Gene Sequencing and Analysis

SPRING, 2020

CO307

SOFTWARE ENGINEERING

Lab Report

PREPARED BY:
(GROUP 7)

Adil Bin Bhutto	CSB17016
Deepankar Acharyya	CSB17017
Parikshit Saikia	CSB17035

Submitted to:

DR. SIDDHARTHA S. SATAPATHY

ACKNOWLEDGEMENT

First of all, we would like to thank our course instructor (Software Engineering), Dr. Siddhartha S. Satapathy, for providing us with such a wonderful opportunity to explore the interdisciplinary field. His guidance throughout the whole project has thoroughly helped us in successfully completing the project. Moreover, his lab assistants and our classmates also played a major role in clearing our doubts and helping us with their technical know-how.

We thank all the people for their help (directly or indirectly) with this lab assignment.

-Adil Bin Bhutto (CSB17016)
-Deepankar Acharyya (CSB17017)
-Parikshit Saikia (CSB17035)

Contents

1. Problem Statement:

As per the specified requirements mentioned in the original pdf regarding the **Gene Sequencing and Analysis**, we are tasked with creating a user interface that enables the user to upload codon sequences as the input file and to be able to perform the following tasks:

- Converting the codon sequences into amino acid sequences.
- Writing the amino acid sequences to a file.
- Writing information related to the codon sequence and corresponding amino acid sequence to a database.
- Plotting the percentage of each amino acid in the output amino acid sequence.
- Plotting AT/GC Skew along with their Maxima and Minima points and printing their intervals of positive and negative values. Link to other technical details shared with us here.

2. General Introduction:

In living organisms, genetic information is stored as DNA sequences that are passed on from one generation to another. The sequence consists of long strings of nucleotides (A, T, G, C). These strings are used to produce proteins which are a sequence of 20 amino acids and this information is very crucial for evaluating the nature and features of the biological component of an individual or even a species.

Given the enormous length of these strings and the added complexities, a high computation machine is able to analyze a given string of nucleotides and provide a base for an inference. In this report, we try to outline our approach to develop a stand-alone GUI based system that can perform a limited amount of analysis by devising appropriate logic and algorithms for each sub-problem associated with the different steps of the assignment-problem. We also implement visualization features so as to enhance the user's understanding and readability of various results from the opacity of the information provided by the mathematical data results and inferences.

3. Our Main Focus:

While developing through this project, we emphasized on the following points:

- Implementing Agile Development Techniques via the use of JIRA software.
- Sticking to our time-time.
- Creating a full-fledged stand-alone GUI system
- Efficiently implementing each and every task of the assignment, while maintaining the elegance of the system.

4. Our Tech-Stack:

- The full project/assignment is implemented using **Python 3** (including the GUI part).
- For the Database, we have used **PostgreSQL**.
- For version control, we have used **BitBucket (git)** .
- For Project Management, we have used **Atlassian JIRA software.**

5. The Implementation of the Assignment:

- ### **Module 1**: Database Module:

  This module mainly focuses only 2 points:
  - Designing an efficient schema that will serve the purpose of this assignment
  - Implementing the database (creating the tables)

  Module's pseudo-code and explanation :
  After thoroughly analyzing the requirements of the assignment, we decided to have a total of 2 tables. The schema for the tables are as follows:

➢ The schema for Table - 01:

- Name of the table: gene_table
- Attributes :
    - Sl.No. INT PRIMARY KEY
    - gene_name TEXT
    - gene_code
    - gene_num
    - nucleotide_sequence TEXT
    - acid_sequence TEXT
    - length_acid INT
    - a INT
    - g INT
    - t INT
    - c INT
    - ILE INT
    - LEU INT
    - VAL INT
    - PHE INT
    - MET INT
    - CYS INT
    - ALA INT
    - GLY INT
    - PRO INT
    - THR INT
    - SER INT
    - TYR INT
    - TYR INT
    - TRP INT
    - GLN INT
    - ASN INT
    - HIS INT
    - GLU INT
    - ASP INT
    - LYS INT
    - ARG INT
    - stop INT
    - remark TEXT

➢ The schema for Table - 02:

- Name of the table: Total_table
- Attributes :
    - Sl_No INT PRIMARY KEY
    - Element TEXT
    - Type Text
    - Count INT

Code for creating the following tables:

```
def create_tables():
   conn = psycopg2.connect(database="proteinSynthesis", user = username,
password = passwd, host = "localhost", port = "5432")
   #print("Opened database successfully")
   cur=conn.cursor()

   cur.execute("DROP TABLE IF EXISTS  GENE_TABLE")
   cur.execute(table1_create_command)
   #print("Table_01 created successfully")


   cur.execute("DROP TABLE IF EXISTS  TOTAL_TABLE")
   cur.execute(table2_create_command)
   #print("Table_02 created successfully")

   conn.commit()
   conn.close()
   #print("\n Connection Closed!\n")
   return
```

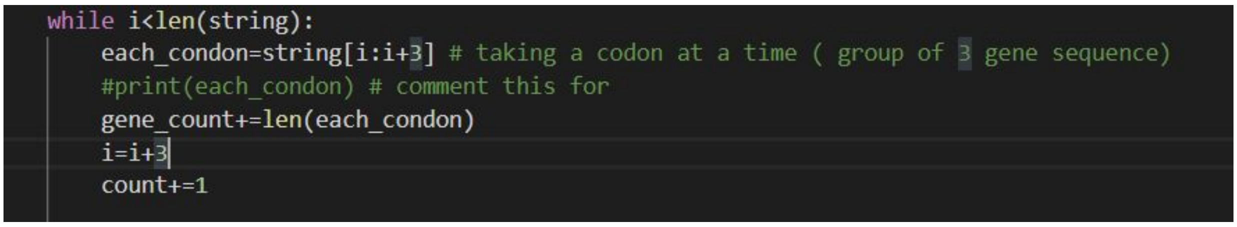
- ## **Module 2:** Parsing and Validation Module

  The main focus of this module are as follows:
  - Reading in the input file
  - Segmenting the input into different parts
  - Checking the validity of the parts
  - Entering the data to the database

  Module Pseudo-code and explanation :

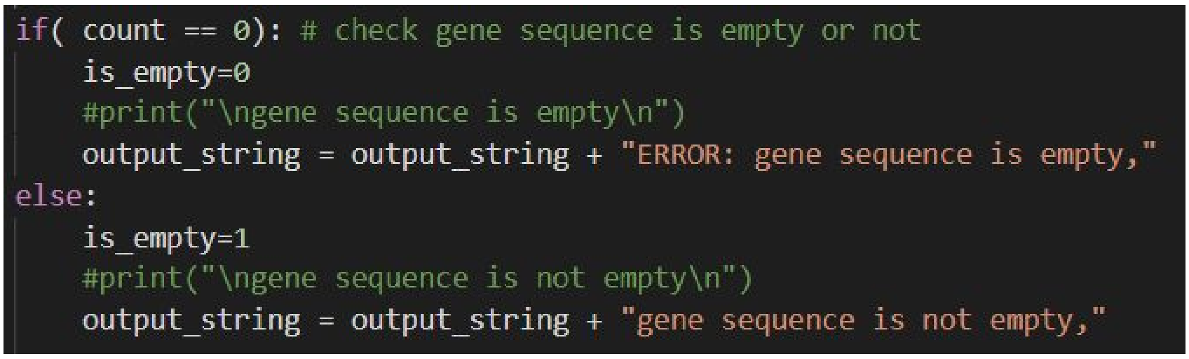
  - Validation module:
    - First, we will take the entire gene sequence for a particular gene as a string, and check whether it is empty or not.
    - Then we group them into codons and check if any nucleotides are left.

- For each codon, we also for each nucleotide consist of A, T, G, C, or not.
- We also check for the stop sequence for the ending codon, we also make sure that stop sequence only appears in the middle.



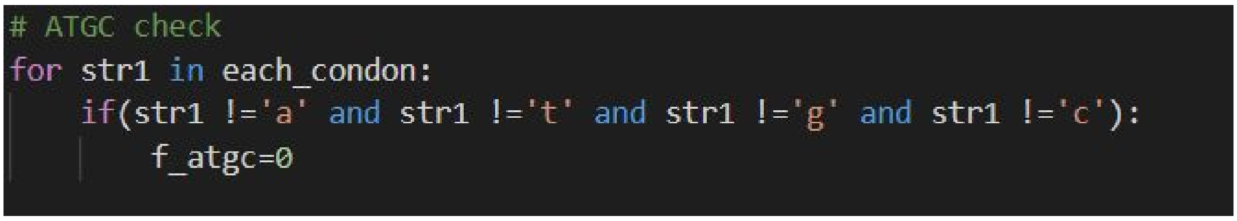
```
while i<len(string):
    each_condon=string[i:i+3] # taking a codon at a time ( group of 3 gene sequence)
    #print(each_condon) # comment this for
    gene_count+=len(each_condon)
    i=i+3
    count+=1
```

# here String is the entire gene sequence. gene_count counts the total no. of nucleotide present. Var count counts the no. of codons.



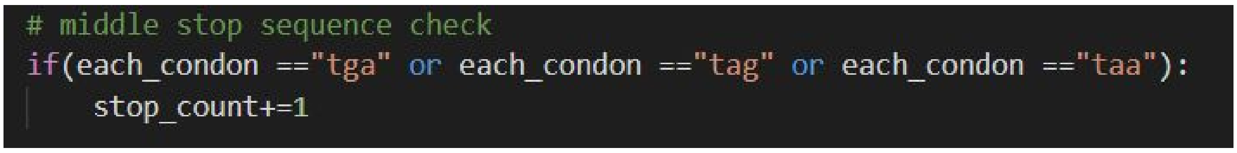
```
if( count == 0): # check gene sequence is empty or not
    is_empty=0
    #print("\ngene sequence is empty\n")
    output_string = output_string + "ERROR: gene sequence is empty,"
else:
    is_empty=1
    #print("\ngene sequence is not empty\n")
    output_string = output_string + "gene sequence is not empty,"
```

# This code checks whether the gene sequence is empty or not. Is_empty flag is used to push an error message in the database,



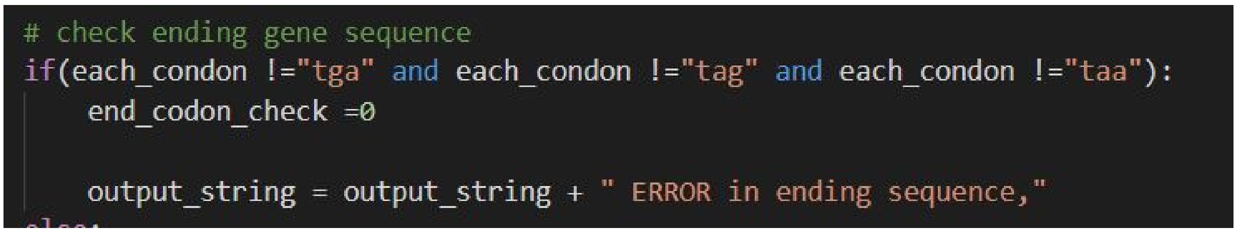
```
# ATGC check
for str1 in each_condon:
    if(str1 !='a' and str1 !='t' and str1 !='g' and str1 !='c'):
        f_atgc=0
```

# here we check if each nucleotide in a codon consists of a,t,g,c or not. If not we set a flag f_atgc = 0, which we will use for generating error messages.



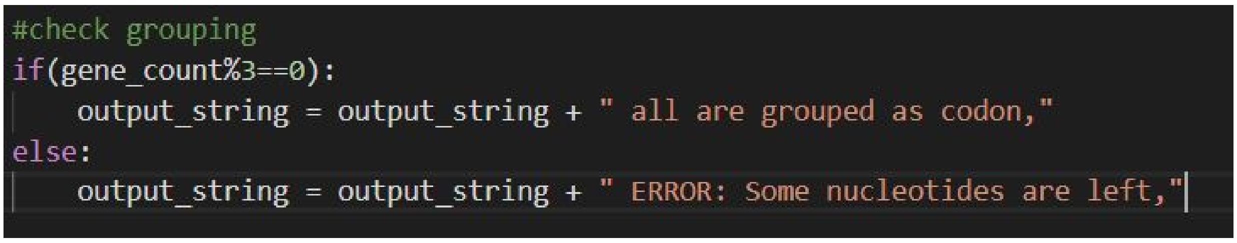
```python
# middle stop sequence check
if(each_condon =="tga" or each_condon =="tag" or each_condon =="taa"):
    stop_count+=1
```

#This code in the while loop check for stop sequence in between the gene sequence. If stop_count is more than 1, then it is assured that stop sequence has occurred in between.



```python
# check ending gene sequence
if(each_condon !="tga" and each_condon !="tag" and each_condon !="taa"):
    end_codon_check =0

    output_string = output_string + " ERROR in ending sequence,"
```

# this code is checked at the end of the while loop, to validate the occurrence of stopping sequence at the end, if not a flag is raised for pushing an error message in the database.



```python
#check grouping
if(gene_count%3==0):
    output_string = output_string + " all are grouped as codon,"
else:
    output_string = output_string + " ERROR: Some nucleotides are left,"
```

# this code checks whether all the nucleotides are grouped into codons or not, by dividing the total gene count by 3.

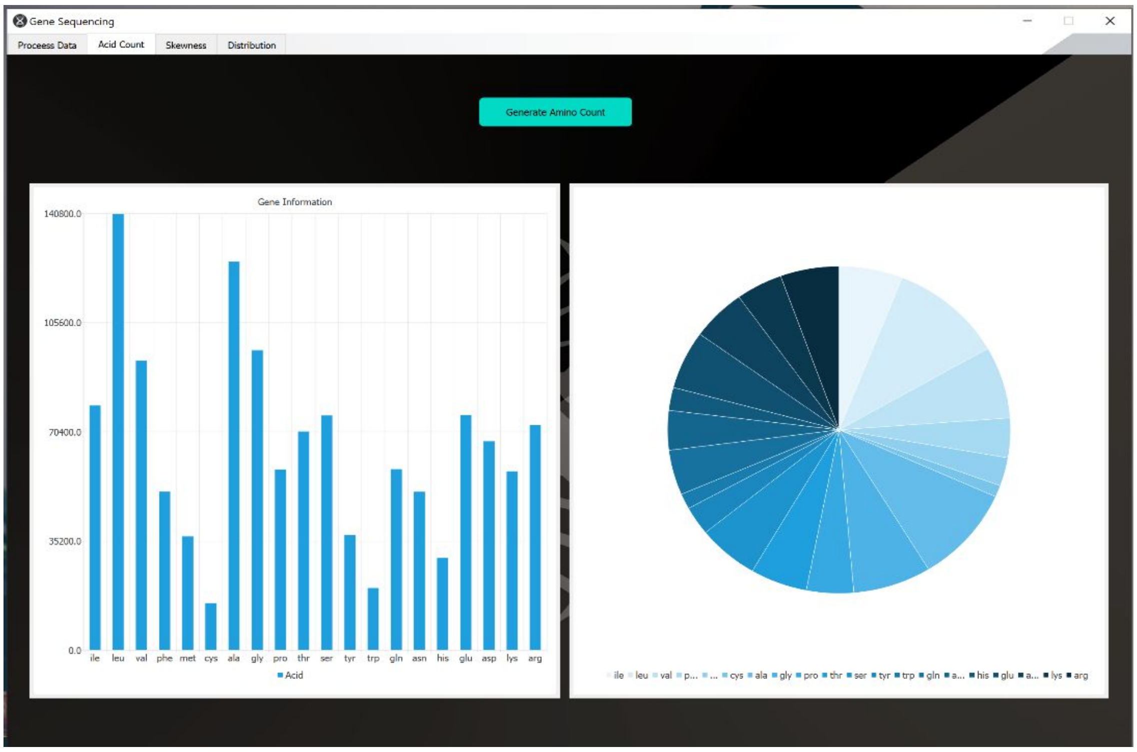
- Module 3: Amino Acid Analysis

  This module gets total amino acid count form the database for the entire gene sequence of a particular organism. Plotting is done with the help of QChart and QChartview of PyQt5 Library.

  It’s easier to analyse the quantity of each amino acid from both the bar diagram and pie chart.

```
from PyQt5 import QtWidgets, QtGui
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *

from PyQt5.QtChart import QChart, QChartView, QBarCategoryAxis,
QValueAxis, QBarSet, QBarSeries, QPieSeries, QPieSlice
from PyQt5.Qt import Qt



import sys
import os

from query import *

class acidCount_module(QWidget):
    def __init__(self, moduleName, *args, **kwargs):
        super(acidCount_module, self).__init__(*args, **kwargs)
        self.setAutoFillBackground(True)


        oImage = QImage("background_tab.svg")
        sImage = oImage.scaled(QSize(2500, 1667))
        palette = QPalette()
        palette.setBrush(10,QBrush(sImage))
        self.setPalette(palette)

        parseData = QPushButton(self, text='Generate Amino Count')
        parseData.pressed.connect(self.showChart)
        parseData.setStyleSheet("QPushButton{background-color: #03DAC5; padding: 20px; border-radius: 10px; font-size: 22px; width: 300px;}" "QPushButton:hover{background-color: white;}")
        parseData.move(1050,100)

    def showChart(self):
        aminoSet = QBarSet('Acid')
        labels = []

        pieSeries = QPieSeries()

        session = DBsession()
        for aminoacids in session.query(Total_table).filter_by(type = "Aminoacid").all():
            aminoSet.append(aminoacids.count)
            labels.append(aminoacids.element)
            pieSeries.append(aminoacids.element, aminoacids.count)
        session.close()
        aminoSeris = QBarSeries()
        aminoSeris.append(aminoSet)

        pieChart = QChart()
        pieChart.addSeries(pieSeries)
        pieChart.createDefaultAxes()
        pieChart.setAnimationOptions(QChart.SeriesAnimations)

        pieChart.legend().setVisible(True)
        pieChart.legend().setAlignment(Qt.AlignBottom)

        pieChartview = QChartView(pieChart, self)
        pieChartview.setRenderHint(QPainter.Antialiasing)

        pieChartview.move(1250, 300)
        pieChartview.setFixedHeight(1200)
        pieChartview.setFixedWidth(1200)
        pieChartview.show()

        chart = QChart()
        chart.addSeries(aminoSeris)
        chart.setTitle('Gene Information')
        chart.setAnimationOptions(QChart.SeriesAnimations)

        labels = tuple(labels)
        axisX=QBarCategoryAxis()
        axisX.append(labels)

        chart.createDefaultAxes()
        chart.setAxisX(axisX, aminoSeris)

        chart.legend().setVisible(True)
        chart.legend().setAlignment(Qt.AlignBottom)
        chartview = QChartView(chart, self)
        chartview.setRenderHints(QPainter.Antialiasing)
        chartview.setFixedHeight(1200)
        chartview.setFixedWidth(1180)
        chartview.move(50, 300)
        chartview.show()
```

Output:


Gene Sequencing
Process Data
Acid Count
Skewness
Distribution
Generate Amino Count
Gene Information
140800.0
105600.0
70400.0
35200.0
0.0
ile leu val phe met cys ala gly pro thr ser tyr trp gln asn his glu asp lys arg
Acid
ile leu val p... ... cys ala gly pro thr ser tyr trp gln a... his glu a... lys arg

- Module 4: Skew Diagrams

  The main focus of this module is as follows:
    - Implementation of the cumulative a-t and g-c skew diagrams

Module's Pseudo-code + explanation :

1. Open the file, that contains the total nucleotides sequence.
2. From the file, read a total of 1000 (window_size) number of nucleotides.
3. Now in the read nucleotide series (of 1000 nucleotides), count the number of occurrences of nucleotides 1 and 2 (either 'g' and 'c' for GC-skew or 'a' and 't' for AT-skew) and store them in n1 and n2 using the following function:

```
def count_nucleotides(self, sequence, nucleotide_1):
       return sequence.count(nucleotide_1)
```

4. Now calculate skew for the n1 and n2 using the following function:

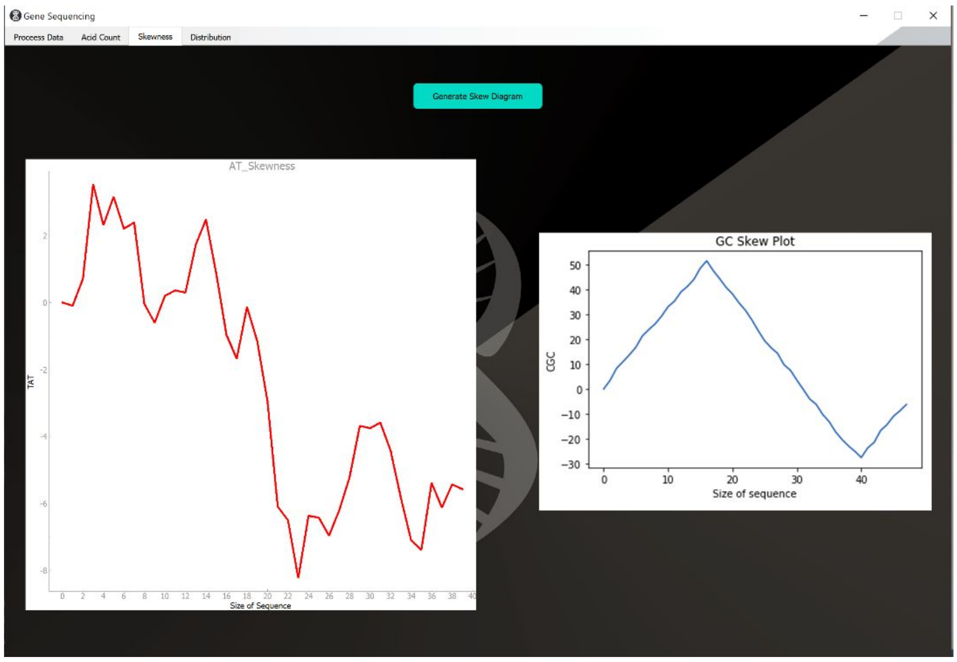
```
   def calculate_skew(self, n1,n2):
        return  (n1-n2)/(n1+n2)
```

5. Append the skew value to a list (skew_value_list).
6. Increment the step_counter.
7. If step_counter is a multiple of step_size:
    a. Append the sum of the last cumulative_value (from the cumulative_value list) with the sum of the contents of the skew_value_list
8. Else continue to step 2

Code for the GC-skew and AT-skew diagrams :

```
   def create_skew(self, window_size=1000,step_size=100):
        geneSequenceFilePointer=open("output/processedFile.txt")
        skew_values_list=[]
        cummulative_values=[0]
        i=0
        while (1):
            read_sequence=geneSequenceFilePointer.read(window_size)
            if(read_sequence==''):
                break
            n1=self.count_nucleotides(read_sequence,"a")
            n2=self.count_nucleotides(read_sequence,"t")
            skew=self.calculate_skew(n1,n2)
            skew_values_list.append(skew)
            i=i+1
            if(i%step_size==0 and i!=0):

cummulative_values.append(cummulative_values[-1]+sum(skew_values_list))
                skew_values_list=[]

        geneSequenceFilePointer.close()

        x=[i for i in range(len(cummulative_values))]

        AT_graph = pg.PlotWidget(self)
        AT_graph.setBackground('w')

        pen = pg.mkPen(color=(255,0,0), width=5)
        AT_graph.plot(x, cummulative_values, pen=pen)
        AT_graph.setTitle("AT_Skewness")
        AT_graph.setLabel('left', 'TAT', color='black', size=30)
        AT_graph.setLabel('bottom', 'Size of Sequence', color='black',
size=40)
        AT_graph.setRenderHints(QPainter.Antialiasing)
        AT_graph.setFixedWidth(1190)
        AT_graph.setFixedHeight(1190)
        AT_graph.move(55, 300)
        AT_graph.show()

        geneSequenceFilePointer=open("output/processedFile.txt")
        skew_values_list=[]
        cummulative_values=[0]
        i=0
        while (1):
            read_sequence=geneSequenceFilePointer.read(window_size)
            if(read_sequence==''):
                break
            n1=self.count_nucleotides(read_sequence,"g")
            n2=self.count_nucleotides(read_sequence,"c")
            skew=self.calculate_skew(n1,n2)
            skew_values_list.append(skew)
            i=i+1
            if(i%step_size==0 and i!=0):

cummulative_values.append(cummulative_values[-1]+sum(skew_values_list))
                skew_values_list=[]

        geneSequenceFilePointer.close()

        x=[i for i in range(len(cummulative_values))]

        GC_graph = pg.PlotWidget(self)
        GC_graph.setBackground('w')

        pen = pg.mkPen(color=(255,0,0), width=5)
        GC_graph.plot(x, cummulative_values, pen=pen)
        GC_graph.setTitle("GC_Skewness")
        GC_graph.setLabel('left', 'TAT', color='black', size=30)
        GC_graph.setLabel('bottom', 'Size of Sequence', color='black',
size=40)
        GC_graph.setRenderHints(QPainter.Antialiasing)
        GC_graph.setFixedWidth(1190)
        GC_graph.setFixedHeight(1190)
        GC_graph.move(1265, 300)
        GC_graph.show()
```

Output:



- ## Module 5: Distribution Analysis

The main focus of this module are as follows:

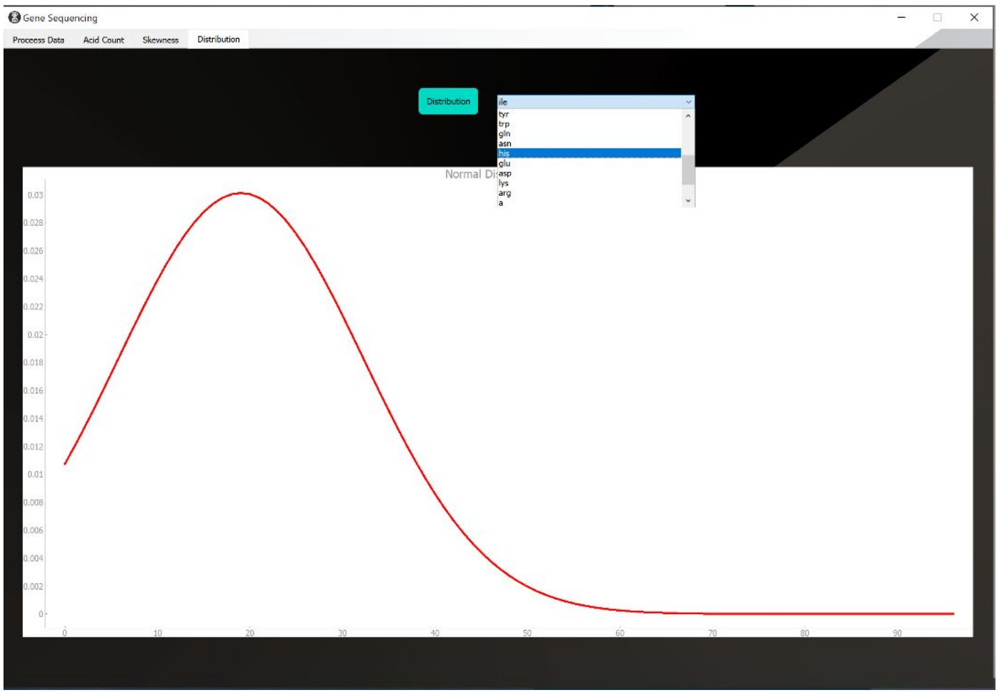
1. Give the overall stats of the presence of different proteins in a gene sequence and their distribution over different gene sequences.
2. In addition to 20 amino acids we have included A G T and C nucleotides too in distribution.

Module's Pseudo-code and explanation :
Data is taken from the database and analysed with numpy, scipy to get the Normal Distribution. All the graphs are plotted with pyqtgraph. Amino acid or nucleotide can be easily selected from the drop down menu which realises QCombobox widget. After selecting with just press of a button distributions can be generated.

```
from PyQt5 import QtWidgets, QtGui
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *

import sys
import os

from pyqtgraph import PlotWidget, plot
import pyqtgraph as pg

import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import norm

from query import *

class distribution_module(QWidget):
    def __init__(self, moduleName, *args, **kwargs):
        super(distribution_module, self).__init__(*args, **kwargs)
        self.setAutoFillBackground(True)



        oImage = QImage("background_tab.svg")
        sImage = oImage.scaled(QSize(2500, 1667))
        palette = QPalette()
        palette.setBrush(10,QBrush(sImage))
```

```
            self.setPalette(palette)

            parseData = QPushButton(self, text='Distribution')
            parseData.pressed.connect(self.generate_distribution)
            parseData.setStyleSheet("QPushButton{background-color:
#03DAC5; padding: 20px; border-radius: 10px; font-size: 22px;}"
"QPushButton:hover{background-color: white;}")
            # parseData.setFixedHeight()
            # parseData.setFixedWidth()
            parseData.move(1050, 100)
            parseData.show()

            normalDistribution_element = []
            session = DBsession()
            for aminoacids in session.query(Total_table).filter_by(type
= "Aminoacid").all():
                normalDistribution_element.append(aminoacids.element)
            for aminoacids in
session.query(Total_table).filter_by(type="Nucleotide").all():
                normalDistribution_element.append(aminoacids.element)
            session.close()

            layout = QHBoxLayout()
            Vlayout = QVBoxLayout(self)
            self.dropDownMenu = QComboBox()
            self.dropDownMenu.addItems(normalDistribution_element)
            self.dropDownMenu.setFixedWidth(500)
            self.dropDownMenu.setFixedHeight(35)
            layout.addStretch(5)
            layout.addWidget(self.dropDownMenu)
            layout.addStretch(3)
            Vlayout.addStretch(2)
            Vlayout.addLayout(layout)
            Vlayout.addStretch(27)

        def generate_distribution(self, value=1):

            aminoCount = []
            entity = self.dropDownMenu.currentText()
            session = DBsession()
            for aminoacids in session.query(Gene_table).all():
                aminoCount.append(getattr(aminoacids, entity))
            session.close()

            aminoCount.sort()
            hmean = np.mean(aminoCount)
            hstd = np.std(aminoCount)
            pdf = norm.pdf(aminoCount, hmean, hstd)

            distributionGraph = pg.PlotWidget(self)
            distributionGraph.setBackground('w')

            pen = pg.mkPen(color=(255,0,0), width=5)
            distributionGraph.plot(aminoCount, pdf, pen=pen)
            title = "Normal Distribution of " + entity
            distributionGraph.setTitle(title)
            distributionGraph.setRenderHints(QPainter.Antialiasing)
            distributionGraph.setFixedWidth(2400)
            distributionGraph.setFixedHeight(1190)
            distributionGraph.move(50, 300)
            distributionGraph.show()
```

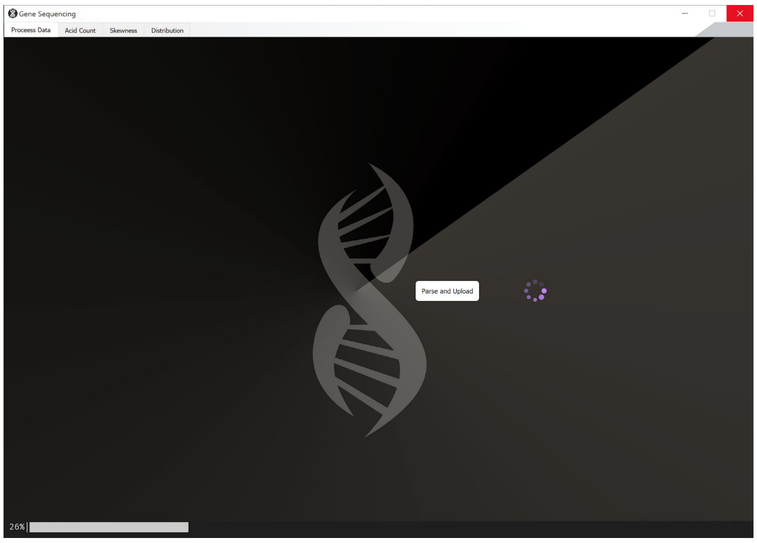
Output:

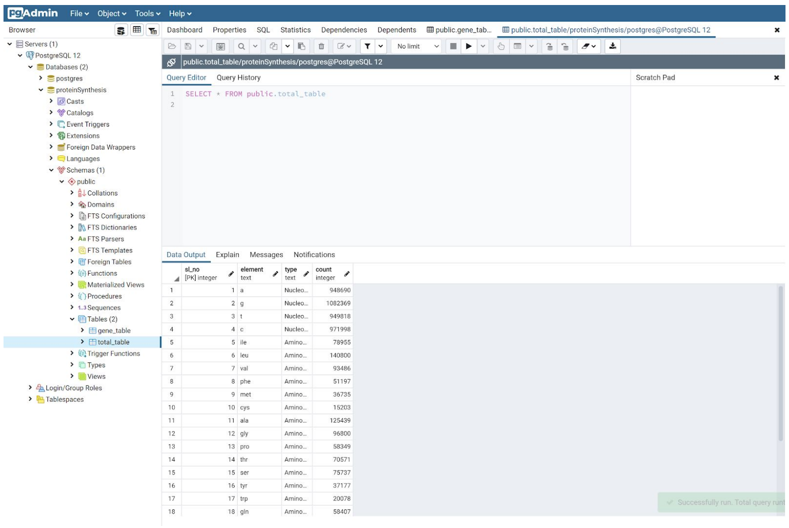


- ## Module 6 : Integrating with the GUI:

  For the final module of the assignment, we have integrated all the above modules together neatly with a GUI. The GUI part was implemented using PyQt5 (Python).

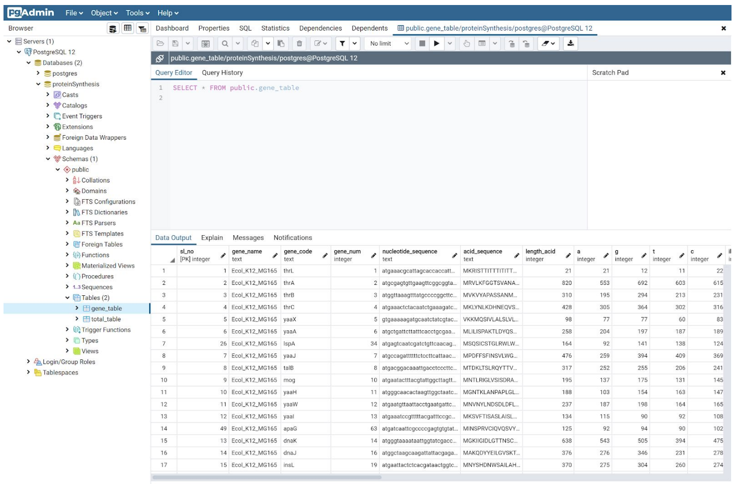
  Screenshots of the GUI are as shown below:





```
SELECT * FROM public.total_table
```

	sl_no [PK] integer	element text	type text	count integer
1	1	a	Nucleo...	948690
2	2	g	Nucleo...	1082369
3	3	t	Nucleo...	949818
4	4	c	Nucleo...	971998
5	5	ile	Amino...	78955
6	6	leu	Amino...	140800
7	7	val	Amino...	93486
8	8	phe	Amino...	51197
9	9	met	Amino...	36735
10	10	cys	Amino...	15203
11	11	ala	Amino...	125439
12	12	gly	Amino...	96800
13	13	pro	Amino...	58349
14	14	thr	Amino...	70571
15	15	ser	Amino...	75737
16	16	tyr	Amino...	37177
17	17	trp	Amino...	20078
18	18	gln	Amino...	58407



```
SELECT * FROM public.gene_table
```

	sl_no [PK] integer	gene_name text	gene_code text	gene_num integer	nucleotide_sequence text	acid_sequence text	length_acid integer	a integer	g integer	t integer	c integer
1	1	Ecol_K12_MG165	thrL	1	atgaaacgcattagcaccaccatt...	MKRISTTITTTITITT...	21	21	12	11	22
2	2	Ecol_K12_MG165	thrA	2	atgcgagtgttgaagttcggcggta...	MRVLKFGGTSVANA...	820	553	692	603	615
3	3	Ecol_K12_MG165	thrB	3	atggttaaagtttatgccccggcttc...	MVKVYAPASSANM...	310	195	294	213	231
4	4	Ecol_K12_MG165	thrC	4	atgaaactctacaatctgaaagatc...	MKLYNLKDHNEQVS...	428	305	364	302	316
5	5	Ecol_K12_MG165	yaaX	5	gtgaaaaagatgcaatctatcgtac...	VKKMQSIVLALSLVL...	98	77	77	60	83
6	6	Ecol_K12_MG165	yaaA	6	atgctgattcttatttcacctgcgaa...	MLILISPAKTLDYQS...	258	204	197	187	189
7	26	Ecol_K12_MG165	lspA	34	atgagtcaatcgatctgttcaacag...	MSQSICSTGLRWLW...	164	92	141	138	124
8	7	Ecol_K12_MG165	yaaJ	7	atgccagatttttctccttcattaac...	MPDFFSFINSVLWG...	476	259	394	409	369
9	8	Ecol_K12_MG165	talB	8	atgacggacaaattgacctcccttc...	MTDKLTSLRQYTTV...	317	252	255	206	241
10	9	Ecol_K12_MG165	mog	10	atgaatactttacgtattggcttagtt...	MNTLRIGLVSISDRA...	195	137	175	131	145
11	10	Ecol_K12_MG165	yaaH	11	atgggcaacactaagttggctaatc...	MGNTKLANPAPLGL...	188	103	154	163	147
12	11	Ecol_K12_MG165	yaaW	12	atgaatgttaattacctgaatgattc...	MNVNYLNDSDLDFL...	237	187	198	164	165
13	12	Ecol_K12_MG165	yaaI	13	atgaaatccgtttttacgatttccgc...	MKSVFTISASLAISL...	134	115	90	92	108
14	49	Ecol_K12_MG165	apaG	63	atgatcaattcgccccgagtgtgtat...	MINSPRVCIQVQSVY...	125	92	94	90	102
15	13	Ecol_K12_MG165	dnaK	14	atgggtaaaataattggtatcgacc...	MGKIIGIDLGTTNSC...	638	543	505	394	475
16	14	Ecol_K12_MG165	dnaJ	16	atggctaagcaagattattacgaga...	MAKQDYYEILGVSKT...	376	276	346	231	278
17	15	Ecol_K12_MG165	insL	19	atgaattactctcacgataactggtc...	MNYSHDNWSAILAH...	370	275	304	260	274

6. Further Scope for Development:

The work done by us in this assignment is just the peak of the mountain of developments, which can be implemented to develop an effective analyzing tool. We sincerely believe that this work of ours, if developed further can evolve into a state-of-the-art analyzing system, which can be used by researchers/students for their research and analysis purposes.

One of the features that can be integrated with our system is the implementation of an online central server. This can be used to share and view the research done by different people around the globe. Another feature may be the inclusion of other diagnosis tools like the g-c/a-t skew analysis.

Many more such ideas can be integrated with the existing system and take it to the next level.

7. Conclusion:

This lab assignment provided us a very gentle introduction to the interdisciplinary field of work. We were not only exposed to the interaction between computer science and bio-science but also were able to successfully apply the Agile Development methodology. This assignment has surely highlighted the importance of good software engineering practices in the development of a project.
We were forced to carefully design the layout of the system and select the best compatible sets of tools available. We also got to experience first-hand the gravity of co-operation within the team members and effective division of work.

Overall it was a great learning as well as working experience for all of us. We really look forward to engaging in such future endeavors again.

References:

- PyQt 5
- Postgre SQL
- Matplotlib
- Geeks for geeks
- Software Engineering Book- Rajib Mall

Code & DEMO:

- Link