HW1

January 29, 2021

[1]:
```
"""
Returns the data on the length of each chromosome and the given chromosome,␣
 ↪each with the same index.
"""

def get_chrom_lens(gen_file):
    chroms = []
    lens = []
    for line in gen_file:
        chrom = line.split("\t")[0]
        length = int(line.split("\t")[1])
        chroms.append(chrom)
        lens.append(length)
    return chroms, lens
```

[2]:
```
"""
Solve 1.1-5
"""
def calc_vals(chroms, lens):
    return_vals = []

    #1.1
    tot_size = 0
    for num in lens:
        tot_size = tot_size + num
    return_vals.append(tot_size)

    #1.2
    num_chroms = len(chroms)
    return_vals.append(num_chroms)

    #1.3,4
    largest_chrom_name = -1
    largest_chrom_size = -1
    smallest_chrom_name = -1
    smallest_chrom_size = -1
    for i in range(0, num_chroms):
```

```
        if lens[i] > largest_chrom_size or i == 0:
            largest_chrom_name = chroms[i]
            largest_chrom_size = lens[i]
        if lens[i] < smallest_chrom_size or i == 0:
            smallest_chrom_name = chroms[i]
            smallest_chrom_size = lens[i]
    return_vals.append(largest_chrom_name)
    return_vals.append(largest_chrom_size)
    return_vals.append(smallest_chrom_name)
    return_vals.append(smallest_chrom_size)

    #1.5
    mean_len = 1.0 * tot_size / num_chroms
    return_vals.append(mean_len)

    return return_vals
```

[3]:
```
"""
Question 1
"""
aratha = open("arabidopsisTAIR10.chrom.sizes", "r")
dromel = open("drosophilalidm6.chrom.sizes", "r")
esccol = open("ecoli.chrom.sizes", "r")
homsap = open("hg38.chrom.sizes", "r")
triaes = open("wheat.chrom.sizes", "r")
saccer = open("yeast.chrom.sizes", "r")
sollyc = open("tomato.chrom.sizes", "r")
caeele = open("ce10.chrom.sizes", "r")

#I do not know why it isn't letting me just feed directly into the next␣
 ↪function; super annoying and I'm mad how ugly this code is
chroms, lens = get_chrom_lens(aratha)
aratha_data = calc_vals(chroms, lens)

chroms, lens = get_chrom_lens(dromel)
dromel_data = calc_vals(chroms, lens)

chroms, lens = get_chrom_lens(esccol)
esccol_data = calc_vals(chroms, lens)

chroms, lens = get_chrom_lens(homsap)
homsap_data = calc_vals(chroms, lens)

chroms, lens = get_chrom_lens(triaes)
triaes_data = calc_vals(chroms, lens)

chroms, lens = get_chrom_lens(saccer)
```

```
saccer_data = calc_vals(chroms, lens)

chroms, lens = get_chrom_lens(sollyc)
sollyc_data = calc_vals(chroms, lens)

chroms, lens = get_chrom_lens(caeele)
caeele_data = calc_vals(chroms, lens)

"""
Plot it!
"""
import pandas as pd
data = {"Aratha": aratha_data, "Dromel": dromel_data, "Esccol": esccol_data,␣
 ↪"Homsap": homsap_data, "Triaes": triaes_data, "Saccer": saccer_data,␣
 ↪"Sollyc": sollyc_data, "Caeele": caeele_data}
df = pd.DataFrame(data)
df.rename(index={0: "Total Genome Size", 1: "Num Chromosomes", 2: "Largest␣
 ↪Chrom. Name", 3: "Largest Chrom. Size", 4: "Smallest Chrom. Name", 5:␣
 ↪"Smallest Chrom. Size", 6: "Mean Len"})
```

[3]:

	Aratha	Dromel	Esccol	Homsap
Total Genome Size	119146348	137547960	4639211	3088269832
Num Chromosomes	5	7	1	24
Largest Chrom. Name	Chr1	chr3R	Ecoli	chr1
Largest Chrom. Size	30427671	32079331	4639211	248956422
Smallest Chrom. Name	Chr4	chr4	Ecoli	chr21
Smallest Chrom. Size	18585056	1348131	4639211	46709983
Mean Len	2.38293e+07	1.96497e+07	4.63921e+06	1.28678e+08

	Triaes	Saccer	Sollyc	Caeele
Total Genome Size	14547261565	12157105	782520033	100286070
Num Chromosomes	22	17	13	7
Largest Chrom. Name	3B	chrIV	ch01	chrV
Largest Chrom. Size	830829764	1531933	90863682	20924149
Smallest Chrom. Name	6D	chrM	ch00	chrM
Smallest Chrom. Size	473592718	85779	9643250	13794
Mean Len	6.61239e+08	715124	6.01938e+07	1.43266e+07

[4]:
```
"""
Question 2
"""
chr22_file = open("chr22.fa/chr22.txt", "r")
chr22 = []
for line in chr22_file:
    for c in line:
        if c is not "\n":
            chr22.append(c.upper())
```

[5]:
```
atcgn = [0, 0, 0, 0, 0]
for base in chr22:
    if base == "A":
        atcgn[0] = atcgn[0] + 1
    elif base == "T":
        atcgn[1] = atcgn[1] + 1
    elif base == "C":
        atcgn[2] = atcgn[2] + 1
    elif base == "G":
        atcgn[3] = atcgn[3] + 1
    elif base == "N":
        atcgn[4] = atcgn[4] + 1
```

[6]:
```
"""
1.1
"""
import pandas as pd
df = pd.DataFrame({"A": atcgn[0], "T": atcgn[1], "C": atcgn[2], "G": atcgn[3],␣
↪"N": atcgn[4]}, index = [0])
df.rename(index={0: "Num Bases"})
```

[6]:

	A	T	C	G	N
Num Bases	10382214	10370725	9160652	9246186	11658691

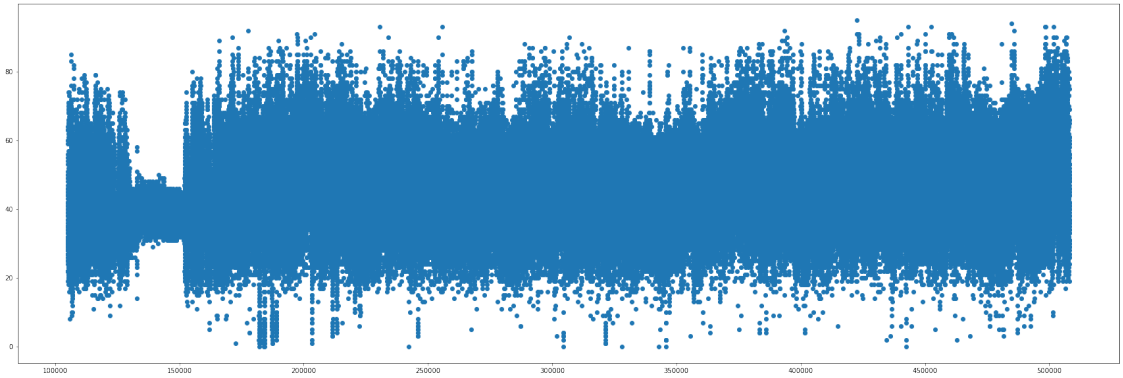
[7]:
```
"""
Break into 100bp bins
"""
import numpy as np
import math
num_base = 0
for n in atcgn:
    num_base = num_base + n
num_cols = math.ceil(1.0 * num_base / 100)
bins = np.chararray((100, num_cols))
i = 0
j = 0
for base in chr22:
    bins[i, j] = base
    i = i+1
    if i == 100:
        j = j+1
        i = 0
```

[9]:
```
"""
2.2 and getting info for 2.3
"""
with_n = 0
```

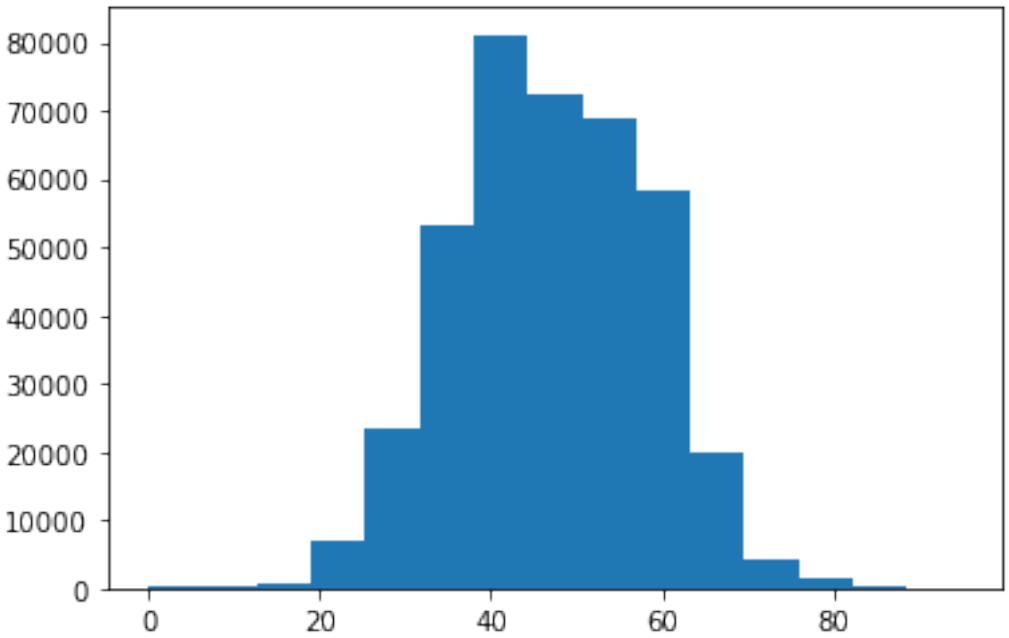
```
has_n = np.zeros(num_cols)
gc_content = np.zeros(num_cols)
i = 0
j = 0
while j < num_cols and i < 100:
        if bins[i][j] == b'N': #numpy array stores the chars as byte type
            has_n[j] = 1
            with_n = with_n + 1
            j = j + 1 # if we've already found an N, skip the rest of the bin.
            i = 0
        else:
            if bins[i][j] == b'G' or bins[i][j] == b'C':
                gc_content[j] = gc_content[j] + 1
            i = i+1
            if i == 100:
                j = j+1
                i = 0
print("Fraction of the 100bp non-overlapping windows/bins across chromosome 22␣
 ↪contain at least one N: " + str(1.0* with_n / num_cols))
```

```
Fraction of the 100bp non-overlapping windows/bins across chromosome 22 contain
at least one N: 0.2295168098231943
```

[10]:
```
"""
2.3: making the scatterplot
"""

gc_no_n = np.zeros(num_cols - with_n)
chrom_loc = np.zeros(num_cols - with_n)

j = 0 #index the data to be graphed
for i in range(0, num_cols): #index the raw data with Ns
    if has_n[i] == 0:
        gc_no_n[j] = gc_content[i]
        chrom_loc[j] = i
        j = j +1

import matplotlib.pyplot as plt
f = plt.figure(figsize = (30, 10))
fig = plt.scatter(chrom_loc, gc_no_n)
plt.show()
```



[11]:
```
"""
2.4
"""
fig = plt.hist(gc_no_n, bins=15)
plt.show()
```



[12]:
```
"""
2.5
"""
num_wo_n = num_cols - with_n
num_out_of_range = 0
for val in gc_no_n:
```

```
    if val <= 30 or val >=65:
        num_out_of_range = num_out_of_range + 1
print("Fraction of bins expected to sequence poorly: " + str(1.0*␣
 ↪num_out_of_range/num_wo_n))
```

```
Fraction of bins expected to sequence poorly: 0.12019471431344306
```

[13]:
```
"""
2.6
"""
print("Expected bases correctly sequenced in human genome: " + str(int((1.0*␣
 ↪num_out_of_range/num_wo_n) * 3088269832)))
```

```
Expected bases correctly sequenced in human genome: 371193710
```

[ ]:
```

```