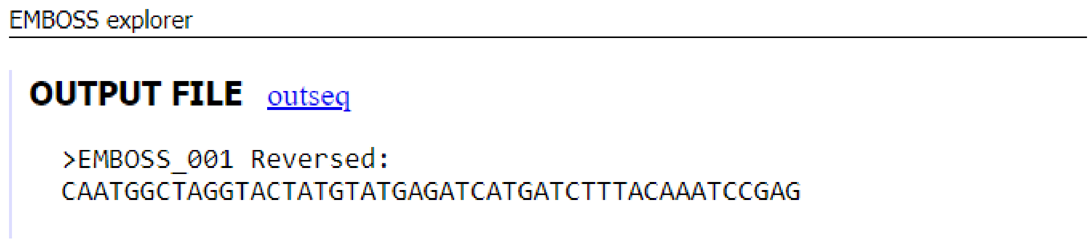
BIOINFORMATICS

PRACTICAL – 1

1. Installed Emboss in Linux using Command line: sudo apt-get install jemboss.
2. Use REVSEQ to find complementary strand

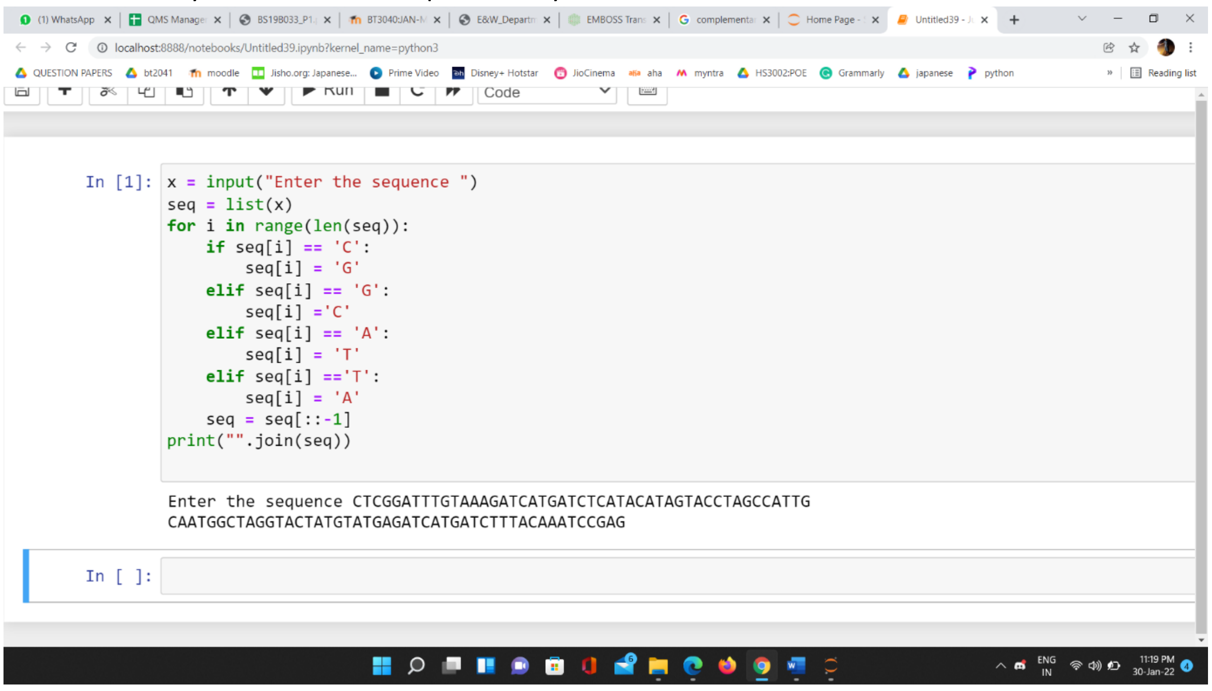


EMBOSS explorer

OUTPUT FILE outseq

```
>EMBOSS_001 Reversed:
CAATGGCTAGGTACTATGTATGAGATCATGATCTTTACAAATCCGAG
```

3. Screenshot of Python Code for complementary strand

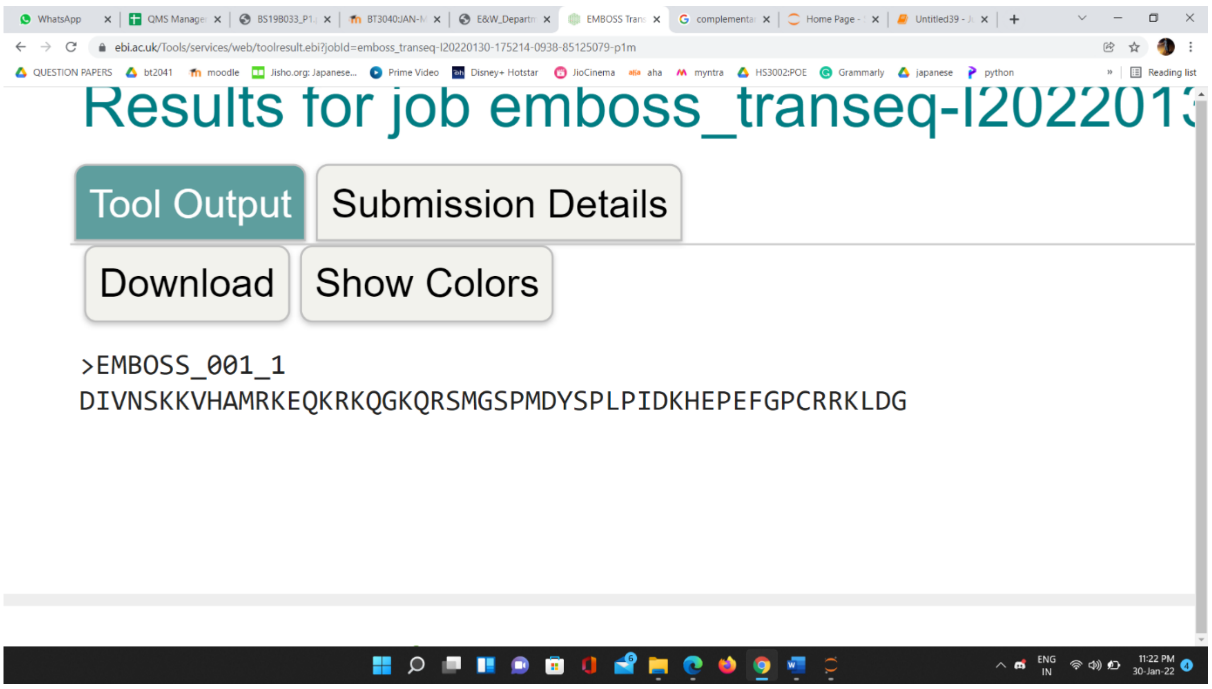


localhost:8888/notebooks/Untitled39.ipynb?kernel_name=python3

```
In [1]: x = input("Enter the sequence ")
        seq = list(x)
        for i in range(len(seq)):
            if seq[i] == 'C':
                seq[i] = 'G'
            elif seq[i] == 'G':
                seq[i] ='C'
            elif seq[i] == 'A':
                seq[i] = 'T'
            elif seq[i] =='T':
                seq[i] = 'A'
            seq = seq[::-1]
        print("".join(seq))

        Enter the sequence CTCGGATTTGTAAAGATCATGATCTCATACATAGTACCTAGCCATTG
        CAATGGCTAGGTACTATGTATGAGATCATGATCTTTACAAATCCGAG

In [ ]:
```

4. Protein Sequence

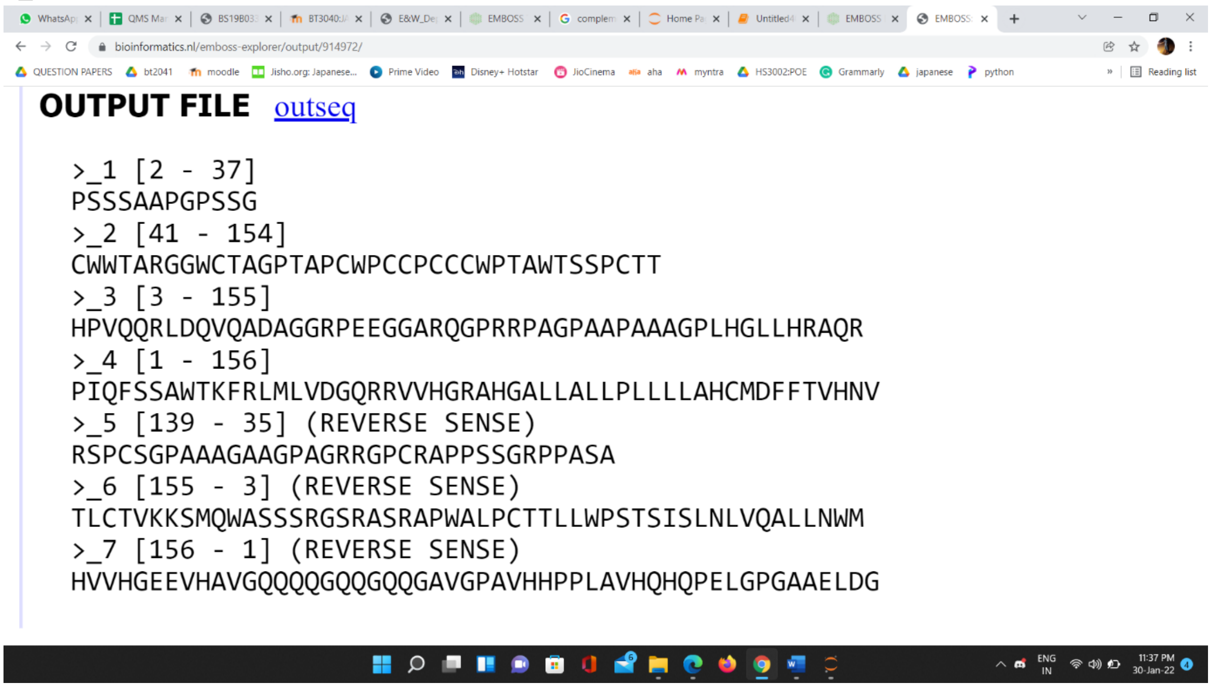
   (i) protein sequence using Emboss


ebi.ac.uk/Tools/services/web/toolresult.ebi?jobId=emboss_transeq-I20220130-175214-0938-85125079-p1m
Results for job emboss_transeq-I2022013
Tool Output
Submission Details
Download
Show Colors

```
>EMBOSS_001_1
DIVNSKKVHAMRKEQKRKQGKQRSMGSPMDYSPLPIDKHEPEFGPCRRKLDG
```

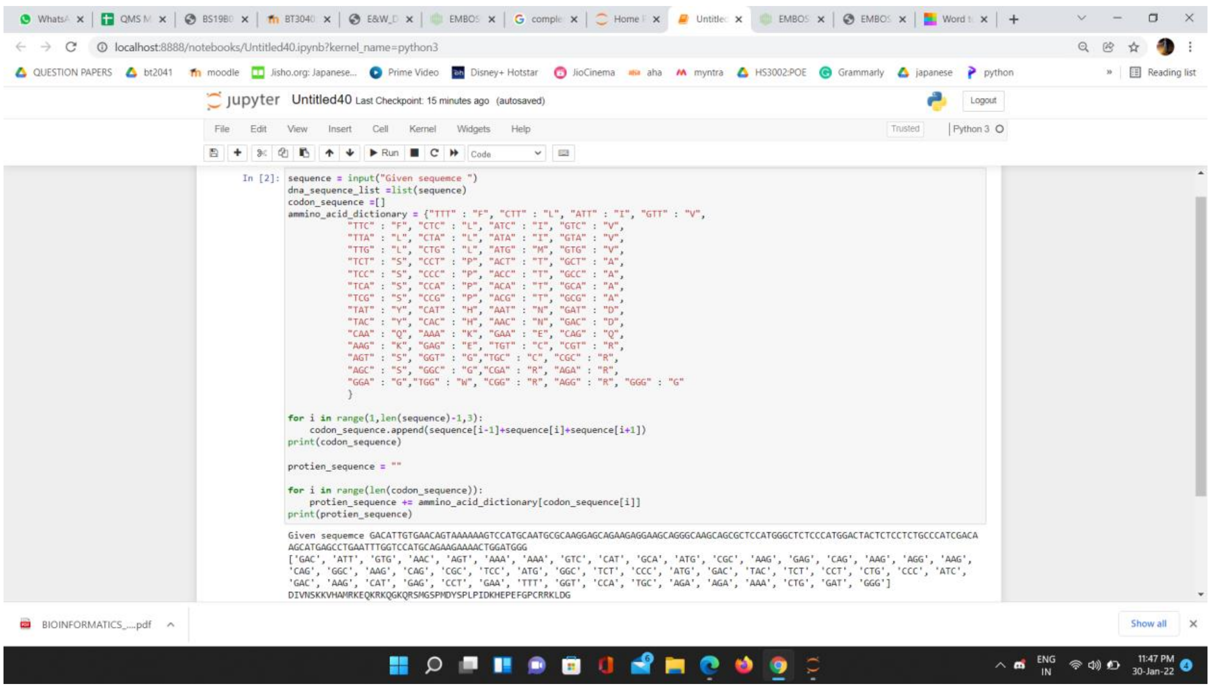
(ii) DNA sequence for given protein sequence

>_4



```
>_1 [2 - 37]
PSSSAAPGPSSG
>_2 [41 - 154]
CWWTARGGWCTAGPTAPCWPCCPCCCWPTAWTSSPCTT
>_3 [3 - 155]
HPVQQRLDQVQADAGGRPEEGGARQGPRRPAGPAAPAAAGPLHGLLHRAQR
>_4 [1 - 156]
PIQFSSAWTKFRLMLVDGQRRVVHGRAHGALLALLPLLLLAHCMDFFTVHNV
>_5 [139 - 35] (REVERSE SENSE)
RSPCSGPAAAGAAGPAGRRGPCRAPPSSGRPPASA
>_6 [155 - 3] (REVERSE SENSE)
TLCTVKKSMQWASSSRGSRASRAPWALPCTTLLWPSTSISLNLVQALLNWM
>_7 [156 - 1] (REVERSE SENSE)
HVVHGEEVHAVGQQQQGQQGQQGAVGPAVHHPPLAVHQHQPELGPGAAELDG
```

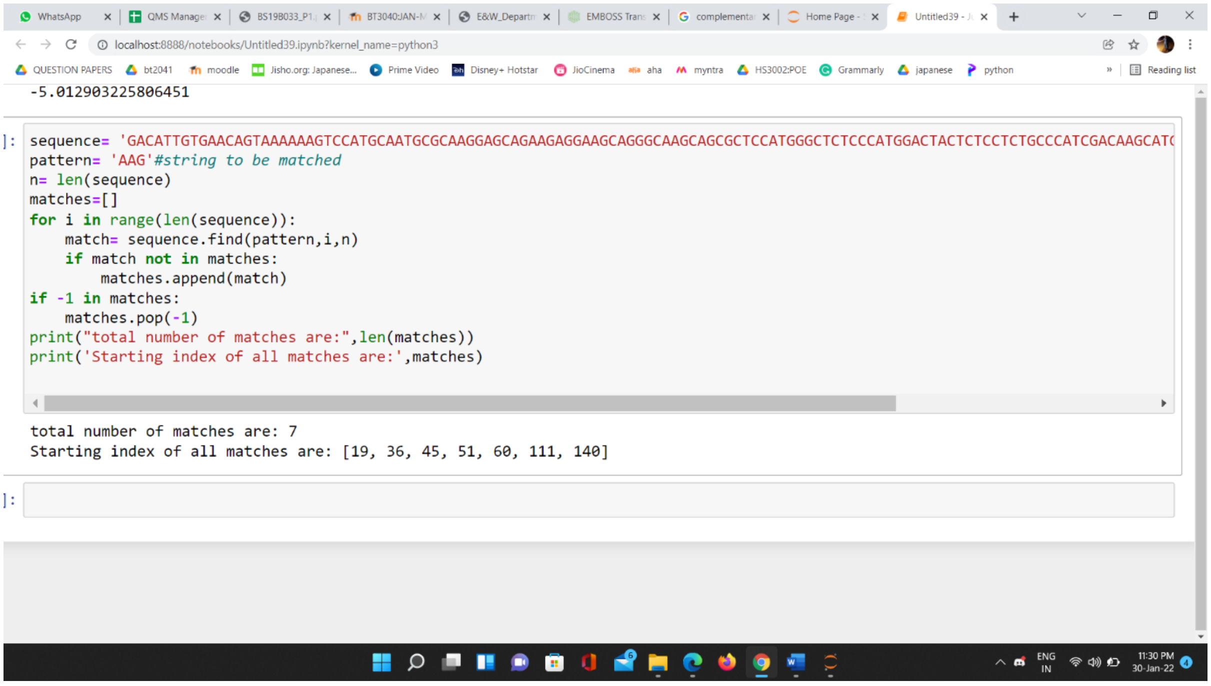
5. Code to find Protein sequence for given DNA Sequence



```
In [2]: sequence = input("Given sequemce ")
        dna_sequence_list =list(sequence)
        codon_sequence =[]
        ammino_acid_dictionary = {"TTT" : "F", "CTT" : "L", "ATT" : "I", "GTT" : "V",
                "TTC" : "F", "CTC" : "L", "ATC" : "I", "GTC" : "V",
                "TTA" : "L", "CTA" : "L", "ATA" : "I", "GTA" : "V",
                "TTG" : "L", "CTG" : "L", "ATG" : "M", "GTG" : "V",
                "TCT" : "S", "CCT" : "P", "ACT" : "T", "GCT" : "A",
                "TCC" : "S", "CCC" : "P", "ACC" : "T", "GCC" : "A",
                "TCA" : "S", "CCA" : "P", "ACA" : "T", "GCA" : "A",
                "TCG" : "S", "CCG" : "P", "ACG" : "T", "GCG" : "A",
                "TAT" : "Y", "CAT" : "H", "AAT" : "N", "GAT" : "D",
                "TAC" : "Y", "CAC" : "H", "AAC" : "N", "GAC" : "D",
                "CAA" : "Q", "AAA" : "K", "GAA" : "E", "CAG" : "Q",
                "AAG" : "K", "GAG" : "E", "TGT" : "C", "CGT" : "R",
                "AGT" : "S", "GGT" : "G","TGC" : "C", "CGC" : "R",
                "AGC" : "S", "GGC" : "G","CGA" : "R", "AGA" : "R",
                "GGA" : "G","TGG" : "W", "CGG" : "R", "AGG" : "R", "GGG" : "G"
                }

        for i in range(1,len(sequence)-1,3):
            codon_sequence.append(sequence[i-1]+sequence[i]+sequence[i+1])
        print(codon_sequence)

        protien_sequence = ""

        for i in range(len(codon_sequence)):
            protien_sequence += ammino_acid_dictionary[codon_sequence[i]]
        print(protien_sequence)
```

```
Given sequemce GACATTGTGAACAGTAAAAAAGTCCATGCAATGCGCAAGGAGCAGAAGAGGAAGCAGGGCAAGCAGCGCTCCATGGGCTCTCCCATGGACTACTCTCCTCTGCCCATCGACA
AGCATGAGCCTGAATTTGGTCCATGCAGAAGAAAACTGGATGGG
['GAC', 'ATT', 'GTG', 'AAC', 'AGT', 'AAA', 'AAA', 'GTC', 'CAT', 'GCA', 'ATG', 'CGC', 'AAG', 'GAG', 'CAG', 'AAG', 'AGG', 'AAG',
'CAG', 'GGC', 'AAG', 'CAG', 'CGC', 'TCC', 'ATG', 'GGC', 'TCT', 'CCC', 'ATG', 'GAC', 'TAC', 'TCT', 'CCT', 'CTG', 'CCC', 'ATC',
'GAC', 'AAG', 'CAT', 'GAG', 'CCT', 'GAA', 'TTT', 'GGT', 'CCA', 'TGC', 'AGA', 'AGA', 'AAA', 'CTG', 'GAT', 'GGG']
DIVNSKKVHAMRKEQKRKQGKQRSMGSPMDYSPLPIDKHEPEFGPCRRKLDG
```

6. Code for finding the pattern. Position of match with indexing starting from 1



```
-5.012903225806451
```

```
sequence= 'GACATTGTGAACAGTAAAAAAGTCCATGCAATGCGCAAGGAGCAGAAGAGGAAGCAGGGCAAGCAGCGCTCCATGGGCTCTCCCATGGACTACTCTCCTCTGCCCATCGACAAGCATC
pattern= 'AAG'#string to be matched
n= len(sequence)
matches=[]
for i in range(len(sequence)):
    match= sequence.find(pattern,i,n)
    if match not in matches:
        matches.append(match)
if -1 in matches:
    matches.pop(-1)
print("total number of matches are:",len(matches))
print('Starting index of all matches are:',matches)
```

```
total number of matches are: 7
Starting index of all matches are: [19, 36, 45, 51, 60, 111, 140]
```

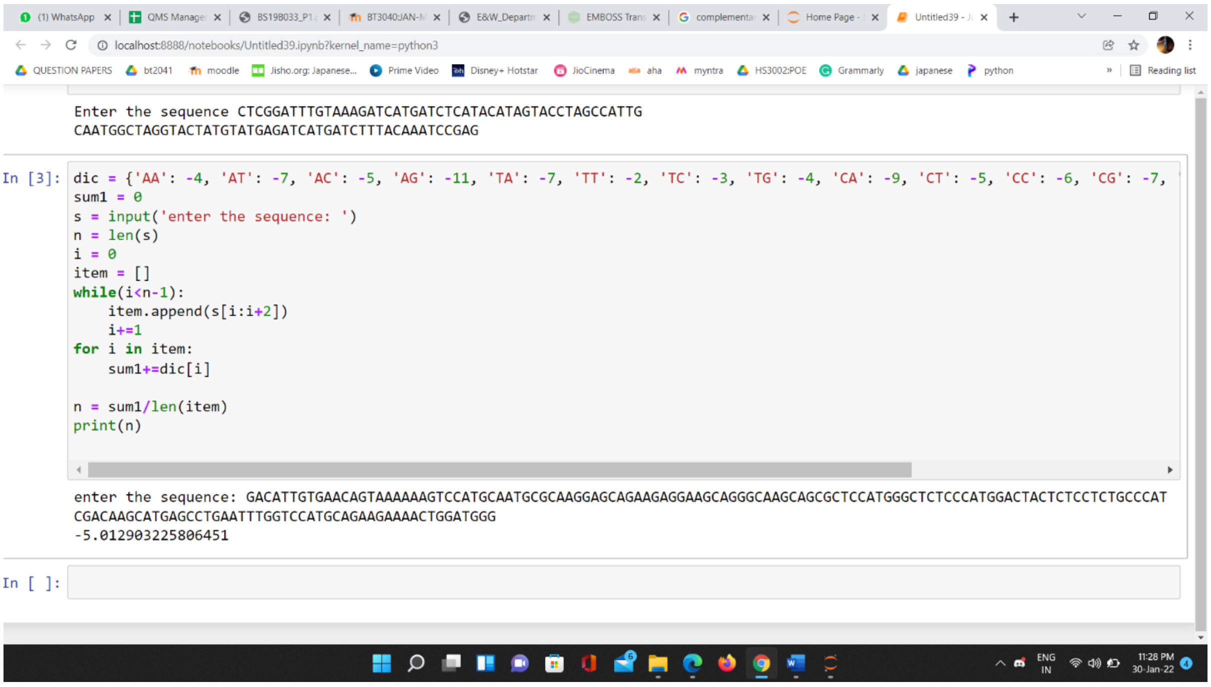
7. Features of Emboss:

Dan: used for calculating the melting temperature of a nucleic acid.

density: used to draw density plot of a nucleic acid.

remap: used to display the binding sites in each nucleotide sequence.

8. Code for calculation of Average Base Stacking Energy:



```
Enter the sequence CTCGGATTTGTAAAGATCATGATCTCATACATAGTACCTAGCCATTG
CAATGGCTAGGTACTATGTATGAGATCATGATCTTTACAAATCCGAG
```

```
In [3]: dic = {'AA': -4, 'AT': -7, 'AC': -5, 'AG': -11, 'TA': -7, 'TT': -2, 'TC': -3, 'TG': -4, 'CA': -9, 'CT': -5, 'CC': -6, 'CG': -7, '
        sum1 = 0
        s = input('enter the sequence: ')
        n = len(s)
        i = 0
        item = []
        while(i<n-1):
            item.append(s[i:i+2])
            i+=1
        for i in item:
            sum1+=dic[i]

        n = sum1/len(item)
        print(n)

        enter the sequence: GACATTGTGAACAGTAAAAAAGTCCATGCAATGCGCAAGGAGCAGAAGAGGAAGCAGGGCAAGCAGCGCTCCATGGGCTCTCCCATGGACTACTCTCCTCTGCCCAT
        CGACAAGCATGAGCCTGAATTTGGTCCATGCAGAAGAAAACTGGATGGG
        -5.012903225806451

In [ ]:
```

9.

(i) For sequence ATATATATA:
Average Melting Temperature for ATATATATAT: 48.0022 degree
(ii) For sequence GCGCGCGCGC:
Average Melting Temperature for GCGCGCGCGC: 107.867 degrees
Because GC pairs form three hydrogen bonds in water whereas AT pairs only establish two hydrogen bonds, they have a larger stacking energy than AT pairs, which explains why GC pairings have a higher melting point.

10. Sequence: AAATGGCCCTA
AT Content: 58.333333 %
GC Content: 41.666667 %