BIOCB 4381/6381 Final Project

Lingyu Zhou

> Please comment your code wherever you feel necessary. You are free to add new cells to solve problems or test your code. Good luck!
>
> This pdf contains all plots, but the formulae are not displayed properly, refer to the KaTeX in .ipynb file.

Modules that you may need in Part 1

In [ ]:
```
import os
import sys
import copy
import traceback
import re
import pandas as pd
import numpy as np
from Bio import SeqIO

from collections import Counter
```

> ## Part 1: Clone-seq

(a) Parse single row from a SAM file

In [ ]:
```
def parse_sam_row(row, headers):
    """
    Parse a single row from alignment section in SAM file to a dictionary
    Args:
        row [str]: a row from the alignment section of a SAM file
        headers [list]: names of 11 mandatory SAM fields
    Returns:
        sam_dict [dict]: maps names of 11 mandatory fields to corresponding values
    """
    ### YOUR SOLUTION STARTS HERE ###
    sam_dict={}
    for i in range(len(headers)):
        sam_dict[headers[i]]=row.split("\t")[i]
    return sam_dict
    ### YOUR SOLUTION ENDS HERE ###
```

In [ ]:
```
# test case
with open('./data/test_sam_row.txt', 'r') as f:
    test_row = f.readlines()

headers = ["QNAME", "FLAG", "RNAME", "POS", "MAPQ", "CIGAR", "RNEXT", "PNEXT", "TLEN", "SEQ"
parse_sam_row(test_row[0].strip(), headers)
```

Out[ ]:
```
{'QNAME': 'HWI-ST397:389:C42DEACXX:8:1101:1161:8228',
 'FLAG': '0',
 'RNAME': '7696',
 'POS': '1149',
 'MAPQ': '60',
 'CIGAR': '29S72M',
 'RNEXT': '*',
 'PNEXT': '0',
 'TLEN': '0',
 'SEQ': 'GAGGTNNNNNNNNNNNNNNNNNNNNNNNNAAAGNATGATCAGGGAGTGGAACCTTAGCTCGAGCCCGCAGATGACCTGGATG
GAGGCACGGAGGAGCAGGG',
 'QUAL': '<<<?@#########################-0<=#0<=??????????=???????????@@??===<<<<<<====<=<<<=
=<=<<<<<::6:95<<8<<6'}
```

In [ ]:
```
with open("./data/Clone_Seq_Aligned.sam", 'r') as sam_file:sam_lines = sam_file.readlines()
sam_lines[0:20]
```

Out[ ]:
```
['@SQ\tSN:7696\tLN:1602\n',
 '@SQ\tSN:13668\tLN:981\n',
 '@SQ\tSN:5174\tLN:1281\n',
 '@SQ\tSN:8833\tLN:1338\n',
 '@SQ\tSN:8522\tLN:996\n',
 '@SQ\tSN:9938\tLN:888\n',
 '@SQ\tSN:10062\tLN:1494\n',
 '@SQ\tSN:1136\tLN:477\n',
 '@SQ\tSN:8593\tLN:1503\n',
 '@SQ\tSN:8466\tLN:1038\n',
 '@SQ\tSN:7540\tLN:300\n',
 '@SQ\tSN:12438\tLN:1251\n',
 '@SQ\tSN:6936\tLN:1023\n',
 '@SQ\tSN:804\tLN:669\n',
 '@PG\tID:bwa\tPN:bwa\tVN:0.7.12-r1039\tCL:bwa mem Gene_Refs.fasta Clone_Seq.fastq\n',
 'HWI-ST397:389:C42DEACXX:8:1101:1161:8228\t0\t7696\t1149\t60\t29S72M\t*\t0\t0\tGAGGTNNNNNN
NNNNNNNNNNNNNNNNNNNAAAGNATGATCAGGGAGTGGAACCTTAGCTCGAGCCCGCAGATGACCTGGATGGAGGCACGGAGGAGCAGGG
\t<<<?@#########################-0<=#0<=??????????=???????????@@??===<<<<<<====<=<<<==<=<<<<
<::6:95<<8<<6\tNM:i:2\tMD:Z:4T20G46\tAS:i:65\tXS:i:0\n',
 'HWI-ST397:389:C42DEACXX:8:1101:19784:8212\t0\t7696\t962\t60\t71M\t*\t0\t0\tAAAGGAGAAAAGAA
ATTGCCCAGAAGGCAGCAGAGGAAAATGAGAGATACCGGAAGGAGATGGAACAGATT\tCCCFFFFFHHHHHJJJJIJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJJJJJJJJJJJIHHHHHFFFFFEEE\tNM:i:0\tMD:Z:71\tAS:i:71\tXS:i:0\n',
 'HWI-ST397:389:C42DEACXX:8:1101:13264:8455\t0\t7696\t127\t60\t101M\t*\t0\t0\tGTGGGAGACCTGA
AGCTGGTCATCAATGAACCCAGCCGTCTGCCTCTGTTTGATGCCATTCGGCCGCTGATCCCACTGAAGCACCAGGTGGAATATGATCA\tD
DDDDDDDDDDDDDDDDDDEDEDCC@DDBADDDDDDDEDFFFHHHHJJJJJJJJJJJJJJJJJJJJJIJJJIJJJJJJIIJJJJJHHHH
HFFFFFCCC\tNM:i:0\tMD:Z:101\tAS:i:101\tXS:i:0\n',
 'HWI-ST397:389:C42DEACXX:8:1101:19899:12747\t0\t7696\t1276\t60\t71M\t*\t0\t0\tCCAGAGCAGATC
ATGGGGAAGGATGTCCGGCTCCTACGCATCAAGAAGGAGGGATCCTTAGACCTGGCCCT\tCCCFFFFFHHHHHJJJJJJJJJIIJJJJJJ
JJJJJJJJJJIIJJJJJJJJJJHHHHFFFFFFEEEEDDDD\tNM:i:0\tMD:Z:71\tAS:i:71\tXS:i:0\n',
 'HWI-ST397:389:C42DEACXX:8:1101:19165:15788\t0\t7696\t1291\t60\t101M\t*\t0\t0\tGGGAAGGATGT
CCGGCTCCTACGCATCAAGAAGGAGGGATCCTTAGACCTGGCCCTGGAAGGCGGTGTGGACTCCCCCATTGGGAAGGTGGTTGTTTCTGC
\tCCCFFFFFHHHHHJJJJIJJIJIJJJJJJJJJIIHIJJJJIJJFIJJJJJJJJIJHHHHFFFFCCDD@DBDDDDDDDDBDDDCDDDDDDD?
CDBDDDDDDDDC\tNM:i:0\tMD:Z:101\tAS:i:101\tXS:i:0\n']
```

(b) Parse SAM file

In [ ]:
```
def parse_sam_file(fname, headers, int_cols):
    """
    Parse a SAM file to a dictionary
    Args:
        fname [str]: path of the SAM file
        headers [list]: names of 11 mandatory SAM fields
        int_cols [list]: names of fields that should be treated as integers
    Returns:
        sam_df [pandas.DataFrame]: a data frame of the parsed SAM file, where each column co
    """

    ### YOUR SOLUTION STARTS HERE ###
```

```
    with open(fname, 'r') as sam_file:lines = sam_file.readlines()
    df={}
    cont=lines[11+4:]
    for i in range(len(cont)):
        df[i]=parse_sam_row(cont[i],headers)

    df=pd.DataFrame(df)
    df=df.T

    df[int_cols]=df[int_cols].astype(int)
    sam_df=df
    return sam_df

    ### YOUR SOLUTION ENDS HERE ###
```

In [ ]:
```
# df=parse_sam_file("./data/Clone_Seq_Aligned.sam",["QNAME", "FLAG", "RNAME", "POS", "MAPQ",
# df
```

(c) Load Clone-seq alignment data

In [ ]:
```
### YOUR SOLUTION HERE ###
df_clone_seq=parse_sam_file("./data/Clone_Seq_Aligned.sam",\
               ["QNAME", "FLAG", "RNAME", "POS", "MAPQ", "CIGAR", "RNEXT", "PNEXT", "TLEN",
               ["FLAG", "POS", "MAPQ", "PNEXT","TLEN"])
df_clone_seq.head()
```

Out[ ]:

	QNAME	FLAG	RNAME	POS	MAPQ	CIGAR	RNEXT	PNEXT	TLEN	
0	HWI-ST397:389:C42DEACXX:8:1101:1161:8228	0	7696	1149	60	29S72M	*	0	0	G
1	HWI-ST397:389:C42DEACXX:8:1101:19784:8212	0	7696	962	60	71M	*	0	0	
2	HWI-ST397:389:C42DEACXX:8:1101:13264:8455	0	7696	127	60	101M	*	0	0	
3	HWI-ST397:389:C42DEACXX:8:1101:19899:12747	0	7696	1276	60	71M	*	0	0	
4	HWI-ST397:389:C42DEACXX:8:1101:19165:15788	0	7696	1291	60	101M	*	0	0	

In [ ]:
```
(df_clone_seq["RNAME"]).value_counts(dropna=False)
```

Out[ ]:
```
8522     10920
*         8859
12438     6048
8833      5740
7696      4555
5174      1997
8593      1174
9938       982
10062      830
804        544
1136       537
8466       386
6936       340
13668      229
7540       148
Name: RNAME, dtype: int64
```

In [ ]:
```
# filtering
```

```
df_clone_seq=df_clone_seq[df_clone_seq["RNAME"]!="*"]
```

In [ ]:
```
# filtering
df_clone_seq=df_clone_seq[df_clone_seq["FLAG"]==0]
```

(d) Generate descriptive variables

1. *tot_alignments* - The total number of reads left after filtering
2. *num_refs* - The number of unique references aligned to across all your reads
3. *reads_per_ref* - A dictionary mapping unique reference IDs to the number of reads aligned to that reference
4. *average_quality* - The average mapping quality for all reads left after filtering

In [ ]:
```
### YOUR SOLUTION STARTS HERE ###
# tot_alignments
tot_alignments=len(df_clone_seq)
tot_alignments
```

Out[ ]:
```
34403
```

In [ ]:
```
# num_refs
num_refs=len(df_clone_seq["RNAME"].unique())
num_refs
```

Out[ ]:
```
14
```

In [ ]:
```
# reads_per_ref
from pandasql import sqldf
reads_per_df=dict(zip(sqldf("select RNAME,count(QNAME) as COUNT from df_clone_seq group by R
                      sqldf("select RNAME,count(QNAME) as COUNT from df_clone_seq group by R
reads_per_df
```

Out[ ]:
```
{'10062': 830,
 '1136': 535,
 '12438': 6040,
 '13668': 229,
 '5174': 1997,
 '6936': 340,
 '7540': 148,
 '7696': 4554,
 '804': 544,
 '8466': 386,
 '8522': 10919,
 '8593': 1174,
 '8833': 5725,
 '9938': 982}
```

In [ ]:
```
# average_quality

average_quality=sum(df_clone_seq["MAPQ"])/len(df_clone_seq["MAPQ"])
average_quality
```

Out[ ]:
```
59.98145510565939
```

(e) Nucleotide position counts for single read

In [ ]:
```
def pos_nucl_count(seq, pos):
    """
    Get counts of A,T,C,G at each position along the sequence
    Args:
```

```
        seq [str]: a sequence string (from SEQ field in SAM)
        pos [int]: the position index describing the left-most aligning place
    Returns:
        pos_count_dict [dict]: maps each position index to counts of A,T,C,G at that positic
    """
    ### YOUR SOLUTION STARTS HERE ###
    pos_count_dict={}
    for i in range(len(seq)):
        pos_count_dict[pos+i]={"A":0 , "T":0, "C":0, "G":0}
        if seq[i] in ['A', 'T', 'C', 'G']: # ignore N
            pos_count_dict[pos+i][seq[i]]+=1

    return pos_count_dict


    ### YOUR SOLUTION ENDS HERE ###
```

In [ ]:
```
# test case 1: using the example in the instruction

pos_nucl_count('CCGA', 4)
```

Out[ ]:
```
{4: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
 5: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
 6: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 7: {'A': 1, 'T': 0, 'C': 0, 'G': 0}}
```

In [ ]:
```
# test case 2: using the first entry in df_clone_seq
record = df_clone_seq.iloc[0]

count_dict = pos_nucl_count(record['SEQ'], record['POS'])
```

In [ ]:
```
count_dict
```

Out[ ]:

```
{1149: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1150: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1151: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1152: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1153: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
 1154: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1155: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1156: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1157: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1158: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1159: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1160: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1161: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1162: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1163: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1164: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1165: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1166: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1167: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1168: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1169: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1170: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1171: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1172: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1173: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1174: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1175: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1176: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1177: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1178: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1179: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1180: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1181: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1182: {'A': 0, 'T': 0, 'C': 0, 'G': 0},
 1183: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1184: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
 1185: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1186: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1187: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
 1188: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
 1189: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1190: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1191: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1192: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1193: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1194: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1195: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
 1196: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1197: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1198: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1199: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1200: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
 1201: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
 1202: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
 1203: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
 1204: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1205: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1206: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
 1207: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
 1208: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
 1209: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1210: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
 1211: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
 1212: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
 1213: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
 1214: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
```

```
1215: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1216: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
1217: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
1218: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1219: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
1220: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
1221: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1222: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
1223: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
1224: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
1225: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
1226: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1227: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1228: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
1229: {'A': 0, 'T': 1, 'C': 0, 'G': 0},
1230: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1231: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1232: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
1233: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1234: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1235: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
1236: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
1237: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
1238: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1239: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1240: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
1241: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1242: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1243: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
1244: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1245: {'A': 0, 'T': 0, 'C': 1, 'G': 0},
1246: {'A': 1, 'T': 0, 'C': 0, 'G': 0},
1247: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1248: {'A': 0, 'T': 0, 'C': 0, 'G': 1},
1249: {'A': 0, 'T': 0, 'C': 0, 'G': 1}}
```

(f) Nucleotide counts at each position for all reads in a data frame

In [ ]:

```python
def pos_nucl_count_all(df, qual_thres=30, qual_base=33, nucl_all=['A', 'T', 'C', 'G']):
    """
    Get counts of A,T,C,G at each position across all reads in a given data frame.
    In particular, the counts are only updated when the a position for a specific read
    has quality score >= quality threshold

    Args:
        df [pandas.DataFrame]: a data frame where reads are aligned to a specific gene
        qual_thres [int]: the quality threshold (default=30)
        qual_base [int]: quality base that we need to subtract to obtain quality score value
        nucl_all [list]: nucleotides to consider in this problem (default=["A", "T", "C", "G
    Returns:
        pos2count_all [dict]: maps each position index to counts of A,T,C,G at that position
    """

    ### YOUR SOLUTION STARTS HERE ###


    pos2count_all={}


    for i in range(len(df)):
        seq=df.iloc[i]['SEQ']
        pos=df.iloc[i]['POS']
        qual=df.iloc[i]['QUAL']
        lenqual=len(qual)
        for j in range(lenqual):
```

```
            if ord(qual[j])-qual_base>=qual_thres and (pos+j not in pos2count_all):
                    pos2count_all[pos+j]=dict(zip(nucl_all,[0,0,0,0]))
                    pos2count_all[pos+j][seq[j]]+=1
            elif ord(qual[j])-qual_base>=qual_thres and (pos+j in pos2count_all):
                    pos2count_all[pos+j][seq[j]]+=1

    return pos2count_all

    ### YOUR SOLUTION ENDS HERE ###
```

In [ ]:

```
# Test case: read counts for gene "7696"

df_gene_x = df_clone_seq.query('RNAME == "7696"')
count_dict_x = pos_nucl_count_all(df_gene_x)
count_dict_x
```

Out[ ]:

```
{1152: {'A': 1, 'T': 0, 'C': 1, 'G': 255},
 1153: {'A': 1, 'T': 243, 'C': 0, 'G': 1},
 1186: {'A': 1, 'T': 1, 'C': 0, 'G': 267},
 1187: {'A': 0, 'T': 2, 'C': 269, 'G': 0},
 1188: {'A': 266, 'T': 1, 'C': 2, 'G': 0},
 1189: {'A': 2, 'T': 0, 'C': 1, 'G': 264},
 1190: {'A': 265, 'T': 0, 'C': 0, 'G': 3},
 1191: {'A': 0, 'T': 266, 'C': 1, 'G': 2},
 1192: {'A': 1, 'T': 0, 'C': 1, 'G': 268},
 1193: {'A': 269, 'T': 0, 'C': 0, 'G': 1},
 1194: {'A': 2, 'T': 0, 'C': 273, 'G': 1},
 1196: {'A': 1, 'T': 270, 'C': 0, 'G': 2},
 1197: {'A': 1, 'T': 0, 'C': 0, 'G': 271},
 1198: {'A': 1, 'T': 1, 'C': 1, 'G': 266},
 1199: {'A': 266, 'T': 0, 'C': 0, 'G': 0},
 1200: {'A': 0, 'T': 265, 'C': 1, 'G': 1},
 1201: {'A': 0, 'T': 0, 'C': 2, 'G': 265},
 1202: {'A': 0, 'T': 2, 'C': 0, 'G': 264},
 1203: {'A': 262, 'T': 1, 'C': 0, 'G': 1},
 1204: {'A': 1, 'T': 1, 'C': 0, 'G': 259},
 1205: {'A': 0, 'T': 1, 'C': 0, 'G': 260},
 1206: {'A': 0, 'T': 0, 'C': 256, 'G': 0},
 1207: {'A': 255, 'T': 1, 'C': 1, 'G': 0},
 1208: {'A': 0, 'T': 1, 'C': 259, 'G': 0},
 1209: {'A': 0, 'T': 1, 'C': 0, 'G': 264},
 1210: {'A': 1, 'T': 1, 'C': 0, 'G': 259},
 962: {'A': 202, 'T': 0, 'C': 0, 'G': 0},
 963: {'A': 212, 'T': 0, 'C': 0, 'G': 0},
 964: {'A': 207, 'T': 0, 'C': 0, 'G': 0},
 965: {'A': 1, 'T': 1, 'C': 0, 'G': 202},
 966: {'A': 0, 'T': 0, 'C': 0, 'G': 207},
 967: {'A': 185, 'T': 0, 'C': 0, 'G': 0},
 968: {'A': 0, 'T': 0, 'C': 0, 'G': 187},
 969: {'A': 177, 'T': 0, 'C': 0, 'G': 0},
 970: {'A': 187, 'T': 0, 'C': 0, 'G': 0},
 971: {'A': 189, 'T': 0, 'C': 0, 'G': 0},
 972: {'A': 184, 'T': 0, 'C': 0, 'G': 0},
 973: {'A': 0, 'T': 0, 'C': 0, 'G': 179},
 974: {'A': 186, 'T': 0, 'C': 0, 'G': 0},
 975: {'A': 185, 'T': 0, 'C': 0, 'G': 0},
 976: {'A': 185, 'T': 0, 'C': 0, 'G': 0},
 977: {'A': 2, 'T': 180, 'C': 0, 'G': 0},
 978: {'A': 0, 'T': 184, 'C': 0, 'G': 0},
 979: {'A': 1, 'T': 0, 'C': 0, 'G': 178},
 980: {'A': 0, 'T': 0, 'C': 182, 'G': 0},
 981: {'A': 0, 'T': 0, 'C': 181, 'G': 0},
 982: {'A': 0, 'T': 0, 'C': 179, 'G': 0},
 983: {'A': 175, 'T': 0, 'C': 0, 'G': 0},
 984: {'A': 0, 'T': 0, 'C': 0, 'G': 178},
 985: {'A': 172, 'T': 0, 'C': 0, 'G': 0},
 986: {'A': 174, 'T': 0, 'C': 0, 'G': 0},
 987: {'A': 0, 'T': 1, 'C': 0, 'G': 172},
 988: {'A': 1, 'T': 0, 'C': 0, 'G': 170},
 989: {'A': 0, 'T': 0, 'C': 172, 'G': 1},
 990: {'A': 176, 'T': 1, 'C': 0, 'G': 0},
 991: {'A': 0, 'T': 0, 'C': 1, 'G': 175},
 992: {'A': 0, 'T': 0, 'C': 174, 'G': 0},
 993: {'A': 169, 'T': 0, 'C': 0, 'G': 0},
 994: {'A': 1, 'T': 0, 'C': 0, 'G': 163},
 995: {'A': 160, 'T': 0, 'C': 1, 'G': 1},
 996: {'A': 0, 'T': 0, 'C': 0, 'G': 161},
 997: {'A': 0, 'T': 0, 'C': 0, 'G': 161},
 998: {'A': 160, 'T': 0, 'C': 0, 'G': 0},
 999: {'A': 162, 'T': 0, 'C': 0, 'G': 0},
 1000: {'A': 161, 'T': 1, 'C': 0, 'G': 0},
 1001: {'A': 162, 'T': 0, 'C': 1, 'G': 0},
```

```
1002: {'A': 0, 'T': 159, 'C': 1, 'G': 0},
1003: {'A': 0, 'T': 1, 'C': 0, 'G': 158},
1004: {'A': 157, 'T': 1, 'C': 1, 'G': 0},
1005: {'A': 0, 'T': 0, 'C': 1, 'G': 158},
1006: {'A': 157, 'T': 0, 'C': 0, 'G': 1},
1007: {'A': 0, 'T': 0, 'C': 1, 'G': 156},
1008: {'A': 155, 'T': 1, 'C': 0, 'G': 1},
1009: {'A': 1, 'T': 140, 'C': 0, 'G': 1},
1010: {'A': 151, 'T': 0, 'C': 0, 'G': 2},
1011: {'A': 0, 'T': 1, 'C': 156, 'G': 0},
1012: {'A': 1, 'T': 0, 'C': 151, 'G': 1},
1013: {'A': 0, 'T': 0, 'C': 0, 'G': 151},
1014: {'A': 1, 'T': 0, 'C': 1, 'G': 148},
1015: {'A': 150, 'T': 0, 'C': 0, 'G': 0},
1016: {'A': 147, 'T': 0, 'C': 0, 'G': 1},
1017: {'A': 1, 'T': 0, 'C': 0, 'G': 151},
1018: {'A': 0, 'T': 0, 'C': 0, 'G': 151},
1019: {'A': 152, 'T': 0, 'C': 1, 'G': 0},
1020: {'A': 0, 'T': 1, 'C': 0, 'G': 152},
1021: {'A': 151, 'T': 0, 'C': 0, 'G': 1},
1022: {'A': 0, 'T': 149, 'C': 1, 'G': 0},
1023: {'A': 1, 'T': 0, 'C': 0, 'G': 152},
1024: {'A': 0, 'T': 0, 'C': 0, 'G': 150},
1025: {'A': 150, 'T': 0, 'C': 0, 'G': 1},
1026: {'A': 149, 'T': 1, 'C': 0, 'G': 0},
1027: {'A': 0, 'T': 0, 'C': 150, 'G': 1},
1028: {'A': 150, 'T': 0, 'C': 0, 'G': 1},
1029: {'A': 1, 'T': 0, 'C': 0, 'G': 149},
1030: {'A': 150, 'T': 0, 'C': 1, 'G': 0},
1031: {'A': 0, 'T': 152, 'C': 1, 'G': 0},
1032: {'A': 1, 'T': 151, 'C': 0, 'G': 0},
127: {'A': 0, 'T': 0, 'C': 2, 'G': 279},
128: {'A': 0, 'T': 278, 'C': 1, 'G': 1},
129: {'A': 1, 'T': 1, 'C': 0, 'G': 285},
130: {'A': 0, 'T': 1, 'C': 1, 'G': 283},
131: {'A': 0, 'T': 0, 'C': 1, 'G': 279},
132: {'A': 278, 'T': 0, 'C': 0, 'G': 1},
133: {'A': 1, 'T': 0, 'C': 0, 'G': 281},
134: {'A': 273, 'T': 0, 'C': 0, 'G': 1},
135: {'A': 1, 'T': 0, 'C': 285, 'G': 0},
136: {'A': 0, 'T': 0, 'C': 292, 'G': 0},
137: {'A': 2, 'T': 293, 'C': 1, 'G': 0},
138: {'A': 0, 'T': 3, 'C': 0, 'G': 297},
139: {'A': 301, 'T': 0, 'C': 0, 'G': 1},
140: {'A': 310, 'T': 0, 'C': 0, 'G': 0},
141: {'A': 1, 'T': 1, 'C': 0, 'G': 302},
142: {'A': 0, 'T': 0, 'C': 302, 'G': 1},
143: {'A': 0, 'T': 313, 'C': 1, 'G': 0},
144: {'A': 1, 'T': 1, 'C': 0, 'G': 310},
145: {'A': 1, 'T': 1, 'C': 0, 'G': 312},
146: {'A': 1, 'T': 308, 'C': 0, 'G': 2},
147: {'A': 0, 'T': 1, 'C': 314, 'G': 0},
148: {'A': 319, 'T': 0, 'C': 1, 'G': 0},
149: {'A': 1, 'T': 316, 'C': 0, 'G': 0},
150: {'A': 0, 'T': 1, 'C': 324, 'G': 0},
151: {'A': 324, 'T': 2, 'C': 1, 'G': 0},
152: {'A': 332, 'T': 0, 'C': 0, 'G': 0},
153: {'A': 1, 'T': 330, 'C': 0, 'G': 0},
154: {'A': 0, 'T': 1, 'C': 1, 'G': 327},
155: {'A': 328, 'T': 0, 'C': 0, 'G': 1},
156: {'A': 317, 'T': 0, 'C': 0, 'G': 0},
157: {'A': 1, 'T': 1, 'C': 335, 'G': 0},
158: {'A': 0, 'T': 0, 'C': 333, 'G': 1},
159: {'A': 1, 'T': 0, 'C': 329, 'G': 0},
160: {'A': 333, 'T': 0, 'C': 1, 'G': 0},
161: {'A': 1, 'T': 0, 'C': 1, 'G': 332},
```

```
162: {'A': 0, 'T': 0, 'C': 333, 'G': 1},
163: {'A': 0, 'T': 0, 'C': 328, 'G': 0},
164: {'A': 1, 'T': 0, 'C': 1, 'G': 325},
165: {'A': 0, 'T': 318, 'C': 0, 'G': 2},
166: {'A': 0, 'T': 1, 'C': 324, 'G': 0},
167: {'A': 1, 'T': 326, 'C': 2, 'G': 0},
168: {'A': 0, 'T': 1, 'C': 0, 'G': 322},
169: {'A': 0, 'T': 1, 'C': 329, 'G': 1},
170: {'A': 0, 'T': 0, 'C': 332, 'G': 0},
171: {'A': 0, 'T': 331, 'C': 1, 'G': 0},
172: {'A': 0, 'T': 1, 'C': 331, 'G': 1},
173: {'A': 0, 'T': 338, 'C': 2, 'G': 0},
174: {'A': 0, 'T': 1, 'C': 1, 'G': 337},
175: {'A': 0, 'T': 342, 'C': 0, 'G': 1},
176: {'A': 0, 'T': 340, 'C': 1, 'G': 0},
177: {'A': 0, 'T': 339, 'C': 0, 'G': 0},
178: {'A': 0, 'T': 1, 'C': 0, 'G': 337},
179: {'A': 332, 'T': 1, 'C': 0, 'G': 1},
180: {'A': 1, 'T': 331, 'C': 0, 'G': 0},
181: {'A': 0, 'T': 2, 'C': 0, 'G': 330},
182: {'A': 0, 'T': 0, 'C': 327, 'G': 2},
183: {'A': 1, 'T': 1, 'C': 329, 'G': 0},
184: {'A': 332, 'T': 1, 'C': 1, 'G': 0},
185: {'A': 2, 'T': 328, 'C': 0, 'G': 1},
186: {'A': 0, 'T': 331, 'C': 1, 'G': 1},
187: {'A': 0, 'T': 1, 'C': 337, 'G': 0},
188: {'A': 2, 'T': 0, 'C': 1, 'G': 342},
189: {'A': 0, 'T': 1, 'C': 0, 'G': 337},
190: {'A': 1, 'T': 1, 'C': 333, 'G': 0},
191: {'A': 1, 'T': 0, 'C': 339, 'G': 0},
192: {'A': 0, 'T': 0, 'C': 0, 'G': 339},
193: {'A': 0, 'T': 0, 'C': 343, 'G': 1},
194: {'A': 0, 'T': 349, 'C': 1, 'G': 0},
195: {'A': 0, 'T': 0, 'C': 1, 'G': 351},
196: {'A': 348, 'T': 0, 'C': 0, 'G': 1},
197: {'A': 0, 'T': 349, 'C': 1, 'G': 0},
198: {'A': 0, 'T': 1, 'C': 358, 'G': 0},
199: {'A': 0, 'T': 0, 'C': 363, 'G': 1},
200: {'A': 1, 'T': 0, 'C': 367, 'G': 0},
201: {'A': 363, 'T': 1, 'C': 0, 'G': 0},
202: {'A': 1, 'T': 0, 'C': 367, 'G': 1},
203: {'A': 0, 'T': 364, 'C': 1, 'G': 1},
204: {'A': 1, 'T': 0, 'C': 1, 'G': 369},
205: {'A': 368, 'T': 1, 'C': 0, 'G': 0},
206: {'A': 369, 'T': 0, 'C': 1, 'G': 0},
207: {'A': 0, 'T': 1, 'C': 0, 'G': 367},
208: {'A': 0, 'T': 0, 'C': 365, 'G': 1},
209: {'A': 364, 'T': 0, 'C': 0, 'G': 0},
210: {'A': 1, 'T': 0, 'C': 366, 'G': 0},
211: {'A': 0, 'T': 1, 'C': 373, 'G': 1},
212: {'A': 373, 'T': 0, 'C': 2, 'G': 0},
213: {'A': 2, 'T': 0, 'C': 0, 'G': 369},
214: {'A': 0, 'T': 0, 'C': 1, 'G': 377},
215: {'A': 0, 'T': 350, 'C': 1, 'G': 1},
216: {'A': 1, 'T': 1, 'C': 0, 'G': 361},
217: {'A': 0, 'T': 0, 'C': 0, 'G': 370},
218: {'A': 367, 'T': 0, 'C': 0, 'G': 2},
219: {'A': 373, 'T': 1, 'C': 0, 'G': 0},
220: {'A': 1, 'T': 365, 'C': 0, 'G': 1},
221: {'A': 366, 'T': 1, 'C': 0, 'G': 1},
222: {'A': 2, 'T': 364, 'C': 1, 'G': 0},
223: {'A': 1, 'T': 1, 'C': 0, 'G': 363},
224: {'A': 367, 'T': 1, 'C': 0, 'G': 1},
225: {'A': 2, 'T': 369, 'C': 0, 'G': 0},
226: {'A': 0, 'T': 2, 'C': 367, 'G': 0},
227: {'A': 370, 'T': 0, 'C': 1, 'G': 1},
```

```
1276: {'A': 0, 'T': 0, 'C': 253, 'G': 1},
1277: {'A': 1, 'T': 0, 'C': 254, 'G': 1},
1278: {'A': 251, 'T': 0, 'C': 1, 'G': 0},
1279: {'A': 1, 'T': 0, 'C': 1, 'G': 250},
1280: {'A': 248, 'T': 0, 'C': 1, 'G': 1},
1281: {'A': 0, 'T': 0, 'C': 1, 'G': 251},
1282: {'A': 1, 'T': 0, 'C': 249, 'G': 0},
1283: {'A': 248, 'T': 1, 'C': 0, 'G': 1},
1284: {'A': 1, 'T': 1, 'C': 0, 'G': 248},
1285: {'A': 249, 'T': 1, 'C': 0, 'G': 1},
1286: {'A': 0, 'T': 252, 'C': 1, 'G': 1},
1287: {'A': 3, 'T': 0, 'C': 253, 'G': 0},
1288: {'A': 254, 'T': 0, 'C': 1, 'G': 1},
1289: {'A': 1, 'T': 256, 'C': 1, 'G': 1},
1290: {'A': 0, 'T': 1, 'C': 1, 'G': 258},
1291: {'A': 0, 'T': 0, 'C': 2, 'G': 259},
1292: {'A': 1, 'T': 1, 'C': 0, 'G': 260},
1293: {'A': 1, 'T': 1, 'C': 0, 'G': 258},
1294: {'A': 257, 'T': 0, 'C': 1, 'G': 1},
1295: {'A': 257, 'T': 1, 'C': 0, 'G': 1},
1296: {'A': 0, 'T': 1, 'C': 1, 'G': 259},
1297: {'A': 0, 'T': 1, 'C': 0, 'G': 265},
1298: {'A': 267, 'T': 0, 'C': 0, 'G': 0},
1299: {'A': 1, 'T': 269, 'C': 0, 'G': 0},
1300: {'A': 0, 'T': 0, 'C': 0, 'G': 269},
1301: {'A': 0, 'T': 273, 'C': 0, 'G': 1},
1302: {'A': 1, 'T': 1, 'C': 269, 'G': 0},
1303: {'A': 0, 'T': 1, 'C': 277, 'G': 0},
1304: {'A': 1, 'T': 0, 'C': 0, 'G': 279},
1305: {'A': 0, 'T': 1, 'C': 1, 'G': 275},
1306: {'A': 0, 'T': 0, 'C': 283, 'G': 0},
1307: {'A': 0, 'T': 286, 'C': 2, 'G': 0},
1308: {'A': 0, 'T': 0, 'C': 286, 'G': 1},
1309: {'A': 1, 'T': 0, 'C': 283, 'G': 1},
1310: {'A': 0, 'T': 280, 'C': 1, 'G': 1},
1311: {'A': 280, 'T': 0, 'C': 0, 'G': 0},
1312: {'A': 0, 'T': 0, 'C': 279, 'G': 1},
1313: {'A': 1, 'T': 0, 'C': 2, 'G': 280},
1314: {'A': 1, 'T': 1, 'C': 281, 'G': 0},
1315: {'A': 282, 'T': 0, 'C': 0, 'G': 1},
1316: {'A': 1, 'T': 285, 'C': 1, 'G': 0},
1317: {'A': 0, 'T': 1, 'C': 284, 'G': 1},
1318: {'A': 287, 'T': 0, 'C': 2, 'G': 0},
1319: {'A': 290, 'T': 0, 'C': 0, 'G': 0},
1320: {'A': 0, 'T': 2, 'C': 0, 'G': 289},
1321: {'A': 291, 'T': 0, 'C': 1, 'G': 1},
1322: {'A': 291, 'T': 0, 'C': 0, 'G': 1},
1323: {'A': 1, 'T': 0, 'C': 0, 'G': 294},
1324: {'A': 1, 'T': 0, 'C': 0, 'G': 296},
1325: {'A': 297, 'T': 0, 'C': 0, 'G': 1},
1326: {'A': 3, 'T': 0, 'C': 0, 'G': 299},
1327: {'A': 0, 'T': 0, 'C': 0, 'G': 302},
1328: {'A': 0, 'T': 0, 'C': 0, 'G': 301},
1329: {'A': 302, 'T': 0, 'C': 0, 'G': 0},
1330: {'A': 0, 'T': 300, 'C': 0, 'G': 2},
1331: {'A': 1, 'T': 0, 'C': 298, 'G': 1},
1332: {'A': 0, 'T': 0, 'C': 306, 'G': 1},
1333: {'A': 1, 'T': 300, 'C': 0, 'G': 0},
1334: {'A': 0, 'T': 302, 'C': 0, 'G': 0},
1335: {'A': 302, 'T': 0, 'C': 1, 'G': 0},
1336: {'A': 0, 'T': 0, 'C': 1, 'G': 296},
1337: {'A': 292, 'T': 1, 'C': 0, 'G': 0},
1338: {'A': 0, 'T': 1, 'C': 298, 'G': 0},
1339: {'A': 1, 'T': 0, 'C': 303, 'G': 0},
1340: {'A': 0, 'T': 308, 'C': 0, 'G': 1},
1341: {'A': 0, 'T': 2, 'C': 0, 'G': 310},
```

```
1342: {'A': 0, 'T': 0, 'C': 0, 'G': 312},
1343: {'A': 0, 'T': 0, 'C': 312, 'G': 0},
1344: {'A': 0, 'T': 0, 'C': 317, 'G': 0},
1345: {'A': 0, 'T': 0, 'C': 323, 'G': 0},
1346: {'A': 0, 'T': 327, 'C': 0, 'G': 0},
1347: {'A': 0, 'T': 0, 'C': 0, 'G': 326},
1348: {'A': 0, 'T': 0, 'C': 1, 'G': 326},
1349: {'A': 322, 'T': 0, 'C': 0, 'G': 0},
1350: {'A': 328, 'T': 0, 'C': 0, 'G': 0},
1351: {'A': 0, 'T': 0, 'C': 0, 'G': 332},
1352: {'A': 0, 'T': 0, 'C': 0, 'G': 327},
1353: {'A': 0, 'T': 0, 'C': 324, 'G': 0},
1354: {'A': 0, 'T': 0, 'C': 0, 'G': 323},
1355: {'A': 0, 'T': 0, 'C': 0, 'G': 326},
1356: {'A': 0, 'T': 304, 'C': 0, 'G': 0},
1357: {'A': 0, 'T': 0, 'C': 0, 'G': 322},
1358: {'A': 0, 'T': 319, 'C': 0, 'G': 0},
1359: {'A': 0, 'T': 0, 'C': 0, 'G': 329},
1360: {'A': 0, 'T': 1, 'C': 0, 'G': 329},
1361: {'A': 326, 'T': 0, 'C': 0, 'G': 0},
1362: {'A': 0, 'T': 0, 'C': 327, 'G': 0},
1363: {'A': 0, 'T': 334, 'C': 0, 'G': 0},
1364: {'A': 0, 'T': 0, 'C': 334, 'G': 0},
1365: {'A': 0, 'T': 0, 'C': 337, 'G': 0},
1366: {'A': 0, 'T': 0, 'C': 332, 'G': 0},
1367: {'A': 0, 'T': 0, 'C': 332, 'G': 0},
1368: {'A': 1, 'T': 0, 'C': 335, 'G': 0},
1369: {'A': 334, 'T': 0, 'C': 0, 'G': 0},
1370: {'A': 0, 'T': 330, 'C': 0, 'G': 0},
1371: {'A': 0, 'T': 328, 'C': 0, 'G': 0},
1372: {'A': 0, 'T': 0, 'C': 0, 'G': 333},
1373: {'A': 0, 'T': 1, 'C': 0, 'G': 331},
1374: {'A': 0, 'T': 1, 'C': 0, 'G': 327},
1375: {'A': 327, 'T': 0, 'C': 0, 'G': 0},
1376: {'A': 326, 'T': 0, 'C': 0, 'G': 0},
1377: {'A': 0, 'T': 0, 'C': 0, 'G': 332},
1378: {'A': 0, 'T': 0, 'C': 0, 'G': 332},
1379: {'A': 2, 'T': 296, 'C': 0, 'G': 0},
1380: {'A': 0, 'T': 1, 'C': 0, 'G': 327},
1381: {'A': 1, 'T': 0, 'C': 0, 'G': 338},
1382: {'A': 0, 'T': 327, 'C': 0, 'G': 0},
1383: {'A': 0, 'T': 337, 'C': 0, 'G': 0},
1384: {'A': 0, 'T': 0, 'C': 0, 'G': 337},
1385: {'A': 0, 'T': 338, 'C': 0, 'G': 0},
1386: {'A': 0, 'T': 336, 'C': 0, 'G': 0},
1387: {'A': 0, 'T': 338, 'C': 0, 'G': 0},
1388: {'A': 0, 'T': 0, 'C': 337, 'G': 0},
1389: {'A': 0, 'T': 335, 'C': 0, 'G': 0},
1390: {'A': 0, 'T': 0, 'C': 0, 'G': 331},
1391: {'A': 0, 'T': 0, 'C': 332, 'G': 0},
1443: {'A': 0, 'T': 1, 'C': 335, 'G': 0},
1444: {'A': 0, 'T': 0, 'C': 0, 'G': 339},
1445: {'A': 336, 'T': 0, 'C': 0, 'G': 0},
1446: {'A': 0, 'T': 0, 'C': 0, 'G': 332},
1447: {'A': 322, 'T': 0, 'C': 0, 'G': 0},
1448: {'A': 0, 'T': 317, 'C': 0, 'G': 0},
1449: {'A': 0, 'T': 0, 'C': 318, 'G': 2},
1450: {'A': 319, 'T': 0, 'C': 0, 'G': 0},
1451: {'A': 0, 'T': 317, 'C': 0, 'G': 0},
1452: {'A': 0, 'T': 0, 'C': 0, 'G': 318},
1453: {'A': 1, 'T': 0, 'C': 0, 'G': 317},
1454: {'A': 0, 'T': 0, 'C': 317, 'G': 0},
1455: {'A': 316, 'T': 0, 'C': 0, 'G': 0},
1456: {'A': 319, 'T': 0, 'C': 0, 'G': 1},
1457: {'A': 0, 'T': 322, 'C': 0, 'G': 0},
1458: {'A': 0, 'T': 0, 'C': 322, 'G': 0},
```

```
1459: {'A': 318, 'T': 1, 'C': 0, 'G': 0},
1460: {'A': 318, 'T': 0, 'C': 0, 'G': 0},
1461: {'A': 0, 'T': 0, 'C': 320, 'G': 0},
1462: {'A': 0, 'T': 0, 'C': 0, 'G': 321},
1463: {'A': 0, 'T': 0, 'C': 0, 'G': 326},
1464: {'A': 0, 'T': 0, 'C': 323, 'G': 0},
1465: {'A': 323, 'T': 0, 'C': 0, 'G': 0},
1466: {'A': 321, 'T': 0, 'C': 0, 'G': 0},
1467: {'A': 0, 'T': 0, 'C': 0, 'G': 327},
1468: {'A': 329, 'T': 0, 'C': 0, 'G': 0},
1469: {'A': 0, 'T': 329, 'C': 0, 'G': 0},
1470: {'A': 0, 'T': 334, 'C': 0, 'G': 0},
1471: {'A': 1, 'T': 0, 'C': 0, 'G': 332},
1472: {'A': 0, 'T': 328, 'C': 0, 'G': 1},
1473: {'A': 1, 'T': 0, 'C': 0, 'G': 335},
1474: {'A': 333, 'T': 0, 'C': 0, 'G': 0},
1475: {'A': 0, 'T': 0, 'C': 334, 'G': 0},
1476: {'A': 333, 'T': 0, 'C': 0, 'G': 0},
1477: {'A': 0, 'T': 1, 'C': 0, 'G': 328},
1478: {'A': 328, 'T': 0, 'C': 1, 'G': 0},
1479: {'A': 0, 'T': 1, 'C': 334, 'G': 0},
1480: {'A': 1, 'T': 333, 'C': 0, 'G': 0},
1481: {'A': 325, 'T': 0, 'C': 0, 'G': 1},
1482: {'A': 1, 'T': 0, 'C': 323, 'G': 0},
1483: {'A': 314, 'T': 0, 'C': 1, 'G': 0},
1484: {'A': 0, 'T': 1, 'C': 328, 'G': 0},
1485: {'A': 0, 'T': 0, 'C': 330, 'G': 0},
1486: {'A': 0, 'T': 0, 'C': 329, 'G': 0},
1487: {'A': 0, 'T': 324, 'C': 0, 'G': 0},
1488: {'A': 0, 'T': 0, 'C': 1, 'G': 323},
1489: {'A': 0, 'T': 0, 'C': 1, 'G': 323},
1490: {'A': 0, 'T': 0, 'C': 326, 'G': 0},
1491: {'A': 0, 'T': 327, 'C': 0, 'G': 0},
1492: {'A': 0, 'T': 0, 'C': 0, 'G': 328},
1493: {'A': 322, 'T': 0, 'C': 0, 'G': 2},
1494: {'A': 0, 'T': 0, 'C': 1, 'G': 322},
1495: {'A': 0, 'T': 1, 'C': 0, 'G': 319},
1496: {'A': 0, 'T': 0, 'C': 320, 'G': 1},
1497: {'A': 1, 'T': 315, 'C': 0, 'G': 0},
1498: {'A': 0, 'T': 1, 'C': 0, 'G': 318},
1499: {'A': 309, 'T': 1, 'C': 0, 'G': 1},
1500: {'A': 0, 'T': 0, 'C': 314, 'G': 1},
1501: {'A': 2, 'T': 1, 'C': 1, 'G': 318},
1502: {'A': 0, 'T': 0, 'C': 318, 'G': 2},
1503: {'A': 1, 'T': 318, 'C': 0, 'G': 0},
1504: {'A': 0, 'T': 1, 'C': 1, 'G': 312},
1505: {'A': 0, 'T': 0, 'C': 307, 'G': 1},
1506: {'A': 0, 'T': 0, 'C': 313, 'G': 0},
1507: {'A': 0, 'T': 2, 'C': 313, 'G': 0},
1508: {'A': 0, 'T': 316, 'C': 2, 'G': 1},
1509: {'A': 0, 'T': 2, 'C': 1, 'G': 313},
1510: {'A': 0, 'T': 0, 'C': 314, 'G': 0},
1511: {'A': 303, 'T': 1, 'C': 1, 'G': 0},
1512: {'A': 0, 'T': 1, 'C': 0, 'G': 300},
1513: {'A': 299, 'T': 0, 'C': 1, 'G': 2},
1514: {'A': 296, 'T': 0, 'C': 1, 'G': 0},
1515: {'A': 1, 'T': 0, 'C': 0, 'G': 287},
1516: {'A': 1, 'T': 0, 'C': 0, 'G': 284},
1517: {'A': 2, 'T': 0, 'C': 285, 'G': 0},
1518: {'A': 1, 'T': 0, 'C': 281, 'G': 1},
1519: {'A': 0, 'T': 277, 'C': 0, 'G': 2},
1520: {'A': 0, 'T': 0, 'C': 1, 'G': 280},
1521: {'A': 0, 'T': 0, 'C': 2, 'G': 274},
1522: {'A': 268, 'T': 1, 'C': 1, 'G': 0},
1523: {'A': 267, 'T': 1, 'C': 0, 'G': 1},
1524: {'A': 0, 'T': 265, 'C': 0, 'G': 2},
```

```
1525: {'A': 2, 'T': 0, 'C': 267, 'G': 1},
1526: {'A': 272, 'T': 1, 'C': 0, 'G': 0},
1527: {'A': 1, 'T': 1, 'C': 0, 'G': 268},
1528: {'A': 0, 'T': 1, 'C': 1, 'G': 266},
1529: {'A': 1, 'T': 0, 'C': 1, 'G': 267},
1530: {'A': 1, 'T': 0, 'C': 264, 'G': 1},
1531: {'A': 1, 'T': 0, 'C': 0, 'G': 261},
1532: {'A': 1, 'T': 0, 'C': 0, 'G': 256},
1533: {'A': 1, 'T': 0, 'C': 1, 'G': 258},
1534: {'A': 1, 'T': 0, 'C': 1, 'G': 253},
1535: {'A': 241, 'T': 1, 'C': 0, 'G': 2},
1536: {'A': 0, 'T': 0, 'C': 239, 'G': 2},
1537: {'A': 0, 'T': 246, 'C': 0, 'G': 2},
1538: {'A': 1, 'T': 0, 'C': 0, 'G': 244},
1539: {'A': 1, 'T': 0, 'C': 1, 'G': 245},
1540: {'A': 245, 'T': 1, 'C': 1, 'G': 0},
1541: {'A': 0, 'T': 241, 'C': 0, 'G': 1},
1542: {'A': 0, 'T': 0, 'C': 240, 'G': 2},
1392: {'A': 0, 'T': 337, 'C': 0, 'G': 0},
1393: {'A': 0, 'T': 0, 'C': 0, 'G': 340},
1394: {'A': 0, 'T': 339, 'C': 0, 'G': 0},
1395: {'A': 0, 'T': 0, 'C': 0, 'G': 341},
1396: {'A': 0, 'T': 344, 'C': 0, 'G': 0},
1397: {'A': 345, 'T': 0, 'C': 0, 'G': 0},
1398: {'A': 0, 'T': 346, 'C': 0, 'G': 0},
1399: {'A': 0, 'T': 0, 'C': 0, 'G': 344},
1400: {'A': 343, 'T': 0, 'C': 0, 'G': 0},
1401: {'A': 1, 'T': 0, 'C': 0, 'G': 346},
1402: {'A': 1, 'T': 0, 'C': 341, 'G': 0},
1403: {'A': 0, 'T': 0, 'C': 0, 'G': 350},
1404: {'A': 0, 'T': 0, 'C': 0, 'G': 346},
1405: {'A': 0, 'T': 0, 'C': 0, 'G': 346},
1406: {'A': 0, 'T': 0, 'C': 0, 'G': 345},
1407: {'A': 324, 'T': 0, 'C': 0, 'G': 0},
1408: {'A': 0, 'T': 0, 'C': 0, 'G': 336},
1409: {'A': 0, 'T': 0, 'C': 337, 'G': 0},
1410: {'A': 0, 'T': 347, 'C': 0, 'G': 0},
1411: {'A': 0, 'T': 0, 'C': 0, 'G': 347},
1412: {'A': 0, 'T': 0, 'C': 349, 'G': 0},
1413: {'A': 0, 'T': 349, 'C': 0, 'G': 0},
1414: {'A': 0, 'T': 0, 'C': 0, 'G': 354},
1415: {'A': 347, 'T': 0, 'C': 0, 'G': 0},
1416: {'A': 0, 'T': 0, 'C': 0, 'G': 347},
1417: {'A': 0, 'T': 0, 'C': 353, 'G': 0},
1418: {'A': 1, 'T': 1, 'C': 0, 'G': 353},
1419: {'A': 0, 'T': 0, 'C': 0, 'G': 350},
1420: {'A': 0, 'T': 0, 'C': 353, 'G': 0},
1421: {'A': 355, 'T': 0, 'C': 0, 'G': 0},
1422: {'A': 0, 'T': 355, 'C': 0, 'G': 0},
1423: {'A': 0, 'T': 0, 'C': 0, 'G': 354},
1424: {'A': 1, 'T': 0, 'C': 0, 'G': 348},
1425: {'A': 0, 'T': 349, 'C': 0, 'G': 1},
1426: {'A': 0, 'T': 1, 'C': 0, 'G': 347},
1427: {'A': 0, 'T': 0, 'C': 0, 'G': 347},
1428: {'A': 0, 'T': 0, 'C': 351, 'G': 0},
1429: {'A': 347, 'T': 0, 'C': 0, 'G': 0},
1430: {'A': 0, 'T': 352, 'C': 0, 'G': 0},
1431: {'A': 0, 'T': 355, 'C': 0, 'G': 0},
1432: {'A': 0, 'T': 0, 'C': 0, 'G': 352},
1433: {'A': 0, 'T': 349, 'C': 0, 'G': 0},
1434: {'A': 0, 'T': 0, 'C': 0, 'G': 352},
1435: {'A': 350, 'T': 0, 'C': 0, 'G': 0},
1436: {'A': 351, 'T': 0, 'C': 0, 'G': 0},
1437: {'A': 352, 'T': 0, 'C': 0, 'G': 0},
1438: {'A': 0, 'T': 0, 'C': 0, 'G': 356},
1439: {'A': 0, 'T': 0, 'C': 0, 'G': 350},
```

```
1440: {'A': 0, 'T': 0, 'C': 0, 'G': 348},
1441: {'A': 0, 'T': 0, 'C': 0, 'G': 343},
1442: {'A': 338, 'T': 0, 'C': 0, 'G': 0},
664: {'A': 1, 'T': 0, 'C': 247, 'G': 2},
665: {'A': 0, 'T': 244, 'C': 1, 'G': 2},
666: {'A': 0, 'T': 243, 'C': 2, 'G': 1},
667: {'A': 0, 'T': 1, 'C': 2, 'G': 236},
668: {'A': 0, 'T': 3, 'C': 1, 'G': 230},
669: {'A': 0, 'T': 2, 'C': 234, 'G': 1},
670: {'A': 0, 'T': 238, 'C': 0, 'G': 2},
671: {'A': 0, 'T': 0, 'C': 1, 'G': 239},
672: {'A': 0, 'T': 1, 'C': 238, 'G': 1},
673: {'A': 236, 'T': 1, 'C': 1, 'G': 1},
674: {'A': 0, 'T': 1, 'C': 1, 'G': 238},
675: {'A': 1, 'T': 0, 'C': 239, 'G': 1},
676: {'A': 238, 'T': 0, 'C': 1, 'G': 1},
677: {'A': 1, 'T': 239, 'C': 1, 'G': 1},
678: {'A': 1, 'T': 236, 'C': 1, 'G': 1},
679: {'A': 1, 'T': 237, 'C': 1, 'G': 0},
680: {'A': 1, 'T': 2, 'C': 236, 'G': 0},
681: {'A': 0, 'T': 3, 'C': 233, 'G': 0},
682: {'A': 233, 'T': 2, 'C': 1, 'G': 0},
683: {'A': 1, 'T': 1, 'C': 2, 'G': 234},
684: {'A': 2, 'T': 0, 'C': 232, 'G': 1},
685: {'A': 0, 'T': 0, 'C': 1, 'G': 229},
686: {'A': 2, 'T': 0, 'C': 1, 'G': 228},
687: {'A': 0, 'T': 0, 'C': 228, 'G': 2},
688: {'A': 0, 'T': 0, 'C': 226, 'G': 1},
689: {'A': 1, 'T': 2, 'C': 224, 'G': 1},
690: {'A': 0, 'T': 0, 'C': 235, 'G': 1},
691: {'A': 239, 'T': 1, 'C': 2, 'G': 0},
692: {'A': 0, 'T': 243, 'C': 3, 'G': 0},
693: {'A': 1, 'T': 0, 'C': 246, 'G': 0},
694: {'A': 1, 'T': 2, 'C': 243, 'G': 0},
695: {'A': 242, 'T': 1, 'C': 1, 'G': 1},
696: {'A': 0, 'T': 1, 'C': 2, 'G': 243},
697: {'A': 244, 'T': 1, 'C': 2, 'G': 0},
698: {'A': 244, 'T': 0, 'C': 1, 'G': 1},
699: {'A': 1, 'T': 0, 'C': 0, 'G': 243},
700: {'A': 2, 'T': 0, 'C': 239, 'G': 0},
701: {'A': 2, 'T': 0, 'C': 243, 'G': 1},
702: {'A': 0, 'T': 244, 'C': 1, 'G': 1},
703: {'A': 0, 'T': 0, 'C': 2, 'G': 242},
704: {'A': 0, 'T': 2, 'C': 1, 'G': 244},
705: {'A': 0, 'T': 1, 'C': 242, 'G': 1},
706: {'A': 239, 'T': 0, 'C': 0, 'G': 2},
707: {'A': 0, 'T': 238, 'C': 1, 'G': 1},
708: {'A': 1, 'T': 0, 'C': 238, 'G': 1},
709: {'A': 1, 'T': 241, 'C': 0, 'G': 1},
710: {'A': 0, 'T': 242, 'C': 1, 'G': 0},
711: {'A': 0, 'T': 240, 'C': 1, 'G': 0},
712: {'A': 234, 'T': 3, 'C': 0, 'G': 0},
713: {'A': 0, 'T': 238, 'C': 0, 'G': 0},
714: {'A': 1, 'T': 1, 'C': 241, 'G': 0},
715: {'A': 238, 'T': 1, 'C': 1, 'G': 0},
716: {'A': 0, 'T': 1, 'C': 1, 'G': 237},
717: {'A': 1, 'T': 0, 'C': 242, 'G': 0},
718: {'A': 1, 'T': 0, 'C': 242, 'G': 1},
719: {'A': 239, 'T': 0, 'C': 2, 'G': 1},
720: {'A': 0, 'T': 238, 'C': 2, 'G': 0},
721: {'A': 1, 'T': 0, 'C': 1, 'G': 242},
722: {'A': 2, 'T': 243, 'C': 0, 'G': 0},
723: {'A': 1, 'T': 1, 'C': 0, 'G': 242},
724: {'A': 244, 'T': 2, 'C': 0, 'G': 1},
725: {'A': 244, 'T': 1, 'C': 0, 'G': 1},
726: {'A': 248, 'T': 0, 'C': 0, 'G': 1},
```

```
727: {'A': 2, 'T': 0, 'C': 251, 'G': 0},
728: {'A': 2, 'T': 0, 'C': 256, 'G': 0},
729: {'A': 1, 'T': 256, 'C': 1, 'G': 0},
730: {'A': 0, 'T': 0, 'C': 2, 'G': 260},
731: {'A': 0, 'T': 1, 'C': 2, 'G': 257},
732: {'A': 0, 'T': 1, 'C': 248, 'G': 1},
733: {'A': 0, 'T': 254, 'C': 0, 'G': 2},
734: {'A': 0, 'T': 0, 'C': 261, 'G': 1},
735: {'A': 0, 'T': 1, 'C': 260, 'G': 0},
736: {'A': 0, 'T': 1, 'C': 265, 'G': 0},
737: {'A': 0, 'T': 263, 'C': 2, 'G': 0},
738: {'A': 0, 'T': 0, 'C': 2, 'G': 261},
739: {'A': 0, 'T': 261, 'C': 1, 'G': 0},
740: {'A': 0, 'T': 1, 'C': 259, 'G': 1},
741: {'A': 0, 'T': 262, 'C': 0, 'G': 1},
742: {'A': 0, 'T': 1, 'C': 1, 'G': 257},
743: {'A': 0, 'T': 1, 'C': 254, 'G': 0},
744: {'A': 0, 'T': 257, 'C': 0, 'G': 1},
745: {'A': 0, 'T': 1, 'C': 1, 'G': 259},
746: {'A': 258, 'T': 1, 'C': 2, 'G': 0},
747: {'A': 1, 'T': 1, 'C': 0, 'G': 259},
748: {'A': 2, 'T': 0, 'C': 0, 'G': 255},
749: {'A': 1, 'T': 241, 'C': 1, 'G': 2},
750: {'A': 0, 'T': 0, 'C': 1, 'G': 252},
751: {'A': 1, 'T': 1, 'C': 0, 'G': 250},
752: {'A': 0, 'T': 2, 'C': 0, 'G': 250},
753: {'A': 251, 'T': 0, 'C': 0, 'G': 3},
754: {'A': 0, 'T': 254, 'C': 0, 'G': 2},
755: {'A': 1, 'T': 258, 'C': 0, 'G': 2},
756: {'A': 2, 'T': 1, 'C': 0, 'G': 257},
757: {'A': 1, 'T': 2, 'C': 0, 'G': 255},
758: {'A': 253, 'T': 1, 'C': 0, 'G': 1},
759: {'A': 0, 'T': 1, 'C': 1, 'G': 256},
760: {'A': 252, 'T': 1, 'C': 1, 'G': 1},
761: {'A': 1, 'T': 255, 'C': 0, 'G': 1},
762: {'A': 251, 'T': 0, 'C': 0, 'G': 2},
763: {'A': 2, 'T': 0, 'C': 0, 'G': 260},
764: {'A': 0, 'T': 1, 'C': 1, 'G': 256},
317: {'A': 0, 'T': 320, 'C': 2, 'G': 48},
318: {'A': 1, 'T': 0, 'C': 2, 'G': 375},
319: {'A': 40, 'T': 1, 'C': 3, 'G': 334},
320: {'A': 336, 'T': 0, 'C': 2, 'G': 39},
321: {'A': 1, 'T': 33, 'C': 0, 'G': 345},
322: {'A': 2, 'T': 358, 'C': 0, 'G': 3},
323: {'A': 0, 'T': 369, 'C': 1, 'G': 3},
324: {'A': 1, 'T': 352, 'C': 1, 'G': 25},
325: {'A': 0, 'T': 1, 'C': 1, 'G': 370},
326: {'A': 0, 'T': 3, 'C': 17, 'G': 354},
327: {'A': 0, 'T': 18, 'C': 355, 'G': 1},
328: {'A': 0, 'T': 362, 'C': 0, 'G': 12},
329: {'A': 0, 'T': 8, 'C': 1, 'G': 363},
330: {'A': 0, 'T': 362, 'C': 0, 'G': 8},
331: {'A': 0, 'T': 0, 'C': 1, 'G': 369},
332: {'A': 0, 'T': 2, 'C': 0, 'G': 375},
333: {'A': 0, 'T': 1, 'C': 1, 'G': 374},
334: {'A': 0, 'T': 3, 'C': 373, 'G': 2},
335: {'A': 0, 'T': 372, 'C': 1, 'G': 1},
336: {'A': 0, 'T': 2, 'C': 360, 'G': 2},
337: {'A': 0, 'T': 360, 'C': 0, 'G': 1},
338: {'A': 0, 'T': 362, 'C': 2, 'G': 0},
339: {'A': 1, 'T': 2, 'C': 363, 'G': 0},
340: {'A': 353, 'T': 1, 'C': 1, 'G': 1},
341: {'A': 0, 'T': 358, 'C': 1, 'G': 0},
342: {'A': 0, 'T': 2, 'C': 356, 'G': 1},
343: {'A': 0, 'T': 351, 'C': 2, 'G': 1},
344: {'A': 1, 'T': 0, 'C': 353, 'G': 1},
```

```
345: {'A': 0, 'T': 1, 'C': 353, 'G': 0},
346: {'A': 1, 'T': 1, 'C': 352, 'G': 0},
347: {'A': 334, 'T': 1, 'C': 2, 'G': 0},
348: {'A': 0, 'T': 1, 'C': 354, 'G': 0},
349: {'A': 0, 'T': 2, 'C': 356, 'G': 0},
350: {'A': 0, 'T': 353, 'C': 3, 'G': 0},
351: {'A': 3, 'T': 0, 'C': 350, 'G': 0},
352: {'A': 346, 'T': 3, 'C': 1, 'G': 0},
353: {'A': 0, 'T': 343, 'C': 3, 'G': 0},
354: {'A': 1, 'T': 2, 'C': 347, 'G': 0},
355: {'A': 345, 'T': 0, 'C': 2, 'G': 0},
356: {'A': 343, 'T': 0, 'C': 1, 'G': 0},
357: {'A': 336, 'T': 1, 'C': 1, 'G': 1},
358: {'A': 1, 'T': 0, 'C': 1, 'G': 337},
359: {'A': 1, 'T': 0, 'C': 2, 'G': 329},
360: {'A': 1, 'T': 0, 'C': 330, 'G': 1},
361: {'A': 2, 'T': 1, 'C': 0, 'G': 332},
362: {'A': 0, 'T': 2, 'C': 1, 'G': 332},
363: {'A': 0, 'T': 324, 'C': 2, 'G': 0},
364: {'A': 1, 'T': 1, 'C': 333, 'G': 0},
365: {'A': 329, 'T': 0, 'C': 0, 'G': 2},
366: {'A': 0, 'T': 0, 'C': 0, 'G': 334},
367: {'A': 0, 'T': 1, 'C': 1, 'G': 328},
368: {'A': 1, 'T': 0, 'C': 324, 'G': 0},
369: {'A': 320, 'T': 0, 'C': 0, 'G': 1},
370: {'A': 1, 'T': 0, 'C': 0, 'G': 321},
371: {'A': 314, 'T': 0, 'C': 1, 'G': 1},
372: {'A': 1, 'T': 0, 'C': 323, 'G': 0},
373: {'A': 321, 'T': 0, 'C': 0, 'G': 1},
374: {'A': 0, 'T': 0, 'C': 1, 'G': 320},
375: {'A': 1, 'T': 0, 'C': 315, 'G': 1},
376: {'A': 0, 'T': 1, 'C': 1, 'G': 310},
377: {'A': 1, 'T': 297, 'C': 0, 'G': 0},
378: {'A': 0, 'T': 0, 'C': 302, 'G': 1},
379: {'A': 0, 'T': 0, 'C': 1, 'G': 298},
380: {'A': 0, 'T': 0, 'C': 0, 'G': 303},
381: {'A': 1, 'T': 1, 'C': 0, 'G': 297},
382: {'A': 0, 'T': 1, 'C': 294, 'G': 0},
383: {'A': 0, 'T': 294, 'C': 0, 'G': 1},
384: {'A': 0, 'T': 0, 'C': 296, 'G': 1},
385: {'A': 0, 'T': 0, 'C': 296, 'G': 1},
386: {'A': 286, 'T': 0, 'C': 1, 'G': 0},
387: {'A': 0, 'T': 1, 'C': 0, 'G': 287},
388: {'A': 0, 'T': 0, 'C': 1, 'G': 290},
389: {'A': 0, 'T': 279, 'C': 1, 'G': 0},
390: {'A': 279, 'T': 0, 'C': 0, 'G': 0},
391: {'A': 0, 'T': 0, 'C': 0, 'G': 284},
392: {'A': 0, 'T': 0, 'C': 0, 'G': 284},
393: {'A': 0, 'T': 1, 'C': 0, 'G': 271},
394: {'A': 1, 'T': 0, 'C': 0, 'G': 270},
395: {'A': 264, 'T': 0, 'C': 0, 'G': 1},
396: {'A': 0, 'T': 0, 'C': 262, 'G': 1},
397: {'A': 0, 'T': 0, 'C': 0, 'G': 259},
398: {'A': 257, 'T': 0, 'C': 0, 'G': 2},
399: {'A': 2, 'T': 0, 'C': 0, 'G': 255},
400: {'A': 254, 'T': 0, 'C': 1, 'G': 1},
401: {'A': 1, 'T': 253, 'C': 0, 'G': 1},
402: {'A': 1, 'T': 1, 'C': 256, 'G': 0},
403: {'A': 53, 'T': 0, 'C': 1, 'G': 197},
404: {'A': 1, 'T': 252, 'C': 0, 'G': 1},
405: {'A': 0, 'T': 2, 'C': 248, 'G': 0},
406: {'A': 0, 'T': 35, 'C': 220, 'G': 0},
407: {'A': 2, 'T': 1, 'C': 1, 'G': 243},
408: {'A': 0, 'T': 2, 'C': 1, 'G': 242},
409: {'A': 241, 'T': 0, 'C': 2, 'G': 2},
410: {'A': 2, 'T': 240, 'C': 1, 'G': 1},
```

```
411: {'A': 1, 'T': 2, 'C': 236, 'G': 2},
412: {'A': 238, 'T': 1, 'C': 1, 'G': 2},
413: {'A': 238, 'T': 0, 'C': 1, 'G': 1},
414: {'A': 3, 'T': 237, 'C': 0, 'G': 0},
415: {'A': 24, 'T': 2, 'C': 1, 'G': 213},
416: {'A': 1, 'T': 1, 'C': 1, 'G': 238},
417: {'A': 236, 'T': 0, 'C': 0, 'G': 2},
1109: {'A': 1, 'T': 0, 'C': 192, 'G': 0},
1110: {'A': 1, 'T': 0, 'C': 193, 'G': 0},
1111: {'A': 191, 'T': 0, 'C': 1, 'G': 0},
1112: {'A': 2, 'T': 180, 'C': 0, 'G': 0},
1113: {'A': 0, 'T': 1, 'C': 191, 'G': 0},
1114: {'A': 186, 'T': 0, 'C': 2, 'G': 0},
1115: {'A': 1, 'T': 0, 'C': 204, 'G': 0},
1116: {'A': 0, 'T': 198, 'C': 1, 'G': 0},
1117: {'A': 0, 'T': 1, 'C': 1, 'G': 203},
1118: {'A': 1, 'T': 0, 'C': 200, 'G': 1},
1119: {'A': 0, 'T': 201, 'C': 2, 'G': 0},
1120: {'A': 0, 'T': 2, 'C': 0, 'G': 198},
1121: {'A': 198, 'T': 0, 'C': 0, 'G': 2},
1122: {'A': 1, 'T': 0, 'C': 1, 'G': 207},
1123: {'A': 0, 'T': 1, 'C': 0, 'G': 204},
1124: {'A': 0, 'T': 186, 'C': 0, 'G': 2},
1125: {'A': 169, 'T': 1, 'C': 0, 'G': 0},
1126: {'A': 1, 'T': 0, 'C': 168, 'G': 1},
1127: {'A': 162, 'T': 0, 'C': 1, 'G': 1},
1128: {'A': 0, 'T': 1, 'C': 218, 'G': 0},
1129: {'A': 1, 'T': 0, 'C': 217, 'G': 0},
1130: {'A': 0, 'T': 0, 'C': 210, 'G': 0},
1131: {'A': 1, 'T': 0, 'C': 0, 'G': 219},
1132: {'A': 0, 'T': 0, 'C': 1, 'G': 204},
1133: {'A': 0, 'T': 173, 'C': 1, 'G': 2},
1134: {'A': 165, 'T': 1, 'C': 1, 'G': 0},
1135: {'A': 0, 'T': 0, 'C': 223, 'G': 1},
1136: {'A': 0, 'T': 0, 'C': 221, 'G': 1},
1137: {'A': 0, 'T': 1, 'C': 218, 'G': 0},
1138: {'A': 1, 'T': 0, 'C': 201, 'G': 0},
1139: {'A': 0, 'T': 220, 'C': 2, 'G': 0},
1140: {'A': 0, 'T': 199, 'C': 1, 'G': 0},
1141: {'A': 0, 'T': 1, 'C': 201, 'G': 0},
1142: {'A': 0, 'T': 0, 'C': 2, 'G': 196},
1143: {'A': 0, 'T': 1, 'C': 185, 'G': 1},
1144: {'A': 239, 'T': 1, 'C': 1, 'G': 0},
1145: {'A': 242, 'T': 0, 'C': 1, 'G': 1},
1146: {'A': 1, 'T': 0, 'C': 1, 'G': 241},
1147: {'A': 0, 'T': 0, 'C': 249, 'G': 1},
1148: {'A': 1, 'T': 0, 'C': 250, 'G': 0},
1149: {'A': 250, 'T': 0, 'C': 1, 'G': 0},
1150: {'A': 253, 'T': 0, 'C': 0, 'G': 1},
1151: {'A': 254, 'T': 0, 'C': 1, 'G': 0},
1154: {'A': 252, 'T': 1, 'C': 0, 'G': 0},
1155: {'A': 3, 'T': 245, 'C': 0, 'G': 0},
1156: {'A': 0, 'T': 1, 'C': 0, 'G': 248},
1157: {'A': 242, 'T': 1, 'C': 0, 'G': 1},
1158: {'A': 2, 'T': 243, 'C': 0, 'G': 0},
1159: {'A': 0, 'T': 2, 'C': 247, 'G': 0},
1160: {'A': 248, 'T': 0, 'C': 1, 'G': 1},
1161: {'A': 2, 'T': 0, 'C': 0, 'G': 251},
1162: {'A': 0, 'T': 1, 'C': 0, 'G': 253},
1163: {'A': 0, 'T': 0, 'C': 1, 'G': 253},
1164: {'A': 257, 'T': 0, 'C': 0, 'G': 1},
1165: {'A': 1, 'T': 0, 'C': 0, 'G': 251},
1166: {'A': 0, 'T': 237, 'C': 0, 'G': 2},
1167: {'A': 1, 'T': 1, 'C': 0, 'G': 257},
1168: {'A': 1, 'T': 0, 'C': 0, 'G': 260},
1169: {'A': 263, 'T': 0, 'C': 0, 'G': 2},
```

```
1170: {'A': 258, 'T': 0, 'C': 0, 'G': 0},
1171: {'A': 1, 'T': 0, 'C': 264, 'G': 1},
1172: {'A': 0, 'T': 0, 'C': 268, 'G': 1},
1173: {'A': 1, 'T': 266, 'C': 1, 'G': 0},
1174: {'A': 1, 'T': 1, 'C': 0, 'G': 265},
1175: {'A': 265, 'T': 0, 'C': 1, 'G': 1},
1176: {'A': 1, 'T': 0, 'C': 2, 'G': 262},
1177: {'A': 0, 'T': 1, 'C': 259, 'G': 1},
1178: {'A': 0, 'T': 259, 'C': 0, 'G': 1},
1179: {'A': 1, 'T': 0, 'C': 266, 'G': 0},
1180: {'A': 0, 'T': 0, 'C': 0, 'G': 272},
1181: {'A': 271, 'T': 0, 'C': 1, 'G': 0},
1182: {'A': 0, 'T': 2, 'C': 0, 'G': 266},
1183: {'A': 0, 'T': 0, 'C': 269, 'G': 1},
1184: {'A': 0, 'T': 0, 'C': 267, 'G': 2},
1185: {'A': 1, 'T': 1, 'C': 267, 'G': 0},
1195: {'A': 0, 'T': 1, 'C': 272, 'G': 1},
1036: {'A': 0, 'T': 0, 'C': 0, 'G': 156},
1037: {'A': 149, 'T': 0, 'C': 0, 'G': 0},
1038: {'A': 0, 'T': 0, 'C': 0, 'G': 150},
1039: {'A': 0, 'T': 0, 'C': 0, 'G': 146},
1040: {'A': 143, 'T': 1, 'C': 0, 'G': 0},
1041: {'A': 0, 'T': 1, 'C': 0, 'G': 145},
1042: {'A': 0, 'T': 0, 'C': 0, 'G': 146},
1043: {'A': 146, 'T': 0, 'C': 0, 'G': 0},
1044: {'A': 141, 'T': 0, 'C': 0, 'G': 0},
1045: {'A': 0, 'T': 0, 'C': 0, 'G': 138},
1046: {'A': 144, 'T': 0, 'C': 0, 'G': 0},
1047: {'A': 0, 'T': 0, 'C': 0, 'G': 145},
1048: {'A': 145, 'T': 0, 'C': 0, 'G': 0},
1049: {'A': 146, 'T': 0, 'C': 0, 'G': 0},
1050: {'A': 0, 'T': 0, 'C': 0, 'G': 146},
1051: {'A': 0, 'T': 140, 'C': 0, 'G': 0},
1052: {'A': 0, 'T': 148, 'C': 0, 'G': 0},
1053: {'A': 0, 'T': 150, 'C': 0, 'G': 0},
1054: {'A': 154, 'T': 0, 'C': 0, 'G': 0},
1055: {'A': 154, 'T': 0, 'C': 0, 'G': 0},
1056: {'A': 0, 'T': 0, 'C': 0, 'G': 157},
1057: {'A': 162, 'T': 0, 'C': 0, 'G': 0},
1058: {'A': 160, 'T': 0, 'C': 0, 'G': 0},
1059: {'A': 0, 'T': 0, 'C': 0, 'G': 161},
1060: {'A': 0, 'T': 0, 'C': 160, 'G': 0},
1061: {'A': 163, 'T': 0, 'C': 0, 'G': 0},
1062: {'A': 164, 'T': 0, 'C': 0, 'G': 0},
1063: {'A': 0, 'T': 163, 'C': 0, 'G': 0},
1064: {'A': 0, 'T': 1, 'C': 0, 'G': 166},
1065: {'A': 0, 'T': 0, 'C': 0, 'G': 169},
1066: {'A': 0, 'T': 0, 'C': 0, 'G': 165},
1067: {'A': 166, 'T': 0, 'C': 0, 'G': 0},
1068: {'A': 165, 'T': 0, 'C': 0, 'G': 0},
1069: {'A': 0, 'T': 0, 'C': 0, 'G': 165},
1070: {'A': 170, 'T': 0, 'C': 0, 'G': 0},
1071: {'A': 170, 'T': 0, 'C': 0, 'G': 0},
1072: {'A': 0, 'T': 0, 'C': 0, 'G': 169},
1073: {'A': 167, 'T': 0, 'C': 0, 'G': 0},
1074: {'A': 0, 'T': 0, 'C': 173, 'G': 0},
1075: {'A': 0, 'T': 174, 'C': 0, 'G': 0},
1076: {'A': 0, 'T': 0, 'C': 0, 'G': 176},
1077: {'A': 0, 'T': 0, 'C': 0, 'G': 179},
1078: {'A': 0, 'T': 0, 'C': 0, 'G': 175},
1079: {'A': 0, 'T': 0, 'C': 0, 'G': 173},
1080: {'A': 0, 'T': 0, 'C': 174, 'G': 0},
1081: {'A': 0, 'T': 173, 'C': 0, 'G': 0},
1082: {'A': 0, 'T': 0, 'C': 172, 'G': 0},
1083: {'A': 178, 'T': 0, 'C': 0, 'G': 0},
1084: {'A': 179, 'T': 0, 'C': 0, 'G': 0},
```

```
1085: {'A': 179, 'T': 0, 'C': 0, 'G': 0},
1086: {'A': 0, 'T': 0, 'C': 0, 'G': 178},
1087: {'A': 0, 'T': 0, 'C': 0, 'G': 173},
1088: {'A': 177, 'T': 0, 'C': 0, 'G': 0},
1089: {'A': 172, 'T': 0, 'C': 0, 'G': 0},
1090: {'A': 0, 'T': 0, 'C': 174, 'G': 0},
1091: {'A': 169, 'T': 0, 'C': 0, 'G': 0},
1092: {'A': 0, 'T': 0, 'C': 0, 'G': 174},
1093: {'A': 0, 'T': 0, 'C': 170, 'G': 0},
1094: {'A': 0, 'T': 172, 'C': 0, 'G': 0},
1095: {'A': 169, 'T': 0, 'C': 0, 'G': 0},
1096: {'A': 0, 'T': 0, 'C': 176, 'G': 0},
1097: {'A': 0, 'T': 176, 'C': 0, 'G': 0},
1098: {'A': 0, 'T': 0, 'C': 177, 'G': 0},
1099: {'A': 0, 'T': 177, 'C': 0, 'G': 0},
1100: {'A': 0, 'T': 183, 'C': 1, 'G': 0},
1101: {'A': 0, 'T': 0, 'C': 0, 'G': 180},
1102: {'A': 0, 'T': 0, 'C': 182, 'G': 0},
1103: {'A': 0, 'T': 1, 'C': 185, 'G': 0},
1104: {'A': 0, 'T': 181, 'C': 0, 'G': 0},
1105: {'A': 186, 'T': 0, 'C': 0, 'G': 1},
1106: {'A': 185, 'T': 0, 'C': 1, 'G': 0},
1107: {'A': 189, 'T': 0, 'C': 1, 'G': 0},
1108: {'A': 181, 'T': 1, 'C': 0, 'G': 0},
79: {'A': 18, 'T': 187, 'C': 13, 'G': 20},
80: {'A': 189, 'T': 19, 'C': 13, 'G': 18},
81: {'A': 24, 'T': 199, 'C': 9, 'G': 15},
82: {'A': 24, 'T': 17, 'C': 13, 'G': 195},
83: {'A': 203, 'T': 21, 'C': 6, 'G': 22},
84: {'A': 25, 'T': 201, 'C': 10, 'G': 16},
85: {'A': 17, 'T': 26, 'C': 6, 'G': 204},
86: {'A': 22, 'T': 203, 'C': 10, 'G': 19},
87: {'A': 17, 'T': 25, 'C': 7, 'G': 211},
88: {'A': 21, 'T': 13, 'C': 211, 'G': 19},
89: {'A': 22, 'T': 217, 'C': 12, 'G': 16},
90: {'A': 21, 'T': 19, 'C': 6, 'G': 222},
91: {'A': 20, 'T': 18, 'C': 209, 'G': 23},
92: {'A': 16, 'T': 20, 'C': 9, 'G': 234},
93: {'A': 228, 'T': 22, 'C': 10, 'G': 19},
94: {'A': 226, 'T': 22, 'C': 12, 'G': 18},
95: {'A': 22, 'T': 232, 'C': 12, 'G': 19},
96: {'A': 19, 'T': 23, 'C': 9, 'G': 231},
97: {'A': 22, 'T': 234, 'C': 9, 'G': 17},
98: {'A': 236, 'T': 20, 'C': 11, 'G': 13},
99: {'A': 18, 'T': 20, 'C': 233, 'G': 16},
100: {'A': 12, 'T': 18, 'C': 226, 'G': 28},
101: {'A': 245, 'T': 16, 'C': 9, 'G': 12},
102: {'A': 1, 'T': 0, 'C': 228, 'G': 0},
103: {'A': 1, 'T': 1, 'C': 227, 'G': 0},
104: {'A': 230, 'T': 0, 'C': 2, 'G': 0},
105: {'A': 0, 'T': 1, 'C': 1, 'G': 231},
106: {'A': 231, 'T': 0, 'C': 2, 'G': 0},
107: {'A': 0, 'T': 1, 'C': 233, 'G': 1},
108: {'A': 3, 'T': 0, 'C': 231, 'G': 0},
109: {'A': 236, 'T': 1, 'C': 1, 'G': 0},
110: {'A': 0, 'T': 238, 'C': 1, 'G': 1},
111: {'A': 3, 'T': 0, 'C': 0, 'G': 241},
112: {'A': 0, 'T': 2, 'C': 0, 'G': 245},
113: {'A': 246, 'T': 0, 'C': 0, 'G': 2},
114: {'A': 0, 'T': 1, 'C': 250, 'G': 1},
115: {'A': 1, 'T': 0, 'C': 0, 'G': 253},
116: {'A': 0, 'T': 253, 'C': 2, 'G': 0},
117: {'A': 0, 'T': 2, 'C': 0, 'G': 259},
118: {'A': 0, 'T': 2, 'C': 0, 'G': 256},
119: {'A': 0, 'T': 0, 'C': 258, 'G': 1},
120: {'A': 0, 'T': 1, 'C': 260, 'G': 1},
```

```
121: {'A': 1, 'T': 0, 'C': 1, 'G': 264},
122: {'A': 1, 'T': 263, 'C': 1, 'G': 1},
123: {'A': 0, 'T': 2, 'C': 0, 'G': 263},
124: {'A': 0, 'T': 1, 'C': 271, 'G': 2},
125: {'A': 0, 'T': 274, 'C': 1, 'G': 0},
126: {'A': 1, 'T': 1, 'C': 274, 'G': 0},
866: {'A': 0, 'T': 2, 'C': 2, 'G': 338},
867: {'A': 0, 'T': 0, 'C': 1, 'G': 348},
868: {'A': 1, 'T': 0, 'C': 315, 'G': 1},
869: {'A': 322, 'T': 0, 'C': 0, 'G': 2},
870: {'A': 0, 'T': 0, 'C': 1, 'G': 339},
871: {'A': 1, 'T': 0, 'C': 335, 'G': 2},
872: {'A': 1, 'T': 1, 'C': 1, 'G': 323},
873: {'A': 0, 'T': 292, 'C': 0, 'G': 3},
874: {'A': 1, 'T': 1, 'C': 1, 'G': 313},
875: {'A': 319, 'T': 0, 'C': 0, 'G': 3},
876: {'A': 1, 'T': 1, 'C': 1, 'G': 339},
877: {'A': 0, 'T': 1, 'C': 332, 'G': 2},
878: {'A': 1, 'T': 331, 'C': 1, 'G': 2},
879: {'A': 0, 'T': 1, 'C': 1, 'G': 341},
880: {'A': 1, 'T': 0, 'C': 339, 'G': 1},
881: {'A': 341, 'T': 1, 'C': 1, 'G': 1},
882: {'A': 1, 'T': 0, 'C': 1, 'G': 342},
883: {'A': 0, 'T': 0, 'C': 336, 'G': 2},
884: {'A': 1, 'T': 0, 'C': 1, 'G': 335},
885: {'A': 0, 'T': 0, 'C': 1, 'G': 341},
886: {'A': 1, 'T': 0, 'C': 325, 'G': 1},
887: {'A': 321, 'T': 0, 'C': 0, 'G': 3},
888: {'A': 0, 'T': 0, 'C': 0, 'G': 331},
889: {'A': 1, 'T': 0, 'C': 1, 'G': 339},
890: {'A': 278, 'T': 0, 'C': 0, 'G': 1},
891: {'A': 0, 'T': 0, 'C': 1, 'G': 299},
892: {'A': 0, 'T': 1, 'C': 270, 'G': 1},
893: {'A': 1, 'T': 289, 'C': 0, 'G': 0},
894: {'A': 0, 'T': 317, 'C': 1, 'G': 1},
895: {'A': 0, 'T': 1, 'C': 294, 'G': 0},
896: {'A': 0, 'T': 307, 'C': 1, 'G': 0},
897: {'A': 1, 'T': 2, 'C': 308, 'G': 0},
898: {'A': 297, 'T': 1, 'C': 1, 'G': 0},
899: {'A': 0, 'T': 301, 'C': 0, 'G': 1},
900: {'A': 0, 'T': 0, 'C': 2, 'G': 301},
901: {'A': 2, 'T': 0, 'C': 306, 'G': 1},
902: {'A': 307, 'T': 1, 'C': 0, 'G': 2},
903: {'A': 2, 'T': 0, 'C': 0, 'G': 311},
904: {'A': 313, 'T': 0, 'C': 1, 'G': 0},
905: {'A': 323, 'T': 0, 'C': 0, 'G': 1},
906: {'A': 0, 'T': 0, 'C': 1, 'G': 297},
907: {'A': 1, 'T': 0, 'C': 290, 'G': 1},
908: {'A': 1, 'T': 0, 'C': 0, 'G': 286},
909: {'A': 0, 'T': 0, 'C': 1, 'G': 311},
910: {'A': 1, 'T': 1, 'C': 272, 'G': 0},
911: {'A': 0, 'T': 280, 'C': 0, 'G': 2},
912: {'A': 0, 'T': 0, 'C': 0, 'G': 284},
913: {'A': 0, 'T': 0, 'C': 0, 'G': 305},
914: {'A': 0, 'T': 0, 'C': 262, 'G': 0},
915: {'A': 0, 'T': 0, 'C': 0, 'G': 270},
917: {'A': 0, 'T': 251, 'C': 0, 'G': 0},
918: {'A': 0, 'T': 0, 'C': 0, 'G': 265},
919: {'A': 0, 'T': 0, 'C': 0, 'G': 285},
920: {'A': 257, 'T': 0, 'C': 0, 'G': 0},
921: {'A': 0, 'T': 0, 'C': 0, 'G': 269},
923: {'A': 0, 'T': 0, 'C': 260, 'G': 0},
924: {'A': 0, 'T': 0, 'C': 285, 'G': 1},
925: {'A': 285, 'T': 0, 'C': 0, 'G': 0},
926: {'A': 294, 'T': 0, 'C': 0, 'G': 0},
927: {'A': 1, 'T': 0, 'C': 276, 'G': 0},
```

```
928: {'A': 283, 'T': 0, 'C': 0, 'G': 1},
929: {'A': 295, 'T': 0, 'C': 0, 'G': 1},
930: {'A': 0, 'T': 0, 'C': 0, 'G': 270},
931: {'A': 273, 'T': 0, 'C': 0, 'G': 2},
932: {'A': 0, 'T': 265, 'C': 1, 'G': 0},
933: {'A': 0, 'T': 1, 'C': 273, 'G': 0},
934: {'A': 0, 'T': 1, 'C': 279, 'G': 0},
935: {'A': 0, 'T': 276, 'C': 1, 'G': 0},
936: {'A': 0, 'T': 1, 'C': 278, 'G': 0},
937: {'A': 0, 'T': 0, 'C': 291, 'G': 0},
938: {'A': 267, 'T': 0, 'C': 0, 'G': 0},
939: {'A': 0, 'T': 3, 'C': 0, 'G': 273},
940: {'A': 0, 'T': 0, 'C': 0, 'G': 281},
941: {'A': 258, 'T': 0, 'C': 1, 'G': 0},
942: {'A': 1, 'T': 0, 'C': 0, 'G': 257},
943: {'A': 0, 'T': 0, 'C': 252, 'G': 1},
944: {'A': 251, 'T': 0, 'C': 0, 'G': 0},
945: {'A': 1, 'T': 0, 'C': 0, 'G': 254},
946: {'A': 0, 'T': 0, 'C': 255, 'G': 1},
947: {'A': 251, 'T': 0, 'C': 1, 'G': 0},
948: {'A': 0, 'T': 0, 'C': 0, 'G': 249},
949: {'A': 0, 'T': 0, 'C': 0, 'G': 252},
951: {'A': 0, 'T': 1, 'C': 0, 'G': 228},
953: {'A': 0, 'T': 224, 'C': 0, 'G': 1},
954: {'A': 0, 'T': 0, 'C': 1, 'G': 232},
955: {'A': 0, 'T': 0, 'C': 0, 'G': 231},
956: {'A': 222, 'T': 0, 'C': 0, 'G': 0},
957: {'A': 0, 'T': 1, 'C': 0, 'G': 221},
960: {'A': 1, 'T': 0, 'C': 0, 'G': 215},
916: {'A': 244, 'T': 0, 'C': 0, 'G': 0},
922: {'A': 0, 'T': 234, 'C': 0, 'G': 1},
950: {'A': 212, 'T': 0, 'C': 1, 'G': 0},
952: {'A': 215, 'T': 0, 'C': 0, 'G': 1},
958: {'A': 0, 'T': 0, 'C': 206, 'G': 1},
959: {'A': 0, 'T': 0, 'C': 0, 'G': 207},
961: {'A': 0, 'T': 0, 'C': 203, 'G': 1},
246: {'A': 317, 'T': 2, 'C': 70, 'G': 0},
247: {'A': 73, 'T': 1, 'C': 2, 'G': 310},
248: {'A': 0, 'T': 1, 'C': 3, 'G': 377},
249: {'A': 298, 'T': 0, 'C': 0, 'G': 79},
250: {'A': 370, 'T': 0, 'C': 0, 'G': 2},
251: {'A': 89, 'T': 0, 'C': 0, 'G': 286},
252: {'A': 2, 'T': 0, 'C': 275, 'G': 94},
253: {'A': 2, 'T': 276, 'C': 98, 'G': 0},
254: {'A': 1, 'T': 105, 'C': 0, 'G': 273},
255: {'A': 270, 'T': 0, 'C': 2, 'G': 110},
256: {'A': 383, 'T': 2, 'C': 0, 'G': 0},
257: {'A': 120, 'T': 1, 'C': 0, 'G': 267},
258: {'A': 2, 'T': 2, 'C': 0, 'G': 382},
259: {'A': 256, 'T': 0, 'C': 0, 'G': 127},
260: {'A': 131, 'T': 0, 'C': 0, 'G': 254},
261: {'A': 1, 'T': 0, 'C': 0, 'G': 382},
262: {'A': 3, 'T': 208, 'C': 0, 'G': 136},
263: {'A': 0, 'T': 131, 'C': 0, 'G': 238},
264: {'A': 0, 'T': 1, 'C': 246, 'G': 141},
265: {'A': 0, 'T': 3, 'C': 147, 'G': 240},
266: {'A': 0, 'T': 238, 'C': 0, 'G': 153},
267: {'A': 0, 'T': 155, 'C': 236, 'G': 0},
268: {'A': 0, 'T': 230, 'C': 157, 'G': 3},
269: {'A': 0, 'T': 166, 'C': 1, 'G': 229},
270: {'A': 0, 'T': 0, 'C': 2, 'G': 389},
271: {'A': 220, 'T': 3, 'C': 0, 'G': 165},
272: {'A': 168, 'T': 0, 'C': 221, 'G': 2},
273: {'A': 1, 'T': 1, 'C': 391, 'G': 2},
274: {'A': 2, 'T': 1, 'C': 182, 'G': 212},
275: {'A': 2, 'T': 205, 'C': 2, 'G': 188},
```

```
276: {'A': 1, 'T': 196, 'C': 208, 'G': 1},
277: {'A': 0, 'T': 205, 'C': 198, 'G': 2},
278: {'A': 0, 'T': 199, 'C': 1, 'G': 199},
279: {'A': 1, 'T': 0, 'C': 195, 'G': 204},
280: {'A': 184, 'T': 2, 'C': 190, 'G': 0},
281: {'A': 184, 'T': 0, 'C': 193, 'G': 1},
282: {'A': 0, 'T': 1, 'C': 402, 'G': 0},
283: {'A': 2, 'T': 0, 'C': 402, 'G': 1},
284: {'A': 0, 'T': 0, 'C': 412, 'G': 0},
285: {'A': 2, 'T': 0, 'C': 228, 'G': 175},
...}
```

(g) Summarize nucleotide counts of all positions for all genes from Clone-seq data

In [ ]:
```
df_summary=pd.DataFrame()
for i in df_clone_seq["RNAME"].unique():
    count_dict_x = pos_nucl_count_all(df_clone_seq[df_clone_seq["RNAME"]==i])
    count_dict_x=pd.DataFrame(count_dict_x).T
    count_dict_x["gene"]=i
    df_summary=pd.concat([df_summary,count_dict_x])

df_summary.reset_index(inplace=True)
df_summary.rename(columns={'index': 'pos'}, inplace=True)
df_summary
```

Out[ ]:

	pos	A	T	C	G	gene
0	1152	1	0	1	255	7696
1	1153	1	243	0	1	7696
2	1186	1	1	0	267	7696
3	1187	0	2	269	0	7696
4	1188	266	1	2	0	7696
...	...	...	...	...	...	...
15726	1315	0	0	11	0	12438
15727	1316	10	0	0	0	12438
15728	1317	0	0	0	6	12438
15729	1318	0	0	0	4	12438
15730	1319	0	2	0	0	12438

15731 rows × 6 columns

In [ ]:
```
# df_summary[(df_summary["gene"]=="7696") & (df_summary["pos"]=="415")]
df_summary.dtypes
```

Out[ ]:
```
pos       int64
A         int64
T         int64
C         int64
G         int64
gene     object
dtype: object
```

In [ ]:
```
df_summary[(df_summary["gene"]=="7696") & (df_summary["pos"]==415)]
```

Out[ ]:

	pos	A	T	C	G	gene
664	415	24	2	1	213	7696

In [ ]:
```
df_summary[(df_summary["gene"]=="1136") & (df_summary["pos"]==409)]
```

Out[ ]:

	pos	A	T	C	G	gene
1901	409	0	16	44	0	1136

(h) Expected frequency

In [ ]:
```
### YOUR SOLUTION STARTS HERE ###
mut_list=open("data/Attempted_Muts.txt", "r").read().split("\n")
mut_list
```

Out[ ]:
```
['7696_G388A',
 '7696_G403A',
 '7696_C406T',
 '7696_G415A',
 '7696_A451G',
 '8466_T148A',
 '13668_G328A',
 '5174_G1093A',
 '5174_A1208T',
 '8593_A1204T',
 '8833_C380A',
 '8833_G482A',
 '8833_G493A',
 '8833_G511C',
 '8833_C568T',
 '8522_G142C',
 '8522_G163A',
 '8522_G178T',
 '8522_C274T',
 '8522_G302A',
 '8522_G364A',
 '8522_G454A',
 '8522_A533G',
 '8522_G579T',
 '8522_A643G',
 '8522_G763A',
 '8522_C764T',
 '8522_G835A',
 '8522_C851T',
 '8522_T860C',
 '8522_G914A',
 '8522_C926T',
 '8522_G934A',
 '8522_A959G',
 '8522_A967G',
 '9938_G340A',
 '9938_C347G',
 '9938_G373A',
 '9938_C469T',
 '804_A251G',
 '804_A367G',
 '10062_C184T',
 '10062_G215A',
 '10062_A265C',
 '10062_C769G',
 '10062_T1058C',
 '1136_C241T',
 '1136_A400G',
 '1136_C409T',
 '1136_G433A',
 '6936_A470G',
 '12438_A578G',
 '12438_G622A',
 '12438_C634T',
 '12438_C676T',
 '12438_G677A',
 '12438_C682T',
 '12438_G683A',
 '7540_A157T',
 '7540_T242C']
```

In [ ]:
```
### YOUR SOLUTION STARTS HERE ###

expected_frequency={}
dict(zip(mut_list,[0 for _ in range(len(mut_list))]))
for i in mut_list:
```

```
    prefix=i.split("_")[0]
    if prefix not in expected_frequency:
        expected_frequency[prefix]=0
        expected_frequency[prefix]+=1
    else:
        expected_frequency[prefix]+=1

expected_frequency = {key: 1/value for key, value in expected_frequency.items()}
expected_frequency
```

Out[ ]:
```
{'7696': 0.2,
 '8466': 1.0,
 '13668': 1.0,
 '5174': 0.5,
 '8593': 1.0,
 '8833': 0.2,
 '8522': 0.05,
 '9938': 0.25,
 '804': 0.5,
 '10062': 0.2,
 '1136': 0.25,
 '6936': 1.0,
 '12438': 0.14285714285714285,
 '7540': 0.5}
```

(i) Generate a mutation summary table

In [ ]:
```
mut_summary=pd.DataFrame(mut_list,columns=["attempt_mut"])
mut_summary["gene"]=mut_summary["attempt_mut"].apply(lambda x: x.split("_")[0])
mut_summary["pos"]=mut_summary["attempt_mut"].apply(lambda x: re.search(r'[A-Za-z]+(\d+)[A-Z
mut_summary["undesired_mut"]=""
# mut_summary["undesired_mut"]=mut_summary["attempt_mut"].apply(lambda x: x[:-1])
mut_summary["main"]=mut_summary["attempt_mut"].apply(lambda x: x.split("_")[1][0])
mut_summary["mut"]=mut_summary["attempt_mut"].apply(lambda x: x.split("_")[1][-1])

# df definition continues in cell below (separated due to order of header) :
```

In [ ]:
```
mut_summary["is_detected"]=False
mut_summary["is_clean"]=False
mut_summary["obs_freq"]=0.0
mut_summary["exp_freq"]=mut_summary["gene"].apply(lambda x: expected_frequency[x])

for i in range(len(mut_summary)):
    # first define these variables, or execution will be extremely slow:
    gene_ms=mut_summary.iloc[i]["gene"]
    pos_ms=int(mut_summary.iloc[i]["pos"])
    main=mut_summary.iloc[i]["main"]
    mut=mut_summary.iloc[i]["mut"]
    am_ms=mut_summary.iloc[i]["attempt_mut"]
    left=list(set(["A","G","C","T"])-set([main,mut])) #remaining nucleotide from attempted m
    genepart=df_summary[df_summary["gene"]==gene_ms]
    mut_summary.loc[i, "left"] = ",".join(left) # this line might be no longer useful, but l
    # print(bool((genepart[genepart["pos"]==pos_ms]["C"].values >0)))

    if pos_ms in genepart["pos"].values and \
        bool((genepart[genepart["pos"]==pos_ms][main].values >0)) and\
        bool((genepart[genepart["pos"]==pos_ms][mut].values >0)):
        mut_summary.loc[i, "is_detected"] = True
        for item in left:
            if bool((genepart[genepart["pos"]==pos_ms][item].values >0)):
                mut_summary.loc[i, "undesired_mut"] +=am_ms[:-1]+item+";"

    mut_summary.loc[i, "obs_freq"]=genepart[genepart["pos"]==pos_ms][mut].values/\
    (genepart[genepart["pos"]==pos_ms]["A"].values+genepart[genepart["pos"]==pos_ms]["T"].va
     +genepart[genepart["pos"]==pos_ms]["C"].values+genepart[genepart["pos"]==pos_ms]["G"].v
```

```
mut_summary["undesired_mut"]=mut_summary["undesired_mut"].apply(lambda x: x[:-1]) #remove th

mut_summary["is_clean"]=mut_summary["undesired_mut"]==""

mut_summary=mut_summary[['attempt_mut','gene','pos','undesired_mut','is_detected','is_clean'

mut_summary\
#.head(10)
```

Out[ ]:

	attempt_mut	gene	pos	undesired_mut	is_detected	is_clean	obs_freq	exp_freq
0	7696_G388A	7696	388		False	True	0.000000	0.200000
1	7696_G403A	7696	403	7696_G403C	True	False	0.211155	0.200000
2	7696_C406T	7696	406		True	True	0.137255	0.200000
3	7696_G415A	7696	415	7696_G415C;7696_G415T	True	False	0.100000	0.200000
4	7696_A451G	7696	451	7696_A451C;7696_A451T	True	False	0.096491	0.200000
5	8466_T148A	8466	148		False	True	1.000000	1.000000
6	13668_G328A	13668	328		False	True	1.000000	1.000000
7	5174_G1093A	5174	1093	5174_G1093T	True	False	0.112994	0.500000
8	5174_A1208T	5174	1208	5174_A1208G	True	False	0.533784	0.500000
9	8593_A1204T	8593	1204	8593_A1204G	True	False	0.310000	1.000000
10	8833_C380A	8833	380	8833_C380T	True	False	0.134791	0.200000
11	8833_G482A	8833	482	8833_G482C;8833_G482T	True	False	0.203200	0.200000
12	8833_G493A	8833	493	8833_G493C;8833_G493T	True	False	0.205298	0.200000
13	8833_G511C	8833	511	8833_G511T;8833_G511A	True	False	0.222222	0.200000
14	8833_C568T	8833	568	8833_C568G;8833_C568A	True	False	0.163755	0.200000
15	8522_G142C	8522	142	8522_G142T	True	False	0.049777	0.050000
16	8522_G163A	8522	163	8522_G163C;8522_G163T	True	False	0.056284	0.050000
17	8522_G178T	8522	178	8522_G178C;8522_G178A	True	False	0.041701	0.050000
18	8522_C274T	8522	274	8522_C274G	True	False	0.036885	0.050000
19	8522_G302A	8522	302	8522_G302C;8522_G302T	True	False	0.048534	0.050000
20	8522_G364A	8522	364	8522_G364T	True	False	0.054152	0.050000
21	8522_G454A	8522	454	8522_G454C	True	False	0.001803	0.050000
22	8522_A533G	8522	533	8522_A533C	True	False	0.053254	0.050000
23	8522_G579T	8522	579	8522_G579A	True	False	0.057842	0.050000
24	8522_A643G	8522	643	8522_A643C	True	False	0.044665	0.050000
25	8522_G763A	8522	763	8522_G763C;8522_G763T	True	False	0.040210	0.050000
26	8522_C764T	8522	764	8522_C764G;8522_C764A	True	False	0.024201	0.050000
27	8522_G835A	8522	835	8522_G835C;8522_G835T	True	False	0.039852	0.050000
28	8522_C851T	8522	851		False	True	0.000000	0.050000
29	8522_T860C	8522	860	8522_T860G;8522_T860A	True	False	0.059049	0.050000
30	8522_G914A	8522	914		True	True	0.049559	0.050000
31	8522_C926T	8522	926	8522_C926G;8522_C926A	True	False	0.112981	0.050000
32	8522_G934A	8522	934	8522_G934C;8522_G934T	True	False	0.054667	0.050000
33	8522_A959G	8522	959	8522_A959C;8522_A959T	True	False	0.058824	0.050000
34	8522_A967G	8522	967	8522_A967C	True	False	0.033399	0.050000
35	9938_G340A	9938	340	9938_G340T	True	False	0.197917	0.250000
36	9938_C347G	9938	347		False	True	0.000000	0.250000

	attempt_mut	gene	pos	undesired_mut	is_detected	is_clean	obs_freq	exp_freq
37	9938_G373A	9938	373	9938_G373C;9938_G373T	True	False	0.344828	0.250000
38	9938_C469T	9938	469	9938_C469G;9938_C469A	True	False	0.197917	0.250000
39	804_A251G	804	251	804_A251C	True	False	0.684615	0.500000
40	804_A367G	804	367	804_A367C;804_A367T	True	False	0.352941	0.500000
41	10062_C184T	10062	184		True	True	0.152778	0.200000
42	10062_G215A	10062	215		True	True	0.142857	0.200000
43	10062_A265C	10062	265		True	True	0.279412	0.200000
44	10062_C769G	10062	769		True	True	0.184211	0.200000
45	10062_T1058C	10062	1058		True	True	0.250000	0.200000
46	1136_C241T	1136	241	1136_C241G	True	False	0.300000	0.250000
47	1136_A400G	1136	400		True	True	0.293103	0.250000
48	1136_C409T	1136	409		True	True	0.266667	0.250000
49	1136_G433A	1136	433	1136_G433C;1136_G433T	True	False	0.250000	0.250000
50	6936_A470G	6936	470		False	True	0.000000	1.000000
51	12438_A578G	12438	578	12438_A578C;12438_A578T	True	False	0.202312	0.142857
52	12438_G622A	12438	622	12438_G622C;12438_G622T	True	False	0.355899	0.142857
53	12438_C634T	12438	634	12438_C634G;12438_C634A	True	False	0.122873	0.142857
54	12438_C676T	12438	676		False	True	0.000000	0.142857
55	12438_G677A	12438	677		True	True	0.093156	0.142857
56	12438_C682T	12438	682	12438_C682G;12438_C682A	True	False	0.057692	0.142857
57	12438_G683A	12438	683	12438_G683C	True	False	0.102119	0.142857
58	7540_A157T	7540	157	7540_A157G;7540_A157C	True	False	0.592593	0.500000
59	7540_T242C	7540	242		True	True	0.578947	0.500000

In [ ]:
```
# Test only:

# 4/15

# df_summary[(df_summary["gene"]=="7696") & (df_summary["pos"]==388)]

# 24/(24+2+1+213)

# mut_summary.head()
```

(j) Successfully generated mutations

In [ ]:
```
### YOUR SOLUTION STARTS HERE ###

success_mut=mut_summary[(mut_summary["is_clean"]==True)& (mut_summary["obs_freq"]>=mut_summa
success_mut
```

Out[ ]:

	attempt_mut	gene	pos	undesired_mut	is_detected	is_clean	obs_freq	exp_freq
5	8466_T148A	8466	148		False	True	1.000000	1.00
6	13668_G328A	13668	328		False	True	1.000000	1.00
43	10062_A265C	10062	265		True	True	0.279412	0.20
45	10062_T1058C	10062	1058		True	True	0.250000	0.20
47	1136_A400G	1136	400		True	True	0.293103	0.25
48	1136_C409T	1136	409		True	True	0.266667	0.25
59	7540_T242C	7540	242		True	True	0.578947	0.50

Part 2: Machine Learning

Please load all the modules that you used below (feel free to load any modules you want to use)

In [ ]:
```
import torch
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.weightstats import CompareMeans
import pickle
#############

import warnings
warnings.filterwarnings('ignore') # Turn off the display of warnings
```

Please use the space below to do Part 2

- Feel free to add cell or markdown below
- Please follow the instructions (`Final Project.pdf`)

Data loading and skimming:

In [ ]:
```
expasy=pd.read_csv("data/Expasy_AA_Scales.txt",delimiter="\t")
expasy.head()
```

Out[ ]:

	AA	Coil_Deleage_Roux	beta_sheet_Deleage_Roux	HPLC_retention_pH74	beta_sheet_Levitt	Hphob_Wolfen
0	A	0.824	0.709	0.5	0.90	1
1	C	0.953	1.191	-6.8	0.74	-1
2	E	0.761	0.567	-16.9	0.75	-1(
3	D	1.197	0.541	-8.2	0.72	-1(
4	G	1.251	0.657	0.0	0.92	2

5 rows × 58 columns

In [ ]:
```
up2s=pd.read_csv("data/UniProt2Seq.txt",delimiter="\t")
up2s.head()
```

Out[ ]:

	UniProt	Position	AA
0	A0R4Q6	1	M
1	A0R4Q6	2	T
2	A0R4Q6	3	Q
3	A0R4Q6	4	M
4	A0R4Q6	5	L

In [ ]:
```
samples=pd.read_csv("data/samples.txt",delimiter="\t")
samples.head()
```

Out[ ]:

	UniProt	Position	is_alpha_helix
0	A6X980	1	0
1	A6X980	2	0
2	A6X980	3	0
3	A6X980	4	0
4	A6X980	5	0

Data construction and cleaning:

We now construct the dataset we want to use for our model. First check the structure of the dataset: The code below is for checking if all `UniProt` in `samples` are in `up2s`, if no, that UniProt will be output and loop will be terminated:

In [ ]:
```
for i in samples["UniProt"].unique():
    if i not in up2s["UniProt"].unique():
        print(i)
        break

# Yes
```

Since yes, we can join the information in our 3 datasets to make it into a large dataset that can be used conveniently:

Detailed steps are below:

1. Join `samples` and `up2s` to get the type of amino-acid at the position in the protein.
2. On top of that, combine with `expasy` to add all features to the corresponding type of amino-acid at the position in the protein (column `AA`)

In [ ]:
```
merged_df = samples.merge(up2s, on=['UniProt', 'Position'])

final_df = merged_df.merge(expasy, on='AA').sort_values(["UniProt","Position"])

# final_df=final_df.drop(["UniProt","Position"], axis=1)
# ↑ After careful consideration, we cant drop these 2 columns as they are the only columns c
final_df
```

Out[ ]:

	UniProt	Position	is_alpha_helix	AA	Coil_Deleage_Roux	beta_sheet_Deleage_Roux	HPLC_retention_pl
0	A6X980	1	0	M	0.810	1.210	
1242	A6X980	2	0	S	1.130	0.928	
1243	A6X980	3	0	S	1.130	0.928	
5563	A6X980	4	0	N	1.167	0.604	
1244	A6X980	5	0	S	1.130	0.928	
...	...	...	...	...	...	...	
7880	T0D7A2	1125	0	N	1.167	0.604	
23505	T0D7A2	1126	0	T	1.148	1.221	
41640	T0D7A2	1127	0	G	1.251	0.657	
26648	T0D7A2	1128	0	D	1.197	0.541	
33063	T0D7A2	1129	0	I	0.886	1.799	

57367 rows × 61 columns

Train/Test Split

I use Train=60% to split the model.

In [ ]:
```
X_train, X_test, y_train, y_test=train_test_split(final_df.drop("is_alpha_helix",axis=1),fir
```

Feature Engineering

Aggregate info.

We first focus on the column "beta_turn_Levitt" and construct a 2d-window average value surrounding each residue in a certain type of protein ("Position"-d , "Position"+d) as following:

In [ ]:
```
d=2
final_df['beta_turn_Levitt_Aggr'] = final_df.groupby('UniProt')['beta_turn_Levitt'].rolling(
final_df
```

Out[ ]:

	UniProt	Position	is_alpha_helix	AA	Coil_Deleage_Roux	beta_sheet_Deleage_Roux	HPLC_retention_pl
0	A6X980	1	0	M	0.810	1.210	
1242	A6X980	2	0	S	1.130	0.928	
1243	A6X980	3	0	S	1.130	0.928	
5563	A6X980	4	0	N	1.167	0.604	
1244	A6X980	5	0	S	1.130	0.928	
...	...	...	...	...	...	...	...
7880	T0D7A2	1125	0	N	1.167	0.604	
23505	T0D7A2	1126	0	T	1.148	1.221	
41640	T0D7A2	1127	0	G	1.251	0.657	
26648	T0D7A2	1128	0	D	1.197	0.541	
33063	T0D7A2	1129	0	I	0.886	1.799	

57367 rows × 63 columns

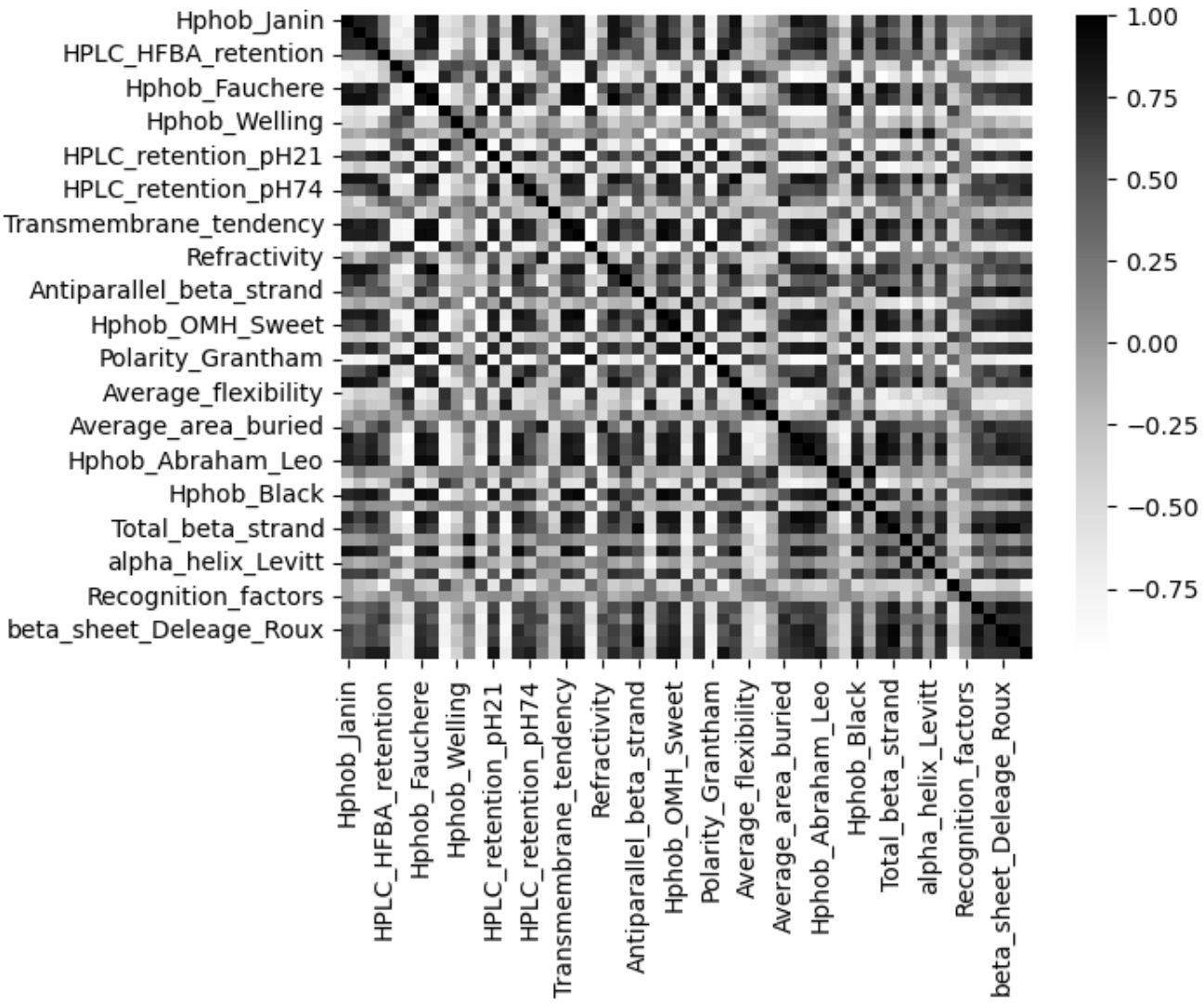
as shown above, this method also brings some problem. For example, the first and last two (since k=2) residues will have no values due to the size of the window and for every feature, not only we should construct aggregate information for all of them , but we also need to make comparison among them, which is not intuitive and might take enumerous of time. So we choose other methods in below to do feature engineering (**order here might be a little messy but feature rankings is after Data Exploratory Analysis**):

Data Exploratory Analysis

First we look at the correlations between each 2 pairs of features by heatmap correlation, where the darkest (positive) and brightest (negative) blocks corresponds to the 2 variables that have strong correlations. We can see few obvious dark blocks in the plot, meaning multiple features are all correlated with each others.

In [ ]:
```
sb.heatmap(expasy[set(expasy.columns)-set(['AA'])].corr(),cmap='Greys')
```

Out[ ]:
```
<AxesSubplot:>
```



This plot above however only gives us an initial overview on what variable(s) might be correlated. However it is still not intuitive for selection. Next we sort the values and display them along with their related variables based on absolute correlations, by excluding values of 1 (which are the correlation to themselves):

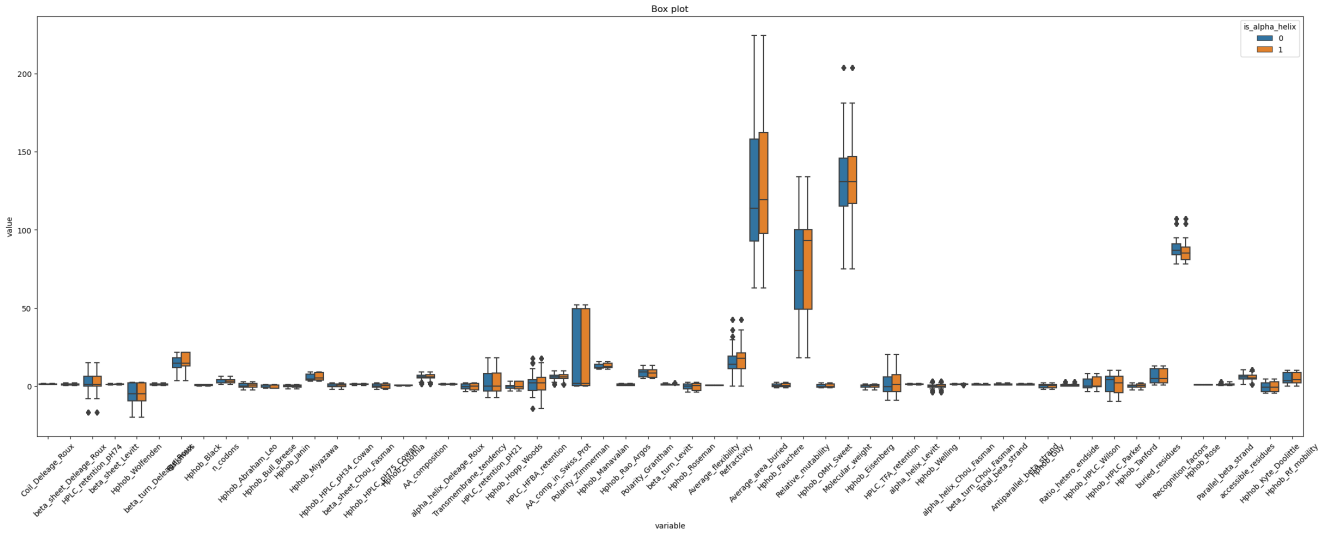
In [ ]:
```
corr=expasy[set(expasy.columns)-set(['AA'])].corr()
corr=corr[corr!=1].stack().abs().sort_values(ascending=False)
corr
```

Out[ ]:
```
AA_composition          AA_comp_in_Swiss_Prot      0.989503
AA_comp_in_Swiss_Prot   AA_composition             0.989503
Hphob_Tanford           Hphob_Eisenberg            0.971888
Hphob_Eisenberg         Hphob_Tanford              0.971888
Hphob_Kyte_Doolittle    Hphob_Chothia              0.963024
                                                      ...
n_codons                beta_sheet_Deleage_Roux    0.001808
HPLC_HFBA_retention     AA_composition             0.001581
AA_composition          HPLC_HFBA_retention        0.001581
                        Hphob_Eisenberg            0.000989
Hphob_Eisenberg         AA_composition             0.000989
Length: 3192, dtype: float64
```

We can also use a box plot to see if certain feature has significantly different values under the categorization of 0 and 1, which proves that this feature may be a good predictor for identifying whether it is an alpha helix (for values 0 and 1 in `is_alpha_helix`) or not.

In [ ]:
```
melted_df = final_df.melt(id_vars='is_alpha_helix', value_vars=final_df.columns[4:])
plt.figure(figsize=(30,10))
sb.boxplot(x='variable', y='value', hue='is_alpha_helix', data=melted_df)
```

```
plt.title('Box plot')
plt.xticks(rotation=45)
plt.show()
```



Once we find out the possible target, we can create a separated plot to kook at it more closely. For exmaple, `Hphob_Fauchere`:

Since the range varies too much in the plot and hence yields a graph that is not too good for us to observe. We do a hypothesis testing (wald test) on the meaning of each feature under condition of `is_alpha_helix`. The hypotheses are:

$$H_0 : \mu_1 = \mu_0$$
$$H_A : \mu_1 \neq \mu_0$$

Afte the test we filter out and make out a dataframe of all features with corresponding p-values less than 5% (which means we reject H_0 that two means are the same, indicating its value differs a lot between `is_alpha_helix` =0 and `is_alpha_helix` =1)

In [ ]:
```
htres={}
for i in final_df.columns[4:]:
    category_0 = final_df[final_df['is_alpha_helix'] == 0][i]
    category_1 = final_df[final_df['is_alpha_helix'] == 1][i]
    cm = CompareMeans.from_data(category_0, category_1)
    zstat, pval = cm.ztest_ind(alternative='two-sided')
    htres[i]={"Z-stats.":zstat,"p-value":pval}

htres=pd.DataFrame(htres).T
htres_sig=htres["p-value"]<0.05
sigftr=list(htres_sig.index) #output the significant features as a list called sigftr

htres
```

Out[ ]:

	Z-stats.	p-value
Coil_Deleage_Roux	32.402492	2.531583e-230
beta_sheet_Deleage_Roux	-11.505478	1.238010e-30
HPLC_retention_pH74	-1.104977	2.691693e-01
beta_sheet_Levitt	-8.116007	4.817711e-16
Hphob_Wolfenden	1.169509	2.421988e-01
beta_turn_Deleage_Roux	25.031497	2.776824e-138
Bulkiness	-18.591722	3.749452e-77
Hphob_Black	-6.524755	6.811276e-11
n_codons	5.710883	1.123918e-08
Hphob_Abraham_Leo	-10.147018	3.416426e-24
Hphob_Bull_Breese	14.239642	5.199256e-46
Hphob_Janin	-2.144356	3.200441e-02
Hphob_Miyazawa	-16.200366	5.012746e-59
Hphob_HPLC_pH34_Cowan	-9.471270	2.764629e-21
beta_sheet_Chou_Fasman	-11.671323	1.786227e-31
Hphob_HPLC_pH75_Cowan	-5.539123	3.039905e-08
Hphob_Chothia	-7.747449	9.375710e-15
AA_composition	-4.922852	8.529189e-07
alpha_helix_Deleage_Roux	-34.249250	4.474987e-257
Transmembrane_tendency	-8.531881	1.439877e-17
HPLC_retention_pH21	-7.300132	2.874854e-13
Hphob_Hopp_Woods	4.528424	5.942523e-06
HPLC_HFBA_retention	-15.748087	7.079085e-56
AA_comp_in_Swiss_Prot	-8.867982	7.448354e-19
Polarity_Zimmerman	-5.622725	1.879690e-08
Hphob_Manavalan	-16.942642	2.181035e-64
Hphob_Rao_Argos	-6.381675	1.751619e-10
Polarity_Grantham	7.720719	1.156756e-14
beta_turn_Levitt	29.835237	1.364528e-195
Hphob_Roseman	-4.575984	4.739871e-06
Average_flexibility	20.750225	1.220202e-95
Refractivity	-17.046322	3.722309e-65
Average_area_buried	-21.939647	1.087425e-106
Hphob_Fauchere	-8.657522	4.821305e-18
Relative_mutability	1.993617	4.619394e-02
Hphob_OMH_Sweet	-13.418331	4.722283e-41
Molecular_weight	-16.943192	2.160769e-64

	Z-stats.	p-value
Hphob_Eisenberg	-4.692245	2.702236e-06
HPLC_TFA_retention	-16.723874	8.782029e-63
alpha_helix_Levitt	-31.655421	6.386463e-220
Hphob_Welling	-8.065764	7.277933e-16
alpha_helix_Chou_Fasman	-33.059065	1.152506e-239
beta_turn_Chou_Fasman	30.346944	2.756911e-202
Total_beta_strand	-16.624674	4.618747e-62
Antiparallel_beta_strand	-17.024511	5.404216e-65
Hphob_Guy	10.326102	5.369857e-25
Ratio_hetero_endside	18.263229	1.623964e-74
Hphob_HPLC_Wilson	-10.179035	2.459876e-24
Hphob_HPLC_Parker	16.692546	1.485002e-62
Hphob_Tanford	-5.920222	3.215080e-09
buried_residues	-1.271316	2.036164e-01
Recognition_factors	19.007134	1.488672e-80
Hphob_Rose	-11.093469	1.349504e-28
Parallel_beta_strand	-12.255478	1.570215e-34
accessible_residues	11.150329	7.134191e-29
Hphob_Kyte_Doolittle	-9.715077	2.600520e-22
Hphob_Rf_mobility	-13.371708	8.848502e-41

As shown above, it only filters out a little proportion of the dataset. Although we can do sorting based on the p-value, it's not meaningful because for most of them the p-values are much less than 5%. We still need to perform other kinds of feature engingeering:

We make another table `corr_df` to show the correlation of each feature with respect to variable `is_alpha_helix`:

In [ ]:
```
corrdict={}
corrl=[]
for i in final_df[sigftr]:
    corrl.append(final_df["is_alpha_helix"].corr(final_df[i]))

corr_df=pd.DataFrame(dict(zip(final_df[sigftr], corrl)),index=["correlation"]).T
corr_df=corr_df.reindex(corr_df['correlation'].abs().sort_values(ascending=False).index)
corr_df
```

Out[ ]:

	correlation
alpha_helix_Deleage_Roux	0.141557
alpha_helix_Chou_Fasman	0.136732
Coil_Deleage_Roux	-0.134065
alpha_helix_Levitt	0.131028
beta_turn_Chou_Fasman	-0.125699
beta_turn_Levitt	-0.123612
beta_turn_Deleage_Roux	-0.103945
Average_area_buried	0.091220
Average_flexibility	-0.086313
Recognition_factors	-0.079110
Bulkiness	0.077391
Ratio_hetero_endside	-0.076032
Refractivity	0.070992
Antiparallel_beta_strand	0.070902
Molecular_weight	0.070565
Hphob_Manavalan	0.070562
HPLC_TFA_retention	0.069656
Hphob_HPLC_Parker	-0.069526
Total_beta_strand	0.069245
Hphob_Miyazawa	0.067485
HPLC_HFBA_retention	0.065610
Hphob_Bull_Breese	-0.059348
Hphob_OMH_Sweet	0.055936
Hphob_Rf_mobility	0.055743
Parallel_beta_strand	0.051102
beta_sheet_Chou_Fasman	0.048672
beta_sheet_Deleage_Roux	0.047982
accessible_residues	-0.046504
Hphob_Rose	0.046268
Hphob_Guy	-0.043073
Hphob_HPLC_Wilson	0.042461
Hphob_Abraham_Leo	0.042328
Hphob_Kyte_Doolittle	0.040529
Hphob_HPLC_pH34_Cowan	0.039513
AA_comp_in_Swiss_Prot	0.037000
Hphob_Fauchere	0.036123
Transmembrane_tendency	0.035600

	correlation
beta_sheet_Levitt	0.033866
Hphob_Welling	0.033657
Hphob_Chothia	0.032330
Polarity_Grantham	-0.032219
HPLC_retention_pH21	0.030465
Hphob_Black	0.027232
Hphob_Rao_Argos	0.026635
Hphob_Tanford	0.024711
n_codons	-0.023837
Polarity_Zimmerman	0.023469
Hphob_HPLC_pH75_Cowan	0.023121
AA_composition	0.020550
Hphob_Eisenberg	0.019587
Hphob_Roseman	0.019102
Hphob_Hopp_Woods	-0.018904
Hphob_Janin	0.008953
Relative_mutability	-0.008323
buried_residues	0.005308
Hphob_Wolfenden	-0.004883
HPLC_retention_pH74	0.004613

Therefore, by the steps above, we now know the top few features with the highest correlations, not only to the response variable, but also to other features. For the later one, it implied us to combine them as one or simply choose only one of them. We can however do this simplier and better.

The steps we did above is manual calculation. Plus, we also don't know how many of the top features should we choose. We can then improve and check our answer using various different packages in python to help us do the decision. Our idea is to first determine the best number of features and can cover most original characteristics of the original data and then find those features out. Below are some machine learning processes can be for this purpose:

1. PCA to get the optimal dimension for certain proportion of features in data wanted.
2. `SelectKBest()` from `sklearn` package to determine the features to keep.

In [ ]:
```
# PCA:
expasy_std = StandardScaler().fit_transform(final_df.drop(["UniProt","AA","is_alpha_helix"],
pca=PCA(n_components = 0.9) # number of component to explain 90% of the characteristics of d
expasy_pca = pca.fit_transform(expasy_std)
pca.n_components_
```

Out[ ]:
```
8
```

Then once we get the number of features that can cover 90% detail of the orignal dataset (6), we can use `SelectKBest()` in `sklearn` package to select our top features. Remember we reserved 2 rows

for "UniProt" and "Position", so besides that we still need 6 more features to include:

In [ ]:
```
# Substitute k=6, which is the n components we get from last question:
selector = SelectKBest(score_func=f_regression, k=pca.n_components_-2) # 2 for reserved `Uni

# we must contain "position" to determine AA, so we drop it here, later we will add it to ou
X_new = selector.fit_transform(X_train[sigftr], y_train)

# Get the indices of the selected features
selected_features = list(X_train[sigftr].columns[selector.get_support()])
selected_features
```

Out[ ]:
```
['Coil_Deleage_Roux',
 'alpha_helix_Deleage_Roux',
 'beta_turn_Levitt',
 'alpha_helix_Levitt',
 'alpha_helix_Chou_Fasman',
 'beta_turn_Chou_Fasman']
```

Therefore, our top 6 features are:

- Coil_Deleage_Roux
- alpha_helix_Deleage_Roux
- beta_turn_Levitt
- alpha_helix_Levitt
- alpha_helix_Chou_Fasman
- beta_turn_Chou_Fasman

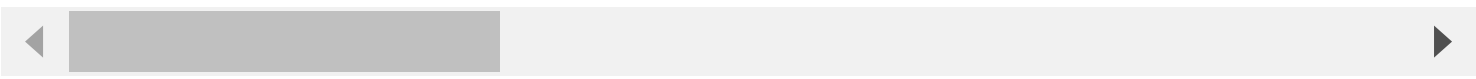
Upon this, `Position` and `UniProt` will also be included. As for UniProt we use a special way to record it, since sklearn cant handle string as label, we make an additional column in `final_df` of a series of labels (numbers, start some 0) called `UniProtIdx` to represent `UniProt`.

In [ ]:
```
final_df['UniProtIdx'] = pd.factorize(final_df['UniProt'])[0]
final_df
```

Out[ ]:

	UniProt	Position	is_alpha_helix	AA	Coil_Deleage_Roux	beta_sheet_Deleage_Roux	HPLC_retention_pl
0	A6X980	1	0	M	0.810	1.210	
1242	A6X980	2	0	S	1.130	0.928	
1243	A6X980	3	0	S	1.130	0.928	
5563	A6X980	4	0	N	1.167	0.604	
1244	A6X980	5	0	S	1.130	0.928	
...	...	...	...	...	...	...	
7880	T0D7A2	1125	0	N	1.167	0.604	
23505	T0D7A2	1126	0	T	1.148	1.221	
41640	T0D7A2	1127	0	G	1.251	0.657	
26648	T0D7A2	1128	0	D	1.197	0.541	
33063	T0D7A2	1129	0	I	0.886	1.799	

57367 rows × 62 columns



In [ ]:
```
# Check if factorization was success:
```

```
len(final_df.UniProt.unique())==len(final_df.UniProtIdx.unique())
# Yes
```

Out[ ]:
```
True
```

Update the dataset:

In [ ]:
```
all_features=selected_features
all_features.extend(["Position","UniProtIdx"])
X_train, X_test, y_train, y_test=\
    train_test_split(final_df[all_features],final_df["is_alpha_helix"],test_size=0.4)
```

In [ ]:
```
X_train.head()
```

Out[ ]:

	Coil_Deleage_Roux	alpha_helix_Deleage_Roux	beta_turn_Levitt	alpha_helix_Levitt	alpha_helix_Chou_Fa
14025	0.810	1.236	0.58	1.30	
45211	0.893	1.224	0.88	0.96	
32538	0.886	1.003	0.51	0.97	
19339	0.761	1.504	0.99	1.44	
13064	0.810	1.236	0.58	1.30	

Model selection

Since outcome is binary, possible models for predict such a values are:

- Decision Tree
- Logistic regression
- KNN/KMeans

First we build a decision tree classifier with default hyperparam. (those will be fine-tuned later if this, decision tree is our selected model):

In [ ]:
```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
print(dtc.score(X_train, y_train),"\n",cross_val_score(dtc, X_train, y_train, cv=5).mean(),s
```

```
1.0
0.828965717606043
```

We get 100% accuracy for traning dataset because for a decision tree, we can always get 100% accuracy by making modes as many as enough to cover all outcomes as singletons. But this might cause overfitting. We will assess this in our future analysis. For the k-fold validation score, we got 82.93%.

Next we move to another model, logistic model, which has the following form:

$$\ln\left(\frac{\mathbb{P}(\texttt{is\_alpha\_helix} = 1)}{1 - \mathbb{P}(\texttt{is\_alpha\_helix} = 1)}\right) = \hat{\beta}$$

, where

$\hat{\beta} \in \mathbb{R}^{1\times n} = [\beta_0, \beta_1, \cdots, \beta_n]$,

$\hat{X} \in \mathbb{R}^{n\times 1} = [1, x_1, \cdots, x_n]^T$,

x's are selected features

In [ ]:
```
# Logit model:
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
print(logreg.score(X_train, y_train),"\n",cross_val_score(logreg, X_train, y_train, cv=5).me
```

```
0.7675188843695526
0.7675188843695526
```

We got 2 similar scores for logit model, but they are both still lower than those of the decision tree classifier.

For our last model candidate, we choose K nearest neighbours:

In [ ]:
```
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
print(knn_model.score(X_train, y_train),"\n",cross_val_score(knn_model, X_train, y_train, cv
```

```
0.887013364323068
0.8003195816385823
```

Surprisingly, as shown above KNN has both pretty good scores, at least much better than the logit model. Note here for all 3 models above I controlled and number of folds used for k-fold validation and also didnt specify any hyperparameter rather than letting computer decide for us. We will then compare those score pairs to get our final model.

Based on the results scores given from the 3 model above, decision tree classifier seems to be a competitve candidate for us to do fine-tuning and improvement. For the fine-tuning below, we used default `cv=` of 5. There are several hyperparams. we usually fine-tune `max_depth`

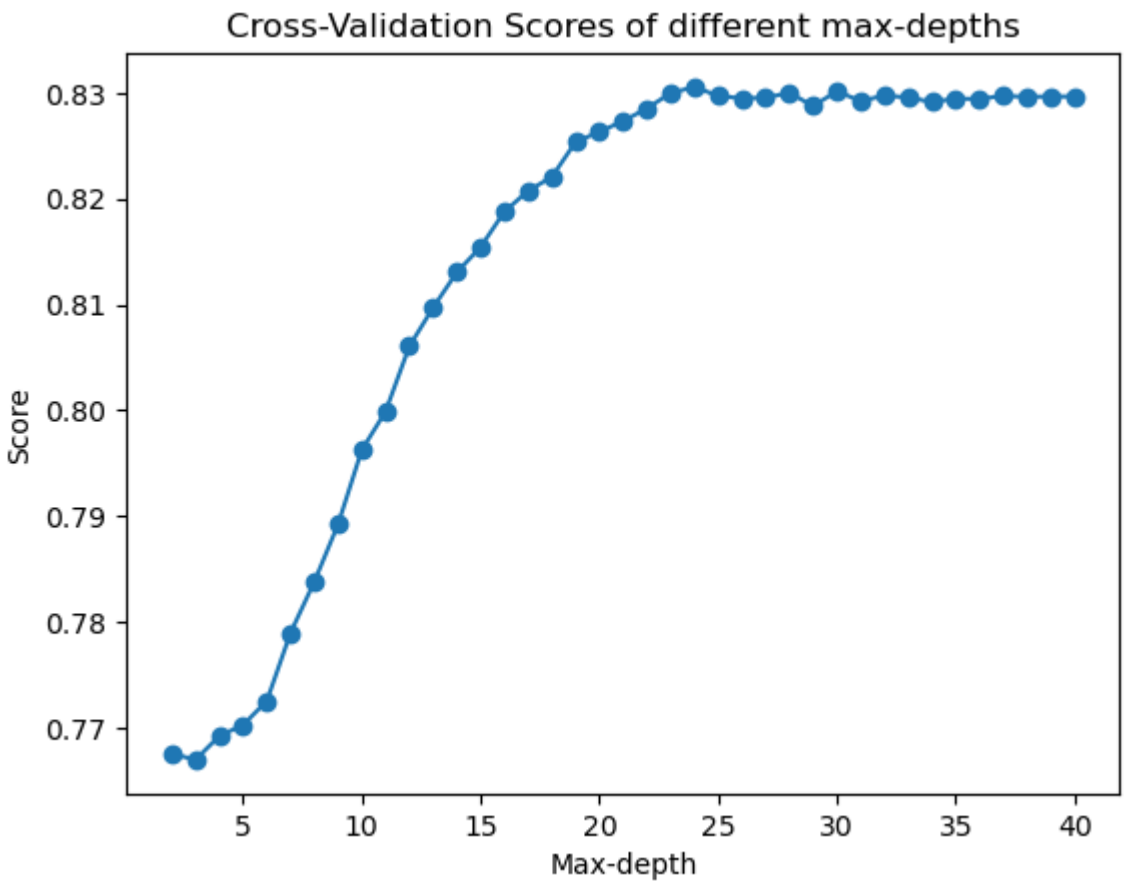
Hyperparameter Tuning

`max_depth`

For `max_depth`, we will use a line plot to visualize the quality of each candidate in range 2-41:

In [ ]:
```
mdres={}
for i in range(2,41):
    clf_t = DecisionTreeClassifier(random_state=6381,max_depth=i)
    clf_t.fit(X_train, y_train)
    mdres[i]=cross_val_score(clf_t, X_train, y_train, cv=5).mean()

plt.plot(mdres.keys(), mdres.values(), marker='o', linestyle='-')
plt.title('Cross-Validation Scores of different max-depths')
plt.xlabel('Max-depth')
plt.ylabel('Score')
plt.show()
```



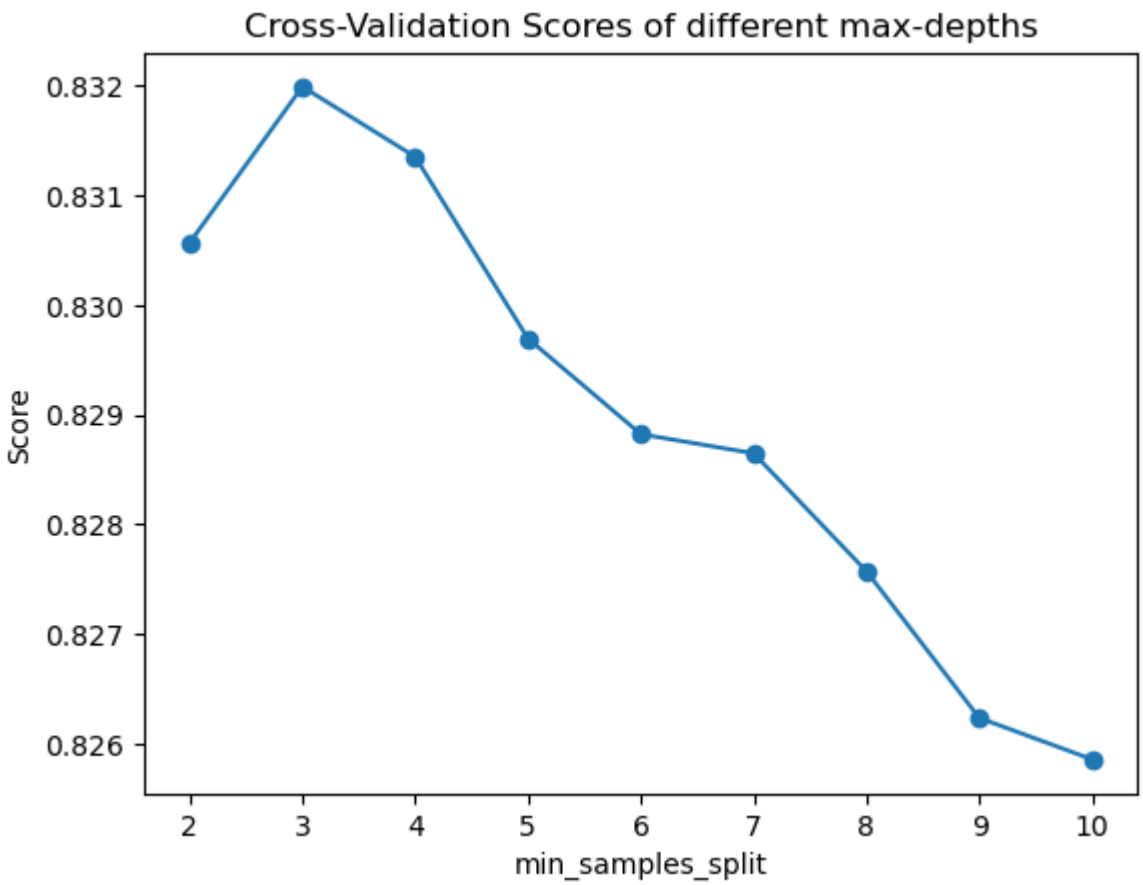
As shown above,when max-depth=24, model yields highest 5-fold validation score. (You can also check by see the value in `mdres`) So our model is a decision tree model with a max depth of 24.

`min_samples_split`

This is the minimum number of samples required to split an internal node. Increasing this parameter can lead to a simpler model with less splitting, which may help prevent overfitting. I will tune this hyperparameter based on the similar method as the previous one, but in the range 2~11:

In [ ]:
```
mssres={}
for i in range(2,11):
    clf_s = DecisionTreeClassifier(random_state=6381,max_depth=24,min_samples_split=i) #plug
    clf_s.fit(X_train, y_train)
    mssres[i]=cross_val_score(clf_s, X_train, y_train, cv=5).mean()

plt.plot(mssres.keys(), mssres.values(), marker='o', linestyle='-')
plt.title('Cross-Validation Scores of different max-depths')
plt.xlabel('min_samples_split')
plt.ylabel('Score')
plt.show()
```



As shown above, it's quite obvious to see the pattern of dropping for min_samples_split>3. So the optimal value of `min_samples_split` should be 3. I will not demonstrate more hyperparam. tunings here due to the length concern, but they should be actually all the same as above.

Results & Evaluation

Once we have the optimal hyperparams., we can get our final model to see results & do evaluations:

Retrain:

In [ ]:
```
dtf = DecisionTreeClassifier(max_depth=24,min_samples_split=3) #plug in the optimal hyperpar
dtf.fit(X_train, y_train)
```

Out[ ]:
```
DecisionTreeClassifier(max_depth=24, min_samples_split=3)
```

Assessment (ROC curve and AUC)

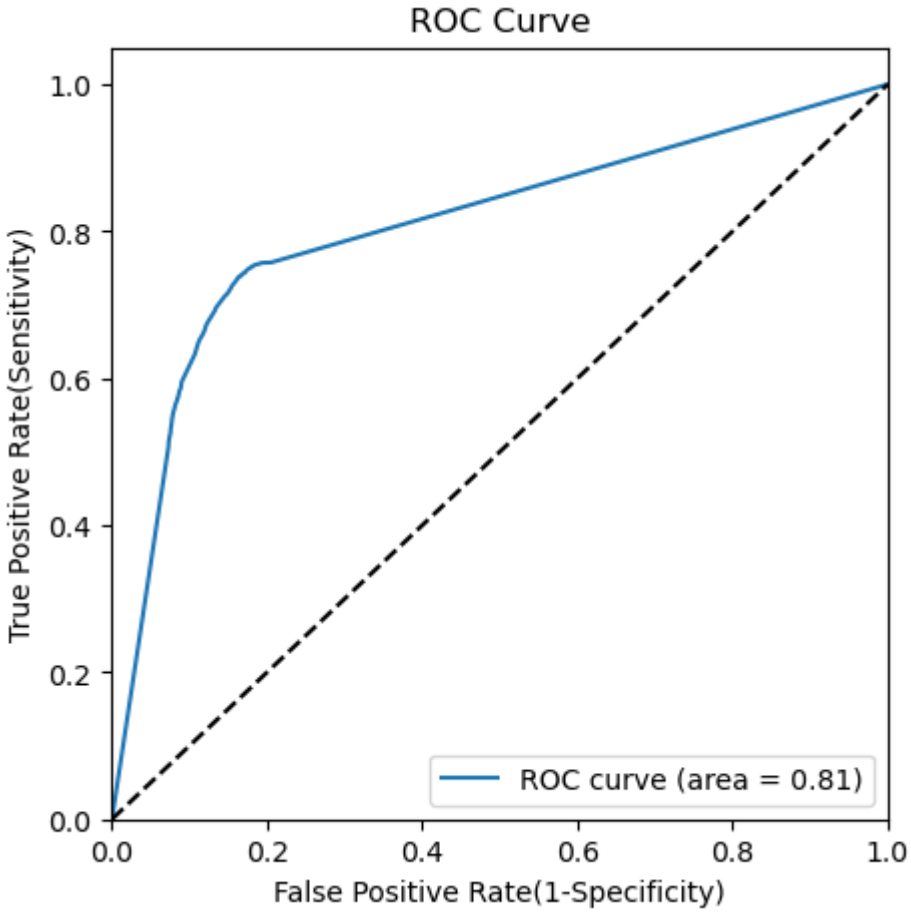
In [ ]:
```
y_pred = dtf.predict(X_test)
y_prob = dtf.predict_proba(X_test)[:, 1]

# plotting roc curve:
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Calculate the AUC score
auc = roc_auc_score(y_test, y_prob)

# Plot the ROC curve
plt.figure(figsize=(5,5))
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], 'k--')  # Random guess line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate(1-Specificity)')
plt.ylabel('True Positive Rate(Sensitivity)')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



As shown above, we got a decent ROC curve with a quite good AUC value of 0.81 (`area=` in the legend in the plot), which means that if one randomly selects a positive instance and a negative instance, the model's probability of assigning a higher score to the positive instance is 0.81.

But it's worth noticing that indicates that for the ending part of my ROC curve I got a straight upward line instead of a curve. This might be telling us the model is making perfect predictions in that region, which is likely a sign of overfitting or a perfect separation of the classes. This might be a future improvement direction.

Model Persistence

Save final model as a `pickle` file:

In [ ]:
```
with open('final_model.pkl', 'wb') as file:
    pickle.dump(dtf, file)
```