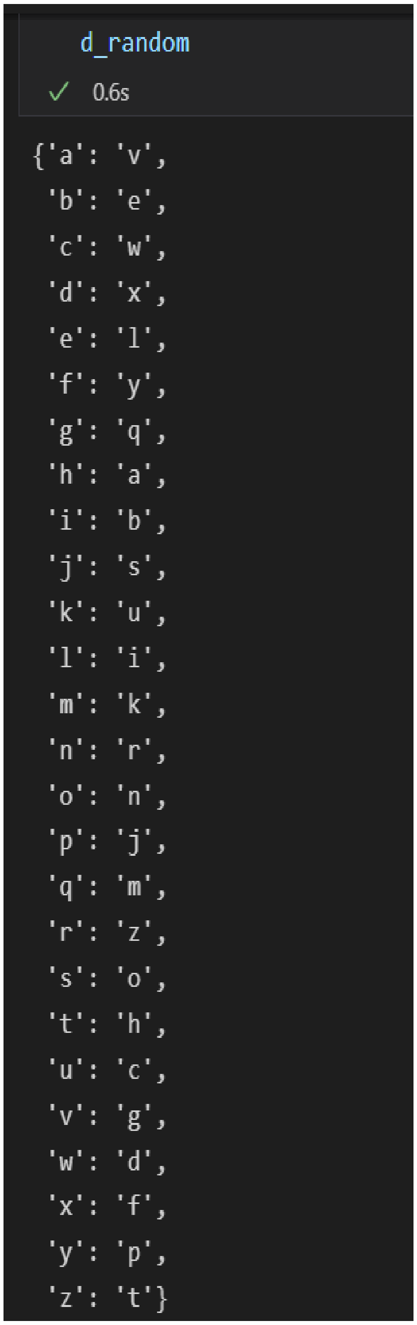
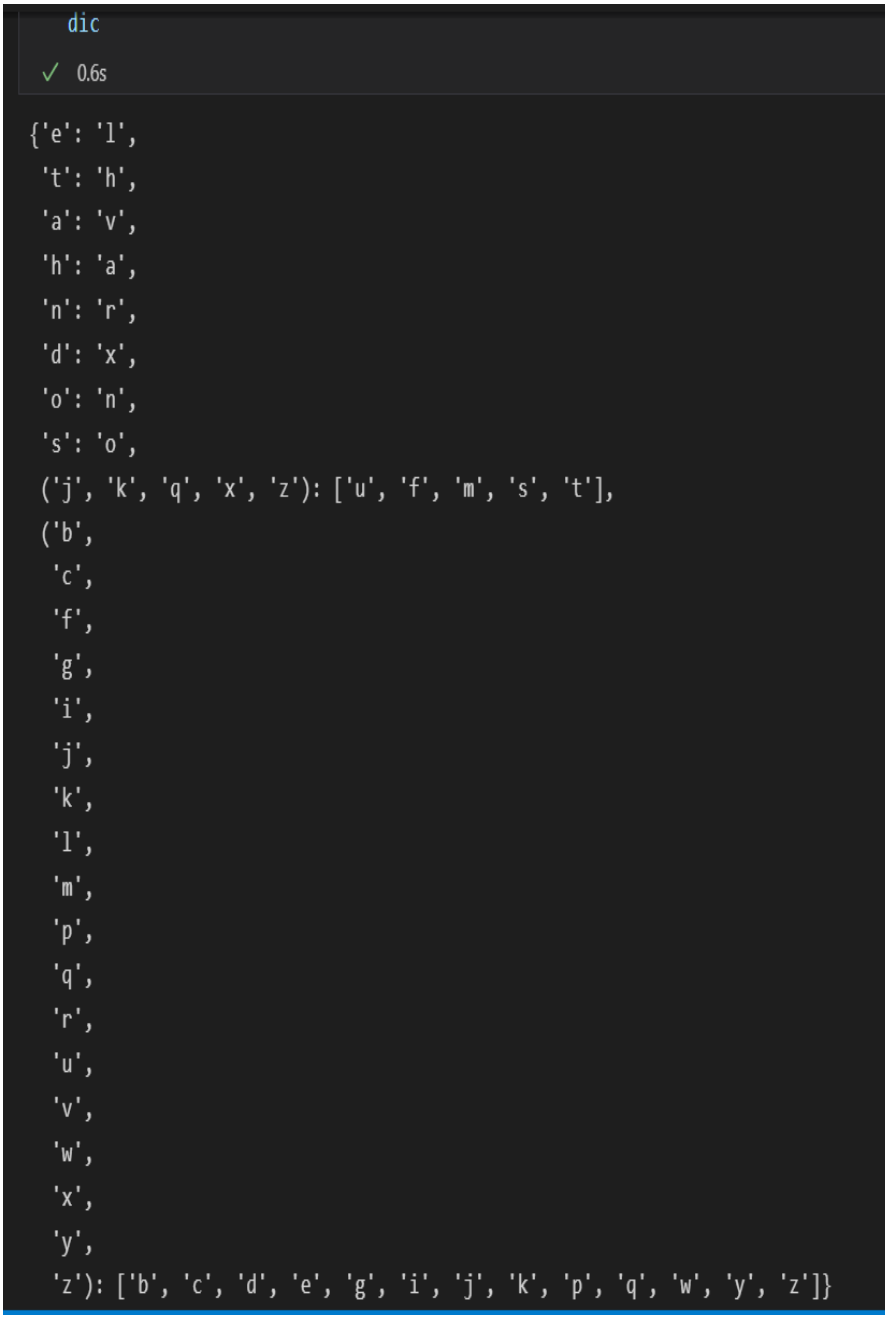
5. *The most frequent double letters are ‘ss,” tt,” oo,’ ‘mm,’ ‘ll,’ and ‘ff’.*
6. *Both ‘o’ and ‘s’ have high frequency and frequency of ‘o’ >> frequency of ‘s.’*

   Similar to what we did in the case of the three-letter substrings, we can make a dictionary with the frequency of two double-letter substrings. Among the top seven, there must be substrings containing ‘oo’ and ‘ss.’ Now by comparing the individual frequencies of each letter in the top substrings, we can successfully narrow down ‘o’ and ‘s.’

7. *The letters ‘j’, ‘k,’ ‘q,”x’, and ‘z’ generally have frequencies less than 1*

8. *We can map the remaining letters to the remaining keys*

Result:

Actual Key Dictionary	Predicted Dictionary
	

```
d_random
✓ 0.6s
{'a': 'v',
 'b': 'e',
 'c': 'w',
 'd': 'x',
 'e': 'l',
 'f': 'y',
 'g': 'q',
 'h': 'a',
 'i': 'b',
 'j': 's',
 'k': 'u',
 'l': 'i',
 'm': 'k',
 'n': 'r',
 'o': 'n',
 'p': 'j',
 'q': 'm',
 'r': 'z',
 's': 'o',
 't': 'h',
 'u': 'c',
 'v': 'g',
 'w': 'd',
 'x': 'f',
 'y': 'p',
 'z': 't'}
```

```
dic
✓ 0.6s
{'e': 'l',
 't': 'h',
 'a': 'v',
 'h': 'a',
 'n': 'r',
 'd': 'x',
 'o': 'n',
 's': 'o',
 ('j', 'k', 'q', 'x', 'z'): ['u', 'f', 'm', 's', 't'],
 ('b',
  'c',
  'f',
  'g',
  'i',
  'j',
  'k',
  'l',
  'm',
  'p',
  'q',
  'r',
  'u',
  'v',
  'w',
  'x',
  'y',
  'z'): ['b', 'c', 'd', 'e', 'g', 'i', 'j', 'k', 'p', 'q', 'w', 'y', 'z']}
```

As we see above, we have predicted the actual key to some extent.

Vigenere Cipher:

In this method, we have a password. We extrapolate the password so that it covers the whole input message. We now calculate the mapping by adding the weightage of the password letter to the original letter.

```
E.g., t h i s  i  s a  s a m  p l  e  t e x t      - (original text = ‘this is a sample text’)
      k i  r u k i  r  u k i  r u k i  r u k     - (password = ‘kiru’)
      D P Z M S A R M K U G F O B V R D        - (encrypted message)
```

Algorithm:

- First we rotate the encrypted message by varying lengths starting from 1,2,3 and so on which are C_1, C_2, C_3 …
- We compare these with the original encrypted message C_0. We count the number of collisions and record them. The C_n with high collisions corresponds to the password length of ‘n.’ This is because as we rotate by ‘n,’ which is the length of the password, we encrypt the letters at that position by the same letter in the password. This ensures a high possibility of collision. (demonstrated in a simple example below)

  E.g., d d d d d d d d d (original message)
  a b c a b c a b c (password = ‘abc’)
  C_0: d e f d e f d e f
  C_1: e f d e f d e f d (no of collisions = 0)
  C_2: f d e f d e f d e (no of collisions = 0)
  C_3: d e f d e f d e f (no of collisions = 9)

  As we can see in the above example we have maximum collisions when the rotation is done by ‘3’ which is the length of the password. Here we took an extreme example to demonstrate the property, but this is generally true for large texts in English.

- Based on the length of the password, we can now break down the encrypted message corresponding to each of the letters of the password, do frequency analysis and predict each letter of the password.

Functions:

- We first set a password. For the message to encrypt, we use the string obtained from the text_strip function defined above.
- Encryption: vigenere_encrypt(s,password)

  We loop through our string and obtain the corresponding password letter by doing password[i%length_of_password]. The value of the password letter is the weightage, and by adding it to the letters from the original string, we calculate the value of the resultant letter. This way, we encrypt the message.

- Decryption:

  We implement the ideas described above through the following functions:

  1. rotate_compare(s,r):

     Rotates the input string by r places and counts the number of collisions.
  2. cryptanalyse _vigenere_findlength(s):

     Using the rotate_compare function above, through many iterations, we find the values for which the rotation is giving us maximum collisions. We then take the GCD of these values, which will give us the length of the password.

  3. cryptanalyse_vigenere _afterlength(s,k):

     Using the above function, we break down the encrypted string into substrings corresponding to each letter of the password through a loop. Now we do frequency analysis for each of these substrings using the 'letter_ditribution' function defined above. Consequently, using the highest frequency letter, which will correspond to 'e' in every substring, we find out the weightage that was added, thereby finding out the letters of the password.

  4. vigenere_decrypt(s, password):

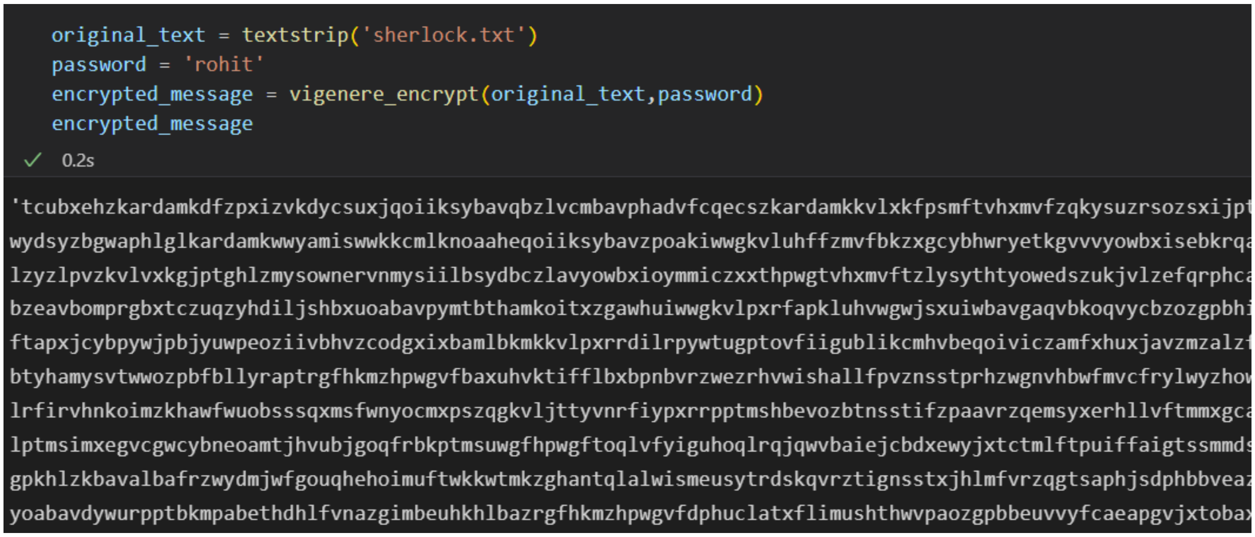
     Using the password found above, we subtract the value of each letter from the decrypted string by the value of the corresponding letter of the password. This will give us the decrypted string.

  5. cryptanalyse_vigenere(s):

     This is cumulative of all the above functions. Give a vigenere encrypted string; this will output decrypted string and the predicted password.

Results:

1. Vigenere encrypted message:

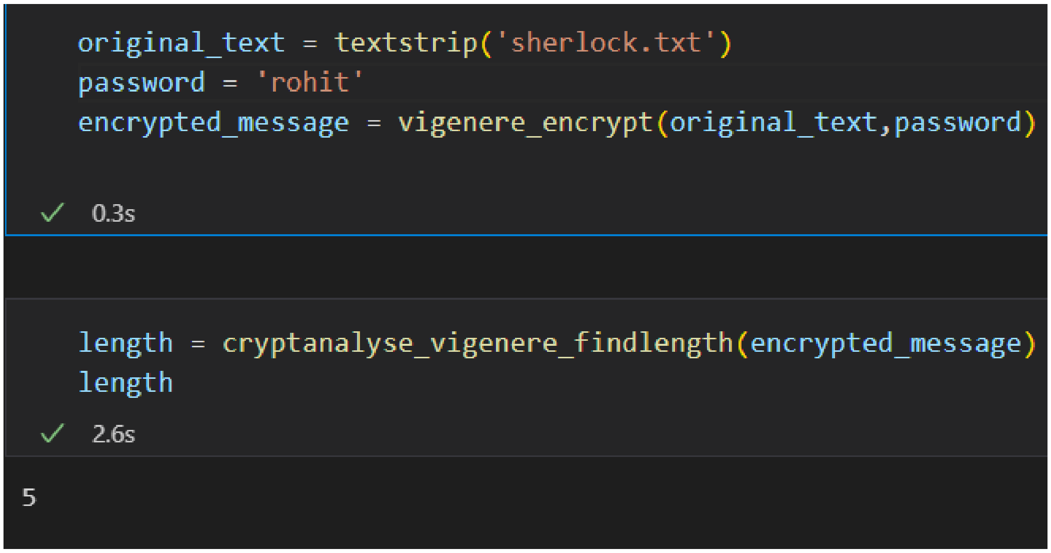


```
original_text = textstrip('sherlock.txt')
password = 'rohit'
encrypted_message = vigenere_encrypt(original_text,password)
encrypted_message
```

✓ 0.2s

```
'tcubxehzkardamkdfzpxizvkdycsuxjqoiiksybavqbzlvcmbavphadvfcqecszkardamkkvlxkfpsmftvhxmvfzqkysuzrsozsxijpt
wydsyzbgwaphlglkardamkwwyamiswwkkcmlknoaaheqoiiksybavzpoakiwwgkvluhffzmvfbkzxgcybhwryetkgvvvyowbxisebkrqa
lzyzlpvzkvlvxkgjptghlzmysownervnmysiilbsydbczlavyowbxioymmiczxxthpwgtvhxmvftzlysythtyowedszukjvlzefqrphca
bzeavbomprgbxtczuqzyhdiljshbxuoabavpymtbthamkoitxzgawhuiwwgkvlpxrfapkluhvwgwjsxuiwbavgaqvbkoqvycbzozgpbhi
ftapxjcybpywjpbjyuwpeoziivbhvzcodgxixbamlbkmkkvlpxrrdilrpywtugptovfiigublikcmhvbeqoiviczamfxhuxjavzmzalzf
btyhamysvtwwozpbfbllyraptrgfhkmzhpwgvfbaxuhvktifflbxbpnbvrzwezrhvwishallfpvznsstprhzwgnvhbwfmvcfrylwyzhow
lrfirvhnkoimzkhawfwuobsssqxmsfwnyocmxpszqgkvljttyvnrfiypxrrpptmshbevozbtnsstifzpaavrzqemsyxerhllvftmmxgca
lptmsimxegvcgwcybneoamtjhvubjgoqfrbkptmsuwgfhpwgftoqlvfyiguhoqlrqjqwvbaiejcbdxewyjxtctmlftpuiffaigtssmmds
gpkhlzkbavalbafrzwydmjwfgouqhehoimuftwkkwtmkzghantqlalwismeusytrdskqvrztignsstxjhlmfvrzqgtsaphjsdphbbveaz
yoabavdywurpptbkmpabethdhlfvnazgimbeuhkhlbazrgfhkmzhpwgvfdphuclatxflimushthwvpaozgpbbeuvvyfcaeapgvjxtobax
```

2. Predicting the length of the password:



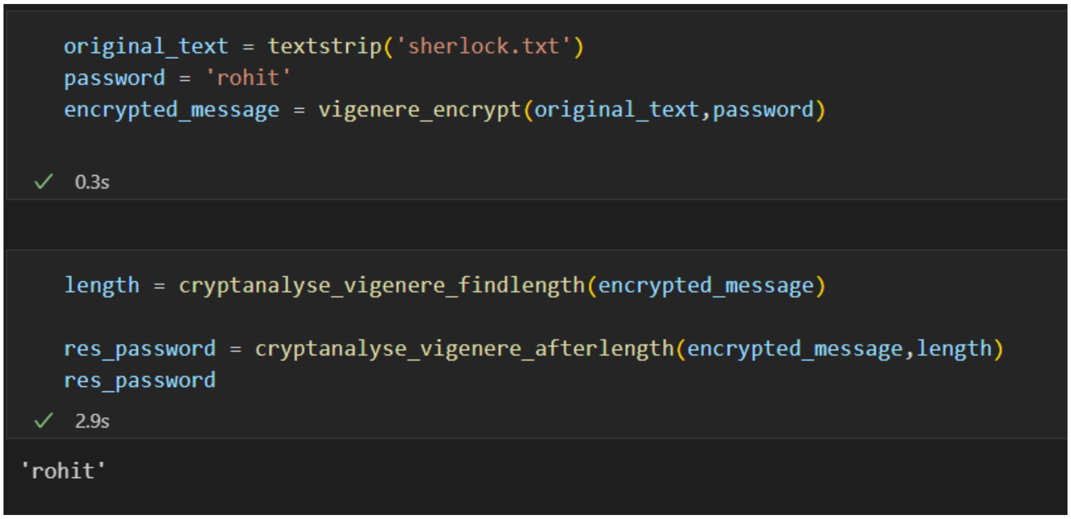
```
original_text = textstrip('sherlock.txt')
password = 'rohit'
encrypted_message = vigenere_encrypt(original_text,password)
```

```
length = cryptanalyse_vigenere_findlength(encrypted_message)
length
```

```
5
```

As we can see, the length of the password is being accurately predicted.

3. Predicting the password:



```
original_text = textstrip('sherlock.txt')
password = 'rohit'
encrypted_message = vigenere_encrypt(original_text,password)
```

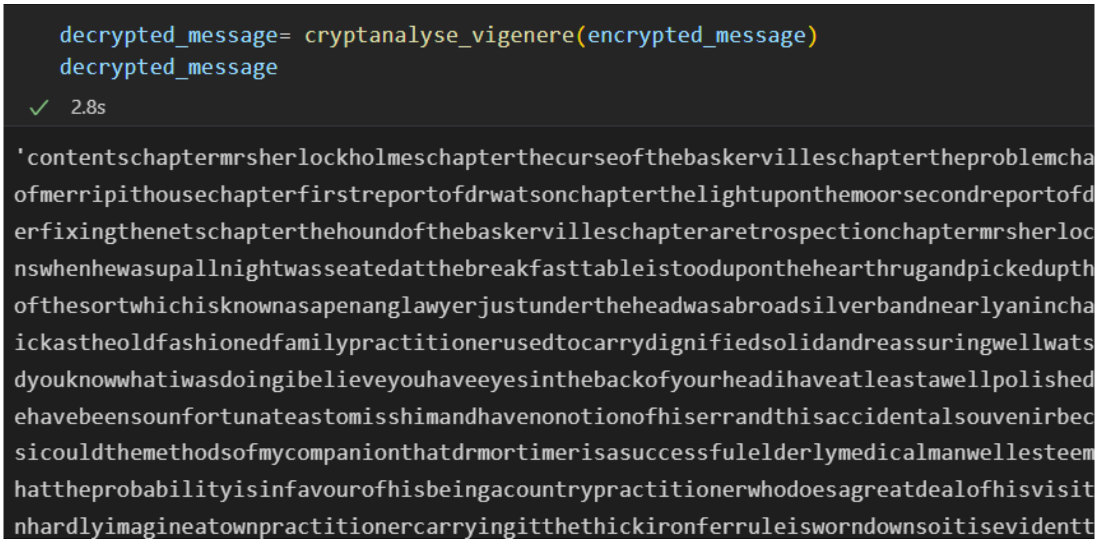
```
length = cryptanalyse_vigenere_findlength(encrypted_message)

res_password = cryptanalyse_vigenere_afterlength(encrypted_message,length)
res_password
```

```
'rohit'
```

We have accurately predicted the password as well.

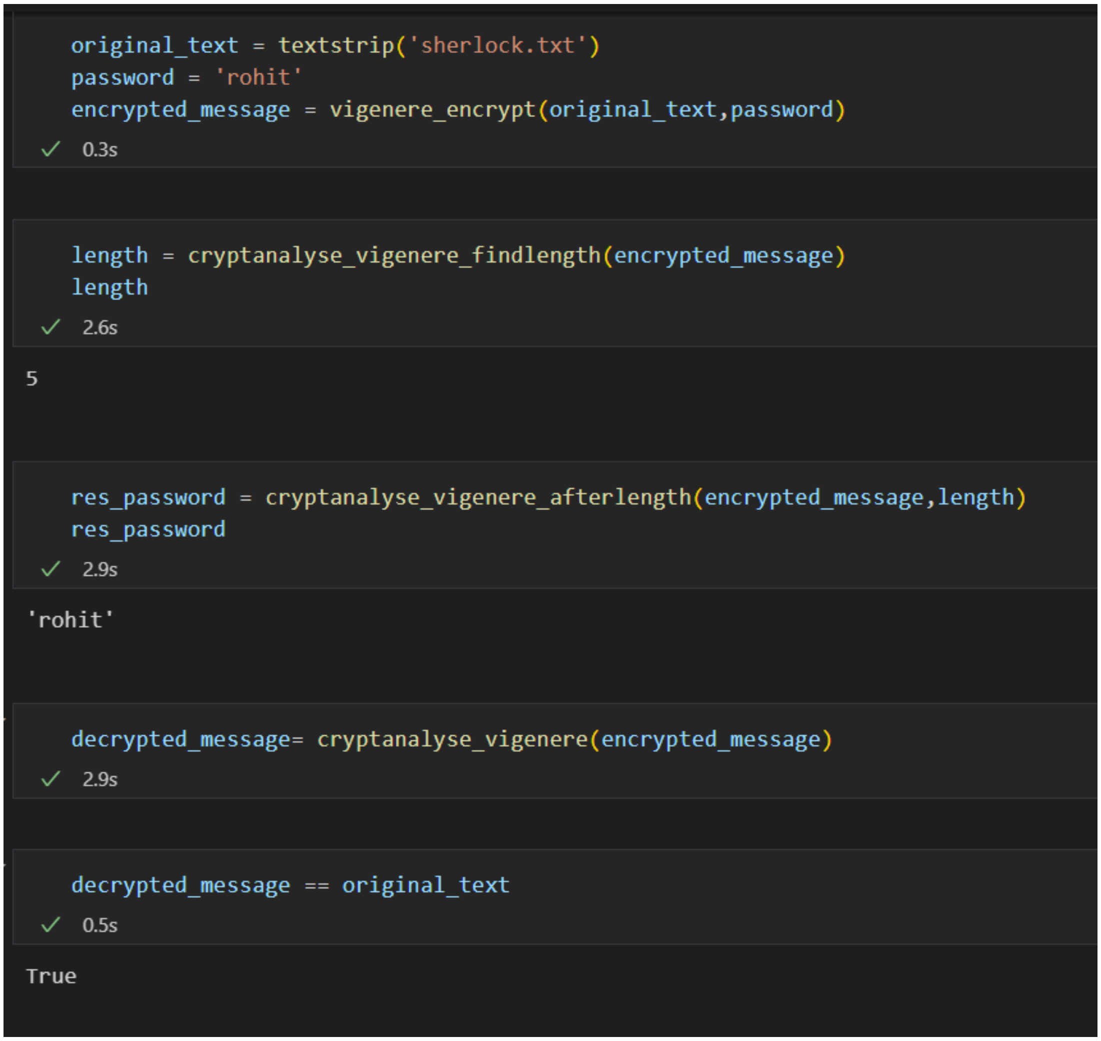
4. Decrypting the message:



```
decrypted_message= cryptanalyse_vigenere(encrypted_message)
decrypted_message
```

✓ 2.8s

```
'contentschaptermrsherlockholmeschapterthecurseofthebaskervilleschaptertheproblemcha
ofmerripithousechapterfirstreportofdrwatsonchapterthelightuponthemoorsecondreportofd
erfixingthenetschapterthehoundofthebaskervilleschapteraretrospectionchaptermrsherloc
nswhenhewasupallnightwasseatedatthebreakfasttableistooduponthehearthrugandpickedupth
ofthesortwhichisknownasapenanglawyerjustundertheheadwasabroadsilverbandnearlyanincha
ickastheoldfashionedfamilypractitionerusedtocarrydignifiedsolidandreassuringwellwats
dyouknowwhatiwasdoingibelieveyouhaveeyesinthebackofyourheadihaveatleastawellpolished
ehavebeensounfortunateastomisshimandhavenonotionofhiserrandthisaccidentalsouvenirbec
sicouldthemethodsofmycompanionthatdrmortimerisasuccessfulelderlymedicalmanwellesteem
hattheprobabilityisinfavourofhisbeingacountrypractitionerwhodoesagreatdealofhisvisit
nhardlyimagineatownpractitionercarryingitthethickironferruleisworndownsoitisevidentt
```



```
original_text = textstrip('sherlock.txt')
password = 'rohit'
encrypted_message = vigenere_encrypt(original_text,password)
```

✓ 0.3s

```
length = cryptanalyse_vigenere_findlength(encrypted_message)
length
```

✓ 2.6s

```
5
```

```
res_password = cryptanalyse_vigenere_afterlength(encrypted_message,length)
res_password
```

✓ 2.9s

```
'rohit'
```

```
decrypted_message= cryptanalyse_vigenere(encrypted_message)
```

✓ 2.9s

```
decrypted_message == original_text
```

✓ 0.5s

```
True
```

References:

1. Singh, Simon. *The Code Book*. Delacorte Press, 2003.
2. Amitashr. *Vigenere Cipher*, 28 July 2011, http://algorithinking.blogspot.com/2011/07/vigenere-cipher.html?m=1.