ECE 458 Assignment 1: Symmetric key encryption

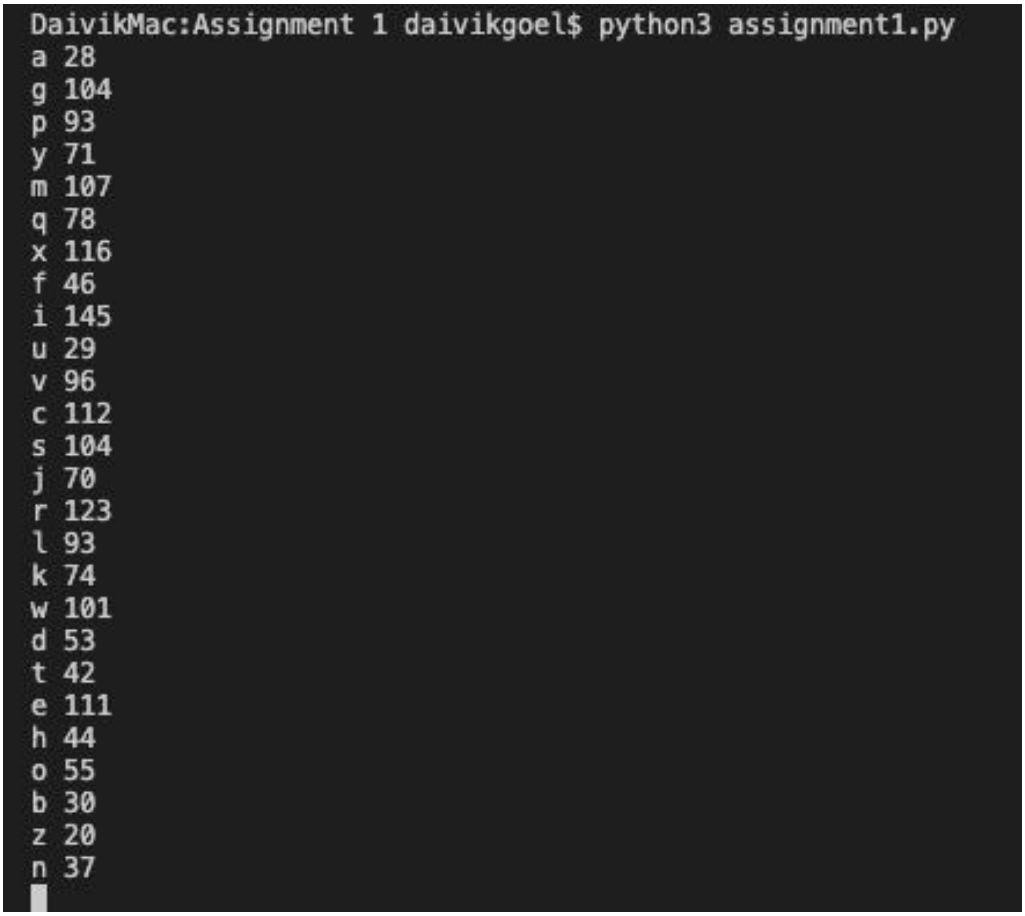
Exercise 2: Non Coding Part (30 marks)

1. **[15 marks]** Describe a method that you would have used to obtain the key length if you did not have the length (Only one method with a valid justification is enough).

Note: You do not have to actually perform the method on the given ciphertext, you only need to explain the method on any given ciphertext; however, you may use the given ciphertext as an example if that helps with your explanation.

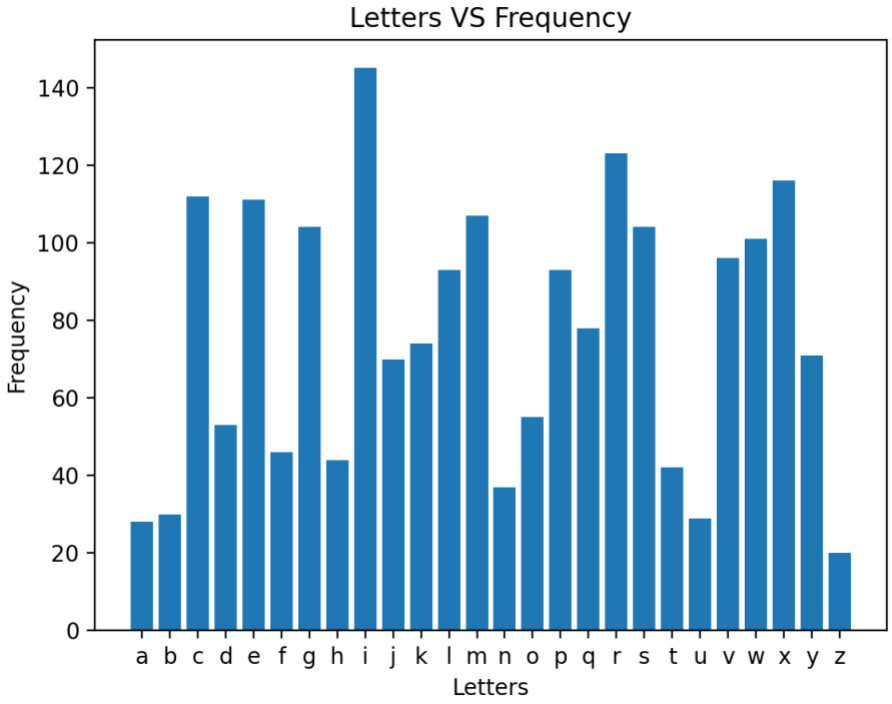
2. **[15 marks]** Describe the steps one need to take in order to crack a ciphertext encrypted using Vigenere Cipher. You may use your comments, parts of code or graphs in Exercise 1.

1A: output from frequency analysis of each letter in the ciphertext.



1B:

From the letters' frequency graph, we can see that the letter 'i' has occurred the most amount of times, followed by letters 'r' and 'x' in the ciphertext we were given. Compared to letters in the English language, the letter 'e' is the most common, followed by 'a' and 'r'. From the letter frequency comparison alone between the given cipher text and the English language, not much info can be revealed, with the exception of the correlation between English letter 'e' and the ciphertext's most frequent letter 'i'.

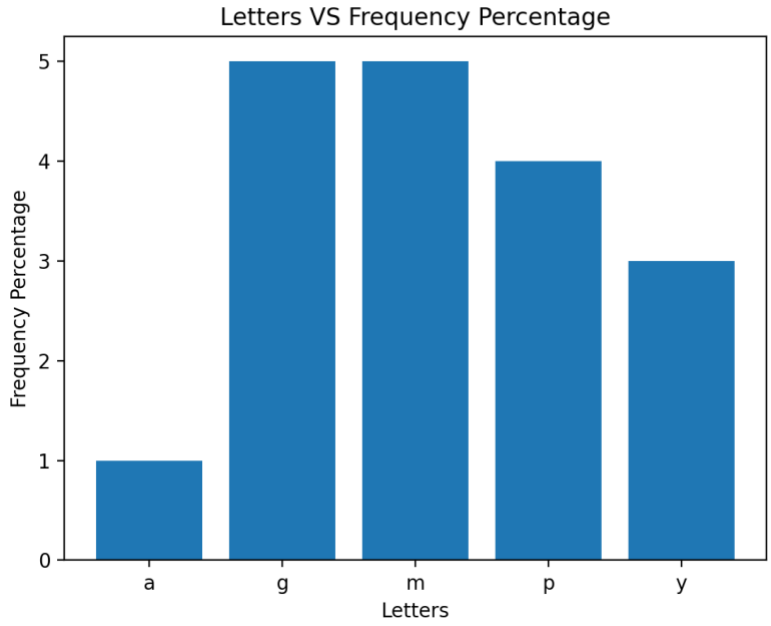


2A: output of list of chunked text of size n = 6:

```
DaivikMac:Assignment 1 daivikgoel$ python3 assignment1.py
CHUNKS:  ['agpygm', 'qgxyxf', 'iuimyp', 'mvcmss', 'vuiyjc', 'ggxrip', 'iuxyjr', 'lkwdiv', 'xkwtyq', 'xtexhm', 'quxejd', 'jtswxf', 'ikrdip', 'rgxdlc', 'gqpyvm', 'jcrsqy', 'pumcfw', 'rqqoel', 'wcqkxr', 'itspgf', 'epgomr', 'hgtorb', 'wqrwel', 'cesxwg', 'hgvkxg', 'spwlyr', 'mpxrik', 'elsbmr', 'cqjmeq', 'iuxorb', 'wvszvm', 'xggdxf', 'icrsqy', 'phvyqb', 'epkovz', 'ctixhc', 'vkrqmr', 'pgwcgm', 'rutsgs', 'swwzip', 'lctcmr', 'qccliq', 'ekhdly', 'xkjmsj', 'stmxkg', 'woesrj', 'crvyxc', 'gvmfir', 'lgvoss', 'kjxdsz', 'idydjc', 'advskf', 'xncmsj', 'stinel', 'moevwr', 'lgvoep', 'ijsgit', 'itryxy', 'jgameq', 'iumxaf', 'melfmt', 'mfgypm', 'vuebir', 'lgqcij', 'zgwzvm', 'xggdmt', 'ivloog', 'rijswf', 'itmdwc', 'phxrss', 'kjwyfp', 'mildpw', 'gqpyvc', 'hkwlcl', 'soikrq', 'icwixm', 'wgidlc', 'fnyoly', 'vosxmx', 'iuasxf', 'xjiger', 'itexhr', 'lgfsvb', 'eumdhy', 'vvwkpm', 'rixriq', 'xtikqj', 'sqocej', 'qqwdpg', 'ogeppy', 'wjspws', 'rnmqlr', 'hgwovr', 'epmwej', 'wcvokc', 'rgvkpj', 'cvlogm', 'pqvyjr', 'lghowc', 'vvxryq', 'jqvsrq', 'xcrmir', 'lgpssl', 'xjikrr', 'insziy', 'rfxriu', 'mnhnsl', 'ogckvc', 'enpcel', 'hesvsp', 'ifmxhc', 'ifwkcq', 'gcryrr', 'vkwdvy', 'qkrdlc', 'hgwovr', 'ajibil', 'ikxrip', 'xtiowz', 'vwwram', 'sfryvc', 'zgrerb', 'ynedmm', 'rqjdlc', 'wwvpea', 'icjpsp', 'hvlowj', 'mildiq', 'xrvyxc', 'gvmyrr', 'skxcjm', 'iuewsb', 'mhmmer', 'mqryja', 'snsbeq', 'wkqspy', 'xghdsr', 'lcxyjr', 'lgwevp', 'swrnml', 'keserr', 'vamcez', 'wqpexc', 'parogc', 'wuebcf', 'ipgoag', 'xjsexc', 'beizxg', 'spxris', 'tribtj', 'yoeqim', 'jgzovw', 'fkvnel', 'hcpcsr', 'lgjevm', 'jcpvxf', 'iuqkpj', 'itqkqk', 'enwkrb', 'xjicog', 'rqjkpj', 'xjicry', 'ogwkrb', 'pkdkvb', 'wkwyjm', 'rgyxmd', 'stqcel', 'hesvsp', 'xjixiv', 'xrssrr', 'muxria', 'snsbsd', 'xjiwer', 'ytimer', 'ittspj', 'etwcsk', 'iqjglg', 'gjebiz', 'vqaxxf', 'mutbsz', 'edpiqy', 'ogwdlc', 'gcxovn', 'mnpkvc', 'zgrwsp', 'iesxwn', 'meyyyq', 'eosxkr', 'lgkbic', 'rnikzc', 'wvlkru', 'swpnsr', 'lgvgmq', 'ididlc', 'gcwopc', 'xwwcic', 'xjixaf', 'ivlovr', 'lglkfg', 'xuspxf', 'ikrcia', 'xymvpr', 'ltsgel', 'cnmqlr', 'yrsxxf', 'itmnhj', 'iylkxu', 'swpncm', 'yfssjw', 'swaovc', 'edmqgy', 'xgvzmj', 'pcvglw', 'pkooqm', 'wvsdlc', 'vfipil', 'wgpowq', 'gtikxs', 'vgwiss', 'aqyvhd', 'ighlcl', 'mildel', 'hnmogm', 'reikpc', 'hdcnew', 'wqhyxf', 'iuimer', 'ittspj', 'etwglc', 'rvloqm', 'vpmxkj', 'mildgm', 'qgwdlc', 'cevoin', 'hqaxxf', 'iuxoqm', 'jvlojm', 'sftvel', 'xcrnpg', 'iesxgc', 'eninek', 'spkdlc', 'xjmmof', 'itfkkc', 'ephnvw', 'wvmmoq', 'ephviy', 'zgwxiy', 'vvlokp', 'swrnel', 'hkxswm', 'fxmyyq', 'xjedyl', 'hgvcya', 'lembgs', 'quxkra', 'iuxriz', 'vqaxgm', 'pqvbiy', 'pnclia', 'soicen', 'vqxogr', 'mqrsxk', 'mildml', 'hginfc', 'etkeib', 'xjedxf', 'ieedip', 'tkpvep', 'wjefml', 'kdimsk', 'idvyal', 'gqrmiy', 'pghdlc', 'quivzc', 'wqrdlc', 'ktserb', 'ephdly', 'xyigip', 'itifip', 'wkrqxf', 'iuxkxc', 'shxrml', 'kufexr', 'lkwswl', 'svwyfc', 'gcyciu', 'lkpoac', 'qcccew', 'eueqil', 'itevvs', 'pgxrer', 'pcvqia', 'evibtg', 'pnebwd', 'ighlcl', 'mildel', 'hnmogm', 'reikpc', 'hdcnew', 'mvmcfw', 'rqqoel', 'wcpgew', 'wvlogy', 'wgxrer', 'xjiiep', 'idvyal', 'wqqosd', 'xjiwwr', 'mnpbir', 'ekrsre', 'xjiqvc', 'ipgypm', 'vyiwew', 'xjixgm', 'repehc', 'xjedxf', 'ijelmr', 'shgyra', 'icpsre', 'xjiwwc', 'pxicfw', 'hccmek', 'ihmbwr', 'ephdly', 'xvlofp', 'syrmsj', 'stmcej', 'evibeb', 'erxkxg', 'sp']
```

2B

Below is the graph for letters VS frequency of the first chunk of 6 letters.



1. Without the key length given, we would have to guess the length *n,* and then divided the message into *n* parts. Essentially brute force thru the entire cipher trying out different length of *n* each time. At each iteration, compare the statistically common letters of our chunk to help in guessing parts of the cipher. The idea of frequency analysis when n is unknown is to check the percent occurrence of the cipher text's most frequent letters in descending order, at each letter, we compare the frequency of occurrence with that of the English language. Using this method, we can estimate the key size by looking at the frequency patterns. With the patterns becoming apparent, we shift every letter in the chunk by the same amount to match the result to the expected frequency distribution of the English language. In our figure 1B, the letter 'i' occurred the most out of the entire cipher text, if we try to match that with the English letter 'e', this suggests that particular letter has been shifted 5 times, as 'i' – 'e' = 5. The next step is to take subsequent chunks and apply the same shift of 5 at that position. For example,

   original text: h **e** l l **o** w o **r** l d
   Cipher text: h **i** m l **s** x o **v** m d
   key: a e b

If we suspect that the key is of length 3, and have found that 'i' corresponds to 'e' (a shift of 5 positions), we will try to decrypt the letter 's' and 'v' by shifting them by 5 positions as they would also be encrypted by the 2nd key letter of 'e'.

A slight improvement to cracking the Vigenère cipher without knowing the key length would be to start with the Kasiski method. This method takes advantage of that every n'th letter is encrypted using the

same shift. Kasiski method takes advantage of that and seeks to find multiple repeated character combinations in the cipher.

For instance, if substring pattern 1 occurred 30 letters apart suggesting a key length of 30, 15, 10, 6,5,3, and 2. Substring pattern 2 occurred 18 letters apart, suggesting a key length of 18, 9, 6,3, and 2.

Kasiski's method tells us that the key length is likely a common multiple of 30 and 18, so we would try a key length of 6 and possibly 3. Thus, Kasiski's method uses cipher substring repetitions to cut down the amount of brute force needed to decrypt the cipher.

One caveat to decrypting Vigenère's cipher is we are doing so with the assumption of:

A. having a cipher text that is long enough for us to perform frequency analysis.
B. having a key length that is much smaller than the cipher text length.

When condition A is not met, frequency analysis becomes nearly impossible. Likewise, when condition B is not met, meaning when the key length is greater than or equal to the cipher text length, Vigenère's cipher becomes a one-time pad(OTP) mechanism where Pr{P | C} = Pr{P}, achieving perfect secrecy, making it impossible to crack.

2. In order to crack the CipherText the first step is to try finding the length of the secret key. As described above there are various ways of going about this.

   Once the key length is found we want to break the cipher text into chunks. As seen in our code below we break up the cipher text using n (cipher text key length) as the dividing factor.

```
# part 2A:
def chunk(cipher, n):
    #list of dictionaries for letters of each position.
    dictionarylist = []

    for x in range(n):
        dictionarylist.append({})

    #splits ciphertext into chunks of length n each
    chunks = [cipher[i:i+n] for i in range(0, len(cipher),n)]
```

   Now from here we need to find out at each position (from 0:n) for all the chunks what letters occur the most frequently. To prepare for this we first set up a list of dictionaries. This list will hold multiple dictionaries that contain the frequency of the letters for their respective position. We then go through every chunk and count the letter per position

```
    for chunk in chunks:
        #update freq of char appearence in dictionary at each position relative to chunk
```

```
        for x in range(len(chunk)):
            if chunk[x] in dictionarylist[x]:
                dictionarylist[x][chunk[x]] += 1
            else:
                dictionarylist[x][chunk[x]] = 1
    print("CHUNKS: " , chunks)
    return chunks, dictionarylist
```

Now that we have the frequencies laid out for each position we should now find out what the most common letter was at each position. For each of these letters we then compare its position relative to the letter “e.” The reason why we do this is because “e” is the most common letter in the English language. We thus take the assumption that the most common letter of the position is the shifted version of “e”. In doing so we can compare the difference and find the shift for each of the letters in a certain position. We save the shift in an array and move onto actually getting the secret key. The shift for each position corresponds directly to the secret key so all we need to do is look at the value of the shift at each position and determine the alphabetic value. The code below is doing exactly this. (Note: We account for the wraparound effect of the letters in the code)

```
#question 3
def get_secret_key(dictionarylist):
    shift = []
    #e is the most common letter in the dictionary. We take the most common letter at each positon and assume it to
be e. We try finding the shift amount accordingly
    for i in dictionarylist:
        i = sorted(i.items(), key=lambda tup: tup[1], reverse=True)
        #Adding shift amount of each position to an array
        shift.append(ord(i[0][0]) - ord('e') )
        print(i)
        print(shift)

    secretkey = ''
    for lettershift in shift:
        #using the shift amount to get the secret key.
        if lettershift > 0:
            #the secret key is the position of each letter relative to a
            secretkey += chr(lettershift + 97)
        else:
            #this is for the wrap around case ie. a - 2 = y
            secretkey += chr(26 + lettershift + 97)
```

```
    print(secretkey)
    return secretkey, shift
```

In this case we got the secret key to be ecekey.

Now since we know the shift at each position all we need to do is go through our chunks and shift the letters back to what they were originally in each chunk and link the chunks together. The code below does this while keeping the wrap around effect into consideration

```
def decrypttext(shift, chunks):
    #after knowing the shift we can find out the plain text
    plaintext =''
    for chunk in chunks:
        for x in range(len(chunk)):
            #we take for each chunk and shift according to what we found the shift to be
            charno = ord(chunk[x]) - shift[x]
            #covering various cases like wrap around
            if charno > 122:
                plaintext += chr((charno - 123)+ 97)
            elif charno > 96:
                plaintext += chr(charno)
            else:
                plaintext += chr(charno + 26)
    print("PLAIN TEXT:" , plaintext)
```

After this is done we have our plain text! In our case it was:

welocometothesecuritycourseofecetherestofthistextisjustrandomstufffromtheinternetthecolorofanimalsisbynomeansamatterofchanceitdepends onmanyconsiderationsbutinthemajorityofcasestendstoprotecttheanimalfromdangerbyrenderingitlessconspicuousperhapsitmaybesaidthatifcolor ingismainlyprotectivethereoughttobebutfewbrightlycoloredanimalstherearehowevernotafewcasesinwhichvividcolorsarethemselvesprotectivethe kingfisheritselfthoughsobrightlycoloredisbynomeanseasytoseetheblueharmonizeswiththewaterandthebirdasitdartsalongthestreamlooksalmostli keaflashofsunlightdesertanimalsaregenerallythecolorofthedesertthusforinstancethelionintheantelopeandthewilddonkeyareallsandcoloredindeeds ayscanontristraminthedesertwhereneithertreesbrushwoodnorevenundulationofthesurfaceaffordtheslightestprotectiontoitsfoesamodificationofc olorassimilatedtothatofthesurroundingcountryisabsolutelynecessaryhencewithoutexceptiontheupperplumageofeverybirdandalsothefurofallthes mallermammalsandtheskinofallthesnakesandlizardsisofoneuniformsandcolorthenextpointisthecolorofthematurecaterpillarssomeofwhicharebro wnthisprobablymakesthecaterpillarevenmoreconspicuousamongthegreenleavesthanwouldotherwisebethecaseletusseethenwhetherthehabitsoft heinsectwillthrowanylightupontheriddlewhatwouldyoudoifyouwereabigcaterpillarwhylikemostotherdefenselesscreaturesyouwouldfeedbynightand lieconcealedbydaysodothesecaterpillarswhenthemorninglightcomestheycreepdownthestemofthefoodplantandlieconcealedamongthethickher bageanddrysticksandleavesnearthegroundanditisobviousthatundersuchcircumstancesthebrowncolorreallybecomesaprotectionitmightindeedbea rguedthatthecaterpillarshavingbecomebrownconcealedthemselvesonthegroundandthatwewerereversingthestateofthingsbutthisisnotsobecause whilewemaysayasageneralrulethatlargecaterpillarsfeedbynightandlieconcealedbydayitisbynomeansalwaysthecasethattheyarebrownsomeofthe mstillretainingthegreencolorwemaythenconcludethatthehabitofconcealingthemselvesbydaycamefirstandthatthebrowncolorisalateradaptation

This is how we went about deciphering the cipher text!