QUESTION 4d: Compare the expected count and actual counts for each DNA sequence.

ANSWER:

When I compare the expected and actual counts across different organism, we see notable deviations. MluCI and HaeIII have significantly higher actual counts than expected in humans, while HpyCH4IV is much lower than expected in humans but closer in other species. MfeI appears much more frequently in yeast, whereas EagI-HF®, ZraI, and BssSI-v2 are consistently lower than expected across all organisms. PacI is slightly more abundant than expected, while NotI is significantly underrepresented in all cases. These variations suggest that DNA sequence composition differs across species, possibly due to evolutionary or functional constraints.

0.0.1 1.3 Accounting for GC Bias

One explanation for the observed results above is that we assumed the DNA nucleotide frequencies were equal in our expectation value calculation. This may not be the case for human and yeast DNA. Therefore, it may be useful to account for bias in nucleotide frequency in our theoretical count function.

QUESTION 5a: Build the function `theoretical_count_with_GC` which returns the expectation value for the number of occurances of a binding recognition seq, `target_seq`, in a DNA sequence with `DNA_length` number of nucleotides, incorporating DNA nucleotide frequency information from the `DNA_GC_proportion` of the input DNA. Ensure that it outputs a whole number. > Hint: > - What is the probability of observing the sequence "ATCG" if the probability of each nucleotide is different? > - If you aren't sure how to determine the nucleotide frequencies/probabilities from only the GC content, consider Chargaff's Rules.

```
In [26]: def theoretical_count_with_GC(DNA_length, target_seq, DNA_GC_proportion):
             at_count_of_target, cg_count_of_target = target_seq.count("A") + target_seq.count("T"), ta
             DNA_AT_proportion = 1 - DNA_GC_proportion
             match_probability = ((DNA_AT_proportion/2) ** (at_count_of_target/2)) * ((DNA_GC_proportio
             expected_num_matches = match_probability * DNA_length
             return round(expected_num_matches, 0)
```

```
In [27]: grader.check("q5a")
```

```
Out[27]: q5a results: All test cases passed!
```

QUESTION 5b: Create a new DataFrame, `re_frequencies_GC` with the following columns, in order: - `"Enzyme"`: str, the name of the restriction enzyme. - `"Recognition Sequence"`: str, the binding recognition sequence for the enzyme. - `"Expected Count Human"`: int, the expected value of observing the recognition sequence in `human_DNA`. - `"Actual Count Human"`: int, the actual number of recognitition sequence matches in `human_dna`. - `"Expected Count Yeast"`: int, the expected value of observing the recognition sequence in `yeast_DNA`. - `"Actual Count Yeast"`: int, the actual number of recognitition sequence matches in `yeast_dna`. - `"Expected Count E. coli"`: int, the expected value of observing the recognition sequence

in `e_coli_DNA`. - `"Actual Count E. coli"`: int, the actual number of recognitition sequence matches in `e_coli_DNA`.

In [28]:
```
expected_counts_GC_human = []
expected_counts_GC_yeast = []
expected_counts_GC_e_coli = []

for target_seq in restriction_enzymes_df["Recognition Sequence"]:
    expected_counts_GC_human.append(theoretical_count_with_GC(1000000, target_seq, gc_df.iloc[
    expected_counts_GC_yeast.append(theoretical_count_with_GC(1000000, target_seq, gc_df.iloc[
    expected_counts_GC_e_coli.append(theoretical_count_with_GC(1000000, target_seq, gc_df.iloc

re_frequencies_GC = restriction_enzymes_df.copy()
re_frequencies_GC["Expected Count Human"] = expected_counts_GC_human
re_frequencies_GC["Actual Count Human"] = actual_counts_human
re_frequencies_GC["Expected Count Yeast"] = expected_counts_GC_yeast
re_frequencies_GC["Actual Count Yeast"] = actual_counts_yeast
re_frequencies_GC["Expected Count E. coli"] = expected_counts_GC_e_coli
re_frequencies_GC["Actual Count E. coli"] = actual_counts_e_coli

re_frequencies_GC
```

Out[28]:

	Enzyme	Recognition Sequence	Expected Count Human	Actual Count Human	Expected Count Yeast	Actual Count Yeast	Expected Count E. coli	Actual Count E. coli
0	MluCI	AATT	5803.0	5645	9235.0	9375	3516.0	4147
1	HaeIII	GGCC	2518.0	4647	1303.0	1535	4329.0	2804
2	HpyCH4IV	ACGT	3822.0	972	3469.0	2458	3901.0	3065
3	MfeI	CAATTG	291.0	182	333.0	520	231.0	193
4	EagI-HF®	CGGCCG	126.0	78	47.0	34	285.0	55
5	ZraI	GACGTC	192.0	61	125.0	101	257.0	136
6	BssSI-v2	CACGAG	192.0	71	125.0	89	257.0	
7	PacI	TTAATTAA	34.0	21	85.0	35	12.0	
8	NotI	GCGGCCGC	6.0	6	2.0	4	19.0	

```
6                    57
7                    26
8                     4
```

```
In [29]: grader.check("q5b")
```

```
Out[29]: q5b results: All test cases passed!
```

QUESTION 5d: Compare the expected count and actual counts for each DNA sequence. In particular, consider the GC content of the recognition sequences and the GC content of the DNA sequences. How do these results compare to the previous DataFrame in Q4c?

ANSWER: Relative to our initial attempt, the GC-adjusted model clearly does a better job at capturing biological reality. We see that it adjust expectations based on the actual sequence composition of the genome and just brings that expected output closer to actual counts for most enzymes.

In particular we saw that in the previous dataframe, the GC-rich seqeunces are generally underrepresented compared to AT-rich ones. In short the GC-adjusted model reduces overestimation of GC-rich sites in AT-rich genomes and vice versa!

0.0.2 1.4 Hypothesis Testing

In Question 5, we looked at the expected versus actual number of restriction enzyme recognition enzymes. If we want to statistically determine whether this value is different between real human DNA versus random DNA with the same GC content, we could set up a hypothesis test. In this section, we will explore whether the number of MfeI recognition sites is statistically significant in real human DNA compared to random DNA.

QUESTION 6a: Write a null and alternative hypothesis for our hypothesis test.

ANSWER: - Null hypothesis: The number of Mfel recognition sites in real human DNA is not significantly less than from the number in random DNA with the same GC content - Alternative hypothesis: The number of Mfel recognition sites in real human DNA is significantly less than the number in random DNA with the same GC content

QUESTION 6b: Now, we want to simulate results under the null hypothesis. Fill in the code below to generate the array `predicted_counts`, which has the counts of MluCI recognition sites for 250 random DNA sequences with the same GC content and length as `human_DNA`. Your code may take around a minute to run. > Hint: Use your function from Q3c to generate a random DNA sequence with the appropriate GC content.

```
In [30]: MluCI_recognition_sequence = "AATT"
```

```
# GC-content of the human DNA
gc_content_human = gc_df.loc["Human"]["GC content"]

predicted_counts = []
for i in range(250):
    random_seq = generate_random_DNA(gc_content_human, i) #DO NOT CHANGE THE SEED, i.
    count_result = count_seq(random_seq, MluCI_recognition_sequence)
    predicted_counts.append(count_result)
```

In [31]:
```
grader.check("q6b")
```

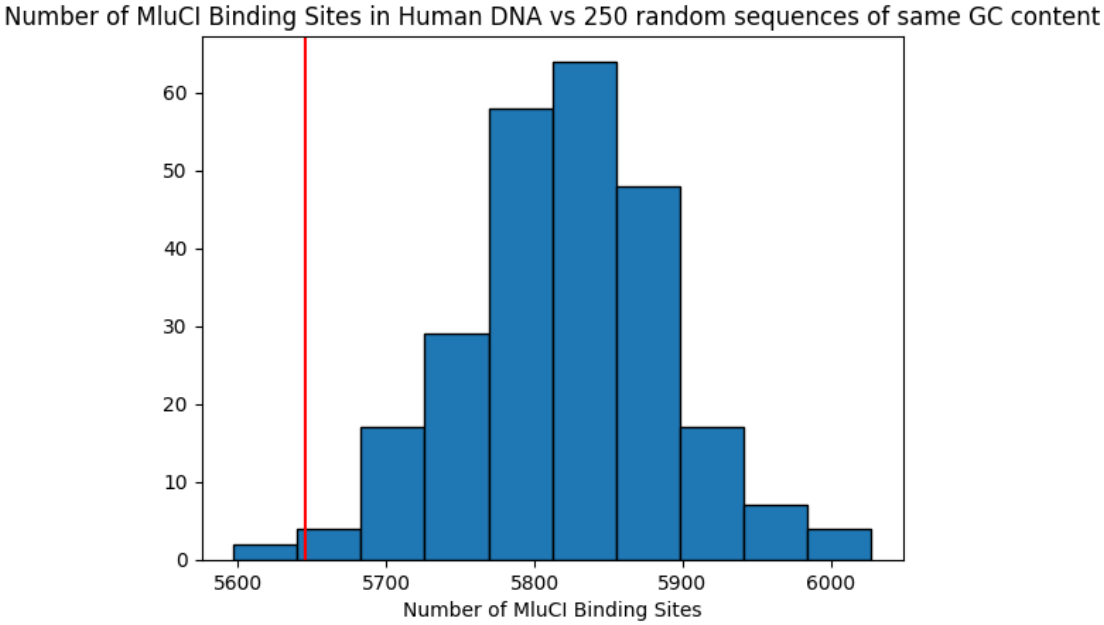
Out[31]:
```
q6b results: All test cases passed!
```

QUESTION 6c: Assign `observed_count` to the number of MluCI recognition sequences in `human_DNA`.

In [32]:
```
observed_count = count_seq(human_dna, MluCI_recognition_sequence)
```

In [33]:
```
#Just run this cell
plt.hist(predicted_counts, fill=True)
plt.axvline(x = observed_count, color = "red")
plt.xlabel("Number of MluCI Binding Sites")
plt.title("Number of MluCI Binding Sites in Human DNA vs 250 random sequences of same GC conte
```

Out[33]:
```
Text(0.5, 1.0, 'Number of MluCI Binding Sites in Human DNA vs 250 random sequences of same GC
```



```
In [34]: grader.check("q6c")
```

```
Out[34]: q6c results: All test cases passed!
```

QUESTION 7a: Calculate the p-value, comparing `observed_count` to `predicted_counts`, and assign it to `p_value`.

```
In [35]: p_value = sum([1 for i in predicted_counts if i <= observed_count]) / len(predicted_counts)

         print(f"The p-value for this hypothesis test is {p_value}.")
```

```
The p-value for this hypothesis test is 0.012.
```

```
In [36]: grader.check("q7a")
```

```
Out[36]: q7a results: All test cases passed!
```

QUESTION 7b: What can we conclude from our results, given a p-value cutoff of 0.05? What about for a p-value of 0.01?

ANSWER: Our results show that there is sigficant statistical evidence at 95% Confidence interval as our p-value $= 0.012 < 0.05$ to reject the null hypothesis. However, at the 99% confidence interval, we accept the null hypothesis!

0.0.3 1.4 Poisson Statistics

The Poisson Probability Distribution

The Poisson probability distribution is a mathematical concept used to model the probability of a given number of **discrete events** happening in a fixed interval of time or space, under certain conditions. It is particularly useful for **rare events** that occur with a known constant mean rate and independently of the time since the last event. For example, the Poisson distribution can be used to model the number of emails a person receives in an hour or the number of stars appearing in a certain area of the sky. The key parameter of this distribution is (lambda), which represents the average number of events in the given

interval. The probability of observing exactly k events (where k is a non-negative integer) is calculated using the formula $P(X = k) = (\lambda^k * e^(-\lambda))/k!$. The mean and variance of this distribution are both equal to . It is unusual for the mean and variance of a distribution to be the same. Often, this occurs in very high-variance distributions.

Recall that in a Gaussian distribution, the mean and variance are independent parameters. A common point of confusion arises when students try to decide between using the Poisson distribution and the Gaussian (or normal) distribution. While both distributions can be used in certain contexts, they are fundamentally different. The Gaussian distribution is typically used for continuous data and is most appropriate when dealing with large sample sizes or events that are not rare. In contrast, the Poisson distribution is better suited for rare events with smaller sample sizes. However, it's important to note that as the sample size increases (i.e., in the limit of large N), the Poisson distribution can converge to a Gaussian distribution, which is a key point in understanding the relationship between these two statistical models.

In this section, we will model the probability of observing a given number of PacI recognition sites ("TTAATTAA") in a random 100,000 nucleotides-long DNA sequence using the Poisson distribution, and then using our model to make probabilistic calculations. Therefore, in our scenario, P(X=2) symbolizes the probability of observing 2 MluCI recognition sites in a DNA sequence of length 100,000 with the same GC content as `human_DNA`.

QUESTION 8a: Assign `lambda_constant` to the value of for a Poisson probability distribution that models the probability of observing a given number of MluCI recognition sites in a random 100,000 nucleotides-long DNA sequence (with the same GC content as a human). > Hint: you may want to use a function that you previously built.

```
In [48]: gc_content_human
```

```
Out[48]: 0.44800000000000001
```

```
In [37]: lambda_constant = theoretical_count_with_GC(100000, MluCI_recognition_sequence, gc_content_huma

         print(f"The lambda for our Poission distribution is {lambda_constant}.")
```

```
The lambda for our Poission distribution is 580.0.
```

QUESTION 8b: Assign `observed_count_100000` to the number of PacI recognition sites in the first 100,000 nucleotides of `human_DNA`.

```
In [38]: # PacI
         PacI_recognition_sequence = "TTAATTAA"
```

```
lambda_constant = theoretical_count_with_GC(100000, PacI_recognition_sequence, gc_content_human)
lambda_constant
```

```
Out[38]: 3.0
```

```
In [39]: # Use the Poisson distribution to calculate the probability of observing the observed count
observed_count_100000 = count_seq(human_dna[:100000], "TTAATTAA")

print(f"There are {observed_count_100000} PacI recognition sites in the first 100,000 nucleotid
```

```
There are 1 PacI recognition sites in the first 100,000 nucleotides of our real human DNA.
```

```
In [40]: grader.check("q8b")
```

```
Out[40]: q8b results: All test cases passed!
```

QUESTION 8c: Use the formula for P(X=k) to calculate the probability of observing exactly `observed_count_100000` number of PacI recognition sites in a random DNA (with the same GC content as human). In other words, calculate P(X = `observed_count_100000`). Assign your results to `p_x_observedcount100000`. > Hint: `math.e` and `math.factorial(int)` may be useful.

```
In [41]: p_x_observedcount100000 = stats.poisson.pmf(observed_count_100000, lambda_constant)

print(f"The probability of observed {observed_count_100000} number of PacI binding sites in ran
```

```
The probability of observed 1 number of PacI binding sites in random DNA of length 100,000 with human GC
```

```
In [42]: grader.check("q8c")
```

```
Out[42]: q8c results: All test cases passed!
```

QUESTION 9a: Fill in the code below to create a list `pmf` (Probability Mass Function), which contains the probability of observing `k` number of PacI recongition sites in random DNA (with the same GC content as human), for up until `k` = 30. In other words, `pmf` should be a list containing P(X=0), P(X=1), P(X=2), and so on.

```
In [43]: k_values = np.arange(0, 31)
         pmf = []
         for k in k_values:
             p_x_k = stats.poisson.pmf(k, lambda_constant)
             pmf.append(p_x_k)
```

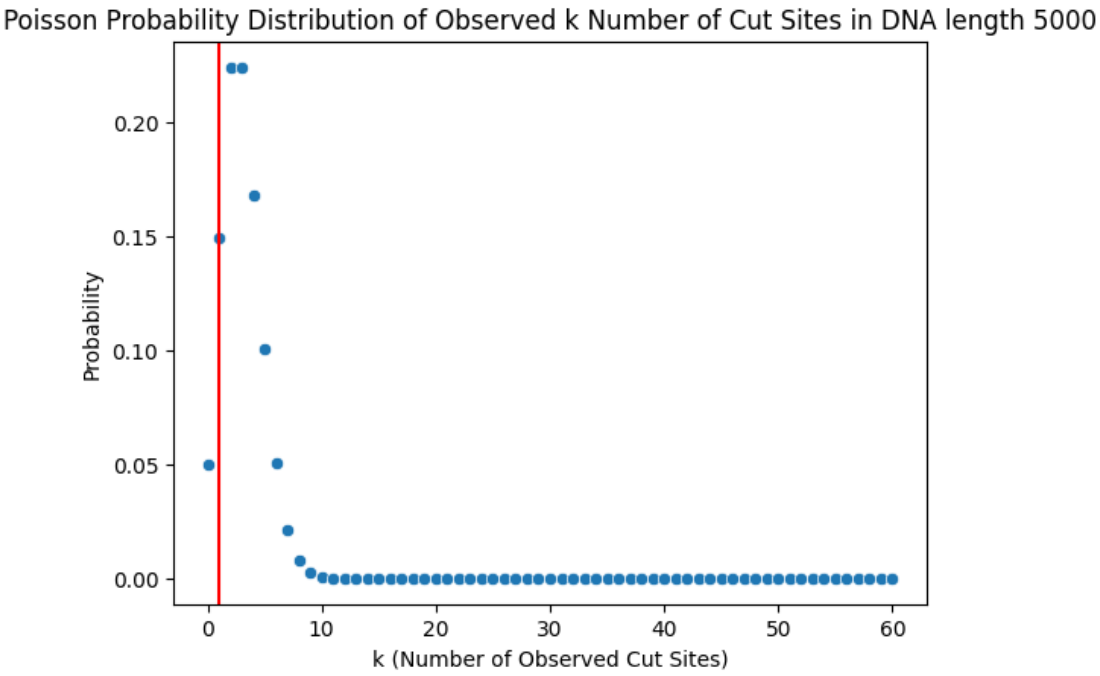
```
In [44]: grader.check("q9a")
```

```
Out[44]: q9a results: All test cases passed!
```

QUESTION 9b: Fill in the code below to generate a `sns` scatterplot of the `pmf` (Probability Mass Function) for values of `k` from 0 to 60 (inclusive). > Note: the vertical red line is the number of PacI sites in 100,000 nucleotides of real DNA that you determined in Q8b.

```
In [45]: k_values = np.arange(0, 61)
         pmf = []
         for k in k_values:
             p_x_k = stats.poisson.pmf(k, lambda_constant)
             pmf.append(p_x_k)

         ax = sns.scatterplot(x = k_values, y = pmf)

         ax.axvline(x = observed_count_100000, color = "red")
         ax.set_xlabel("k (Number of Observed Cut Sites)")
         ax.set_ylabel("Probability")
         ax.set_title("Poisson Probability Distribution of Observed k Number of Cut Sites in DNA length
```

```
Out[45]: Text(0.5, 1.0, 'Poisson Probability Distribution of Observed k Number of Cut Sites in DNA leng
```




```
In [46]: grader.check("q9b")
```

```
Out[46]: q9b results: All test cases passed!
```

QUESTION 9c: Calculate the probability of observing at least 8 PacI cut sites. Assign your result to `p_x_atleast8`. Round your result to three signifcant figures > Note: assume the P(k>30) is negligible for these questions.

```
In [47]: p_x_atleast8 = 1 - stats.poisson.cdf(7, lambda_constant)

         print(f"The probability of observing at least 8 cut sites is  {p_x_atleast8:.4f}.")
```

```
The probability of observing at least 8 cut sites is  0.0119.
```

```
In [71]: grader.check("q9c")
```

```
Out[71]: q9c results: All test cases passed!
```

QUESTION 9d: Calculate the probability of observing between 5 and 10 (inclusive) PacI cut sites. Assign your result to `p_5_x_10`. Round your result to three signficant figures.

```
In [49]: p_5_x_10 = stats.poisson.cdf(10, lambda_constant) - stats.poisson.cdf(4, lambda_constant)

         print(f"The probability of observing between 5 and 10 (inclusive) cut sites is {p_5_x_10:.3f}.
```

```
The probability of observing between 5 and 10 (inclusive) cut sites is 0.184.
```

```
In [50]: grader.check("q9d")
```

```
Out[50]: q9d results: All test cases passed!
```

QUESTION 10: Explain whether must always be discrete for the Poisson distribution. Could the expected number of a cut sites be 1.5? Is it possible to observe 1.5 cut sites? Why?

ANSWER: No the λ parameter of the Poisson distribution does not have to be discrete, in fact we can choose λ to be any non-negative real number and we can also have the continous analog of the poisson known as the exponential distribution.

So yes, the expexted number of cut sites can be 1.5 which would just mean that on average, across many DNA strands of the same length and composition, you'd expect 1.5 cut sites per sequence. However, as the next question wuuld insinuate, you can't actually observe 1.5 cut sites in a single sequence simply because there is no such thing as a "half-cut"

0.1 2. Fragment Lengths

In this final section, we will be exploring trends in the fragment lengths between restriction enzyme cut sites in DNA. Throughout this section, we will assume that a cut site is made immediately at the end of a restriction enzyme binding site. Note the cut location can vary between enzymes but make this assumption for simplicity.

QUESTION 11: Build a function `fragment_lengths` that takes in a `dna_sequence` and a `cut_site_sequence` and returns a list of lengths of the fragments generated. Choose either approach to yeild results, but only pick 1, and delete the other approach. > Hint: > - For example, `fragment_lengths("c", "GGG")` should output a list like [3, 6, 5] (not necessarily in that order). It has three cut sites **after** each "GGG", highlighted: "GGG|TTTGGG|TTGGG|". Your function calculates the distances between adjacent cut sites which subqeuntly give us fragment lenghts. > - Assume cutting goes total completion > - You do not have to follow the skeleton code if you have a different approach to this answering this question

```
In [ ]: #Approach 1
        def fragment_lengths(sequence, cut_site):
            site_len = len(cut_site)
            results = []
            prev_cut = 0
            while True:
                # Use the find method to index the next cutting site
                idx = sequence.find(cut_site, prev_cut)
                # If the cutting site is not found, break the loop
                if idx == -1:
                    break
                # Add the lenght of the cut site to the start index value to get the cut position
                cut_position = idx + site_len
                results.append(cut_position - prev_cut)
                prev_cut = cut_position
            if prev_cut < len(sequence):
                # Add the remaining sequence after the last cut
                results.append(len(sequence) - prev_cut)
            return results
```

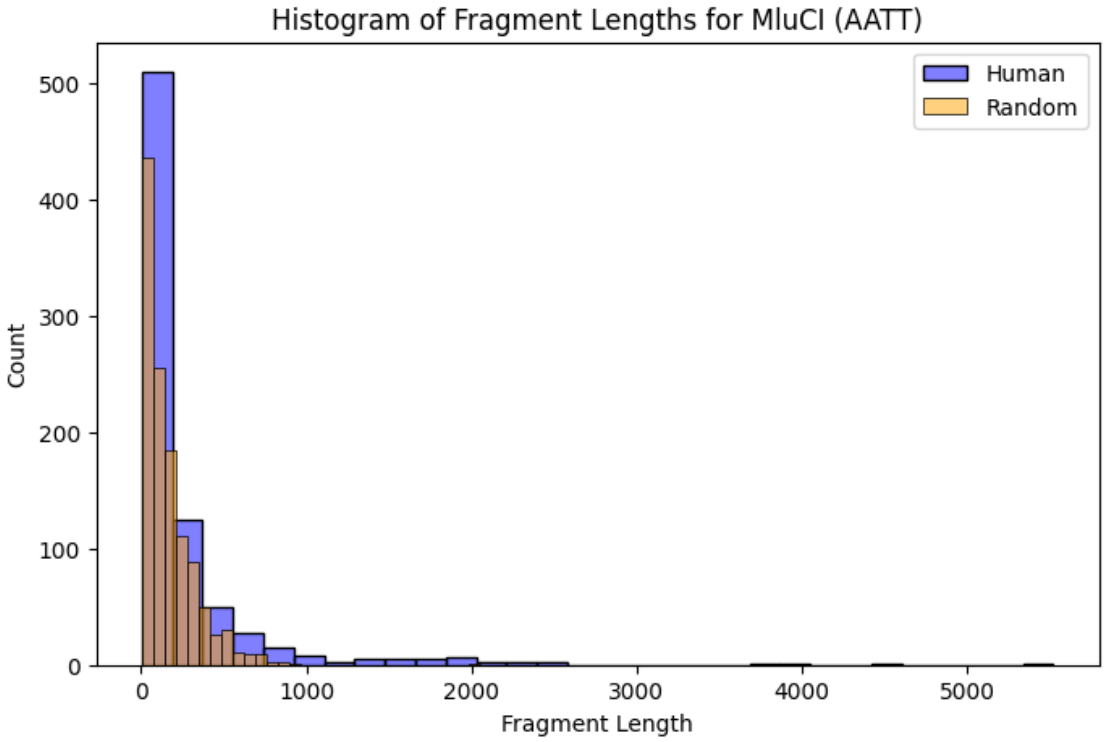
```
In [52]: grader.check("q11")
```

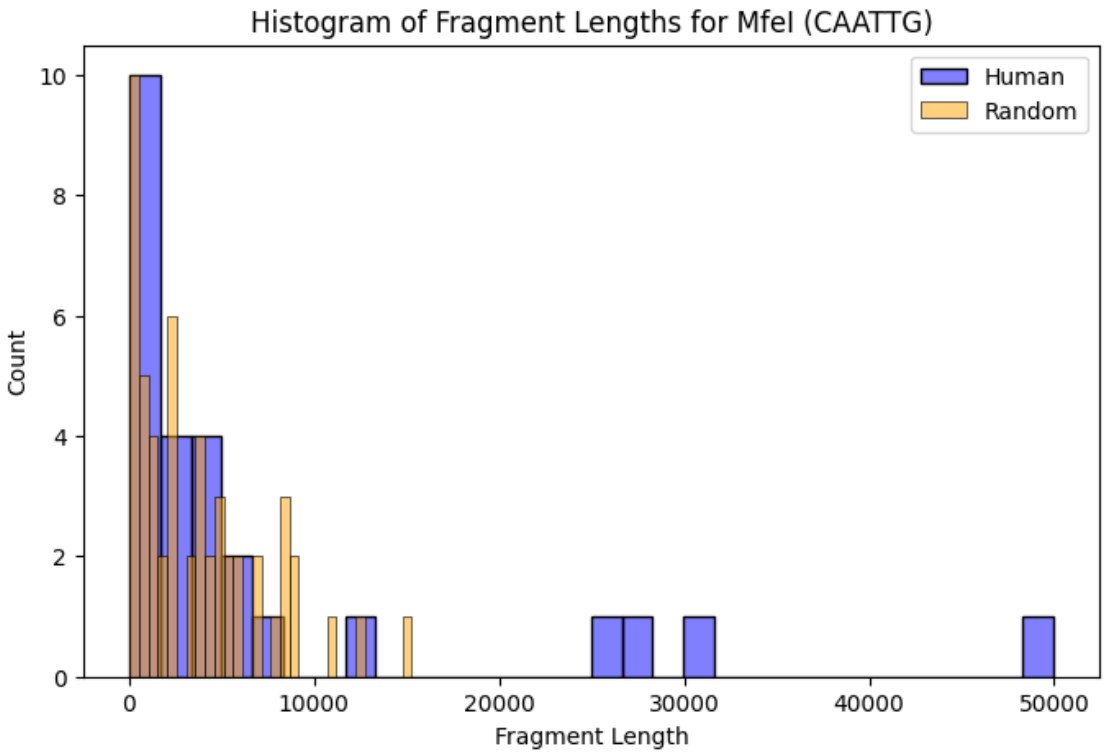
```
Out[52]: q11 results: All test cases passed!
```

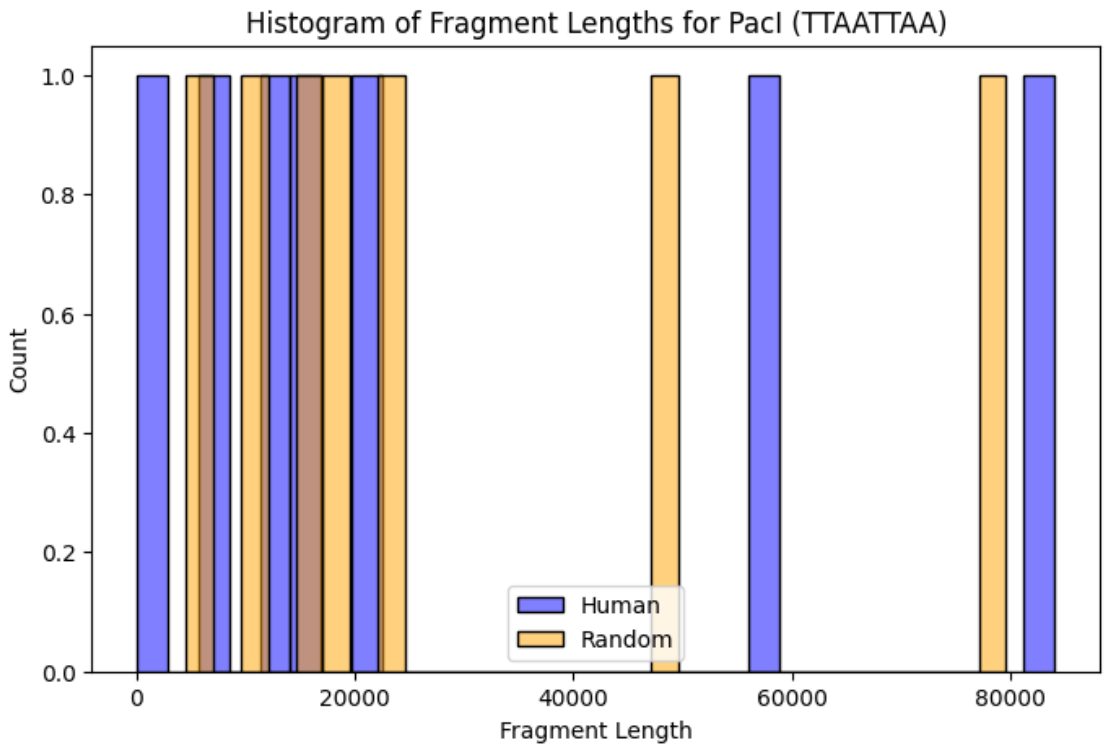
Just run the cell below to generate plots of the distribution of the fragment lengths for each restriction enzyme. It compares the distribution of the fragment lengths for our real human dna and some random DNA with the same GC content. The cell may take up to a minute to run.

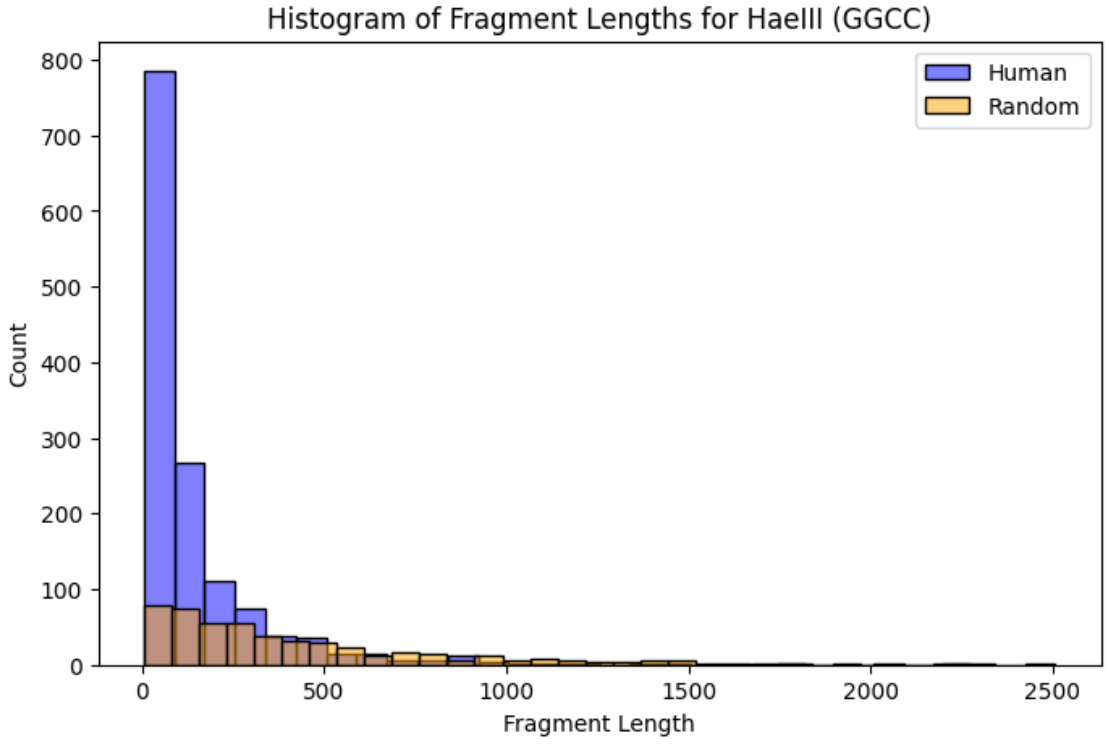
```
In [53]: #Just run this cell
         enzyme_list = [
             ("MluCI", "AATT"),
             ("MfeI", "CAATTG"),
             ("PacI", "TTAATTAA"),
             ("HaeIII", "GGCC"),
             ("EagI-HF®", "CGGCCG"),
             ("NotI", "GCGGCCGC"),
             ("HpyCH4IV", "ACGT"),
             ("ZraI", "GACGTC"),
             ("BssSI-v2", "CACGAG")
         ]

         # Loop through each enzyme and generate a separate plot
         for enzyme_name, cut_seq in enzyme_list:
             fragment_lengths_human = fragment_lengths(human_dna[:200000], cut_seq)
             fragment_lengths_random = fragment_lengths(random_human[:200000], cut_seq)

             plt.figure(figsize=(8, 5))
             sns.histplot(fragment_lengths_human, bins=30, alpha=0.5, label="Human", color="blue")
             sns.histplot(fragment_lengths_random, bins=30, alpha=0.5, label="Random", color="orange")
```

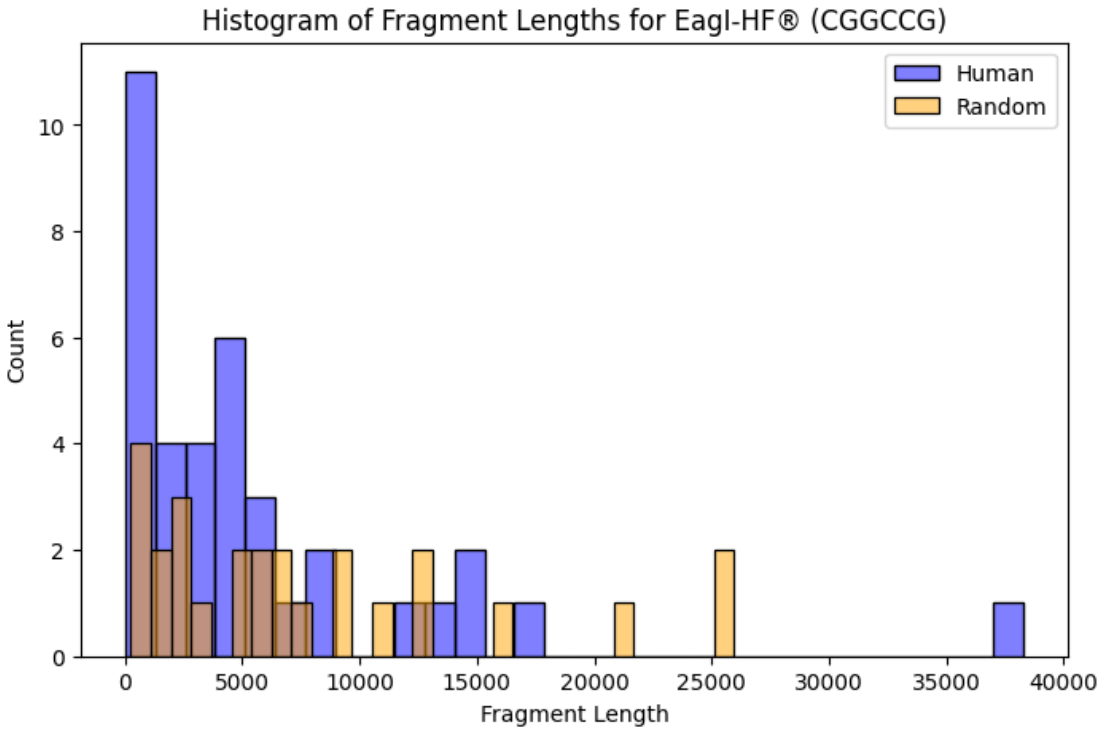
```
plt.xlabel("Fragment Length")
plt.ylabel("Count")
plt.title(f"Histogram of Fragment Lengths for {enzyme_name} ({cut_seq})")
plt.legend()
plt.show()
```

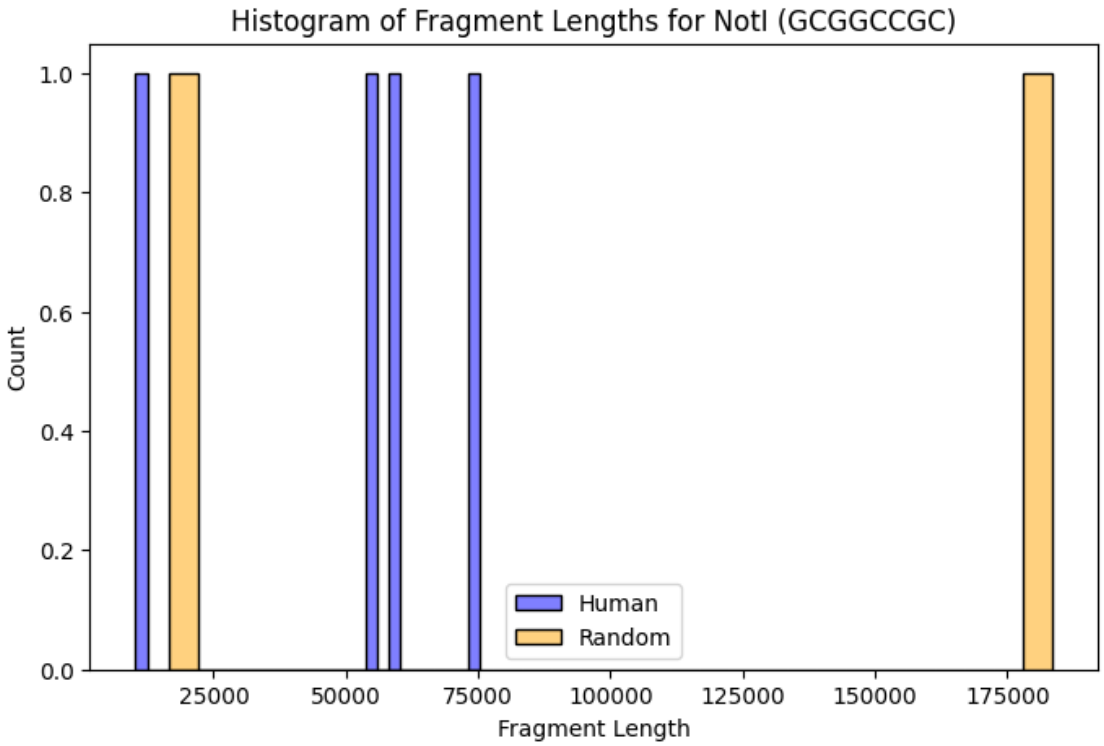


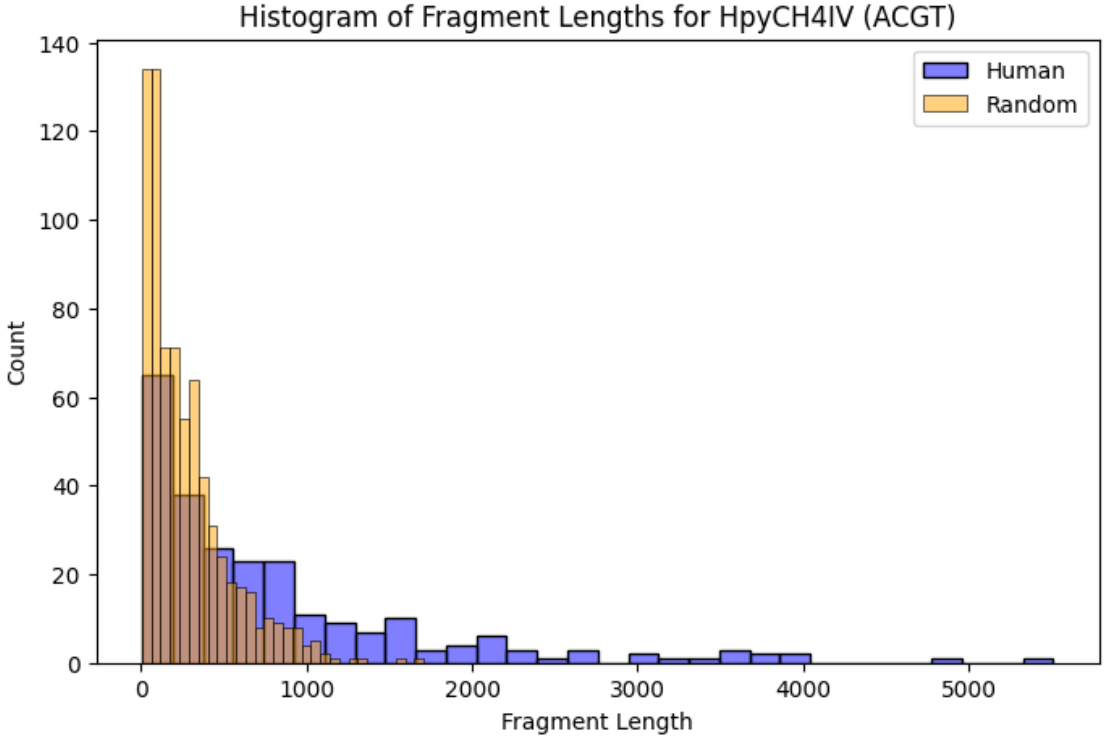

Histogram of Fragment Lengths for MfeI (CAATTG)
Human
Random
Count
Fragment Length
0
2
4
6
8
10
0
10000
20000
30000
40000
50000

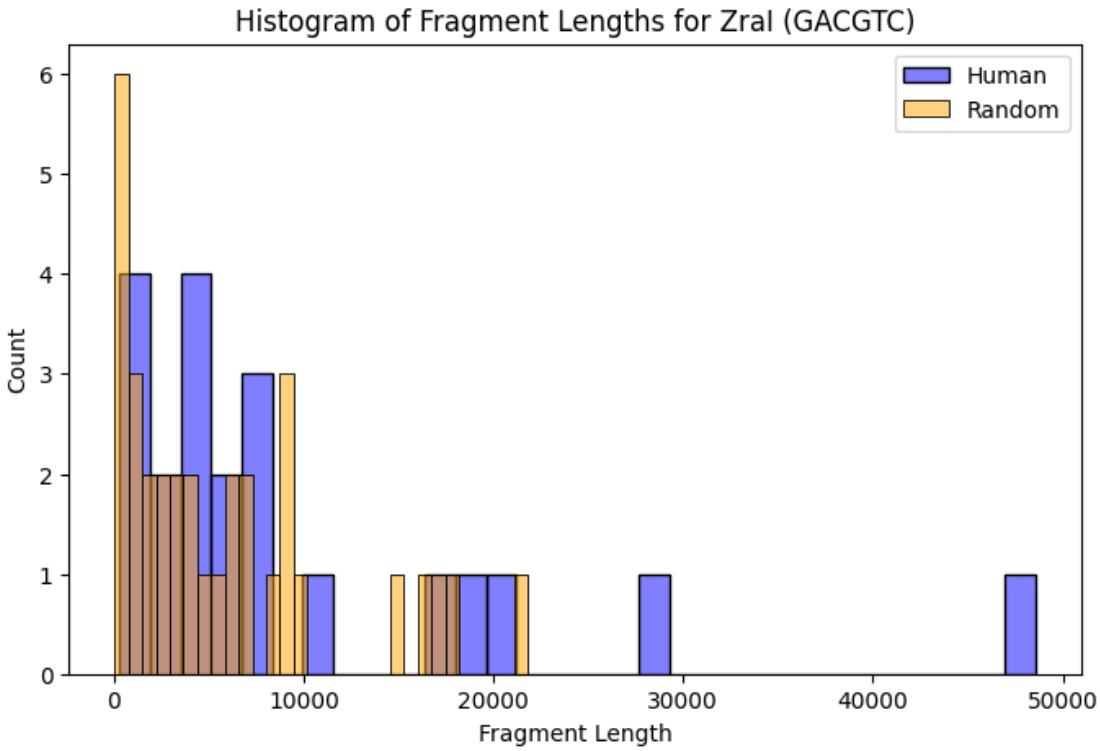

Histogram of Fragment Lengths for PacI (TTAATTAA)
Count
Fragment Length
Human
Random
0.0
0.2
0.4
0.6
0.8
1.0
0
20000
40000
60000
80000

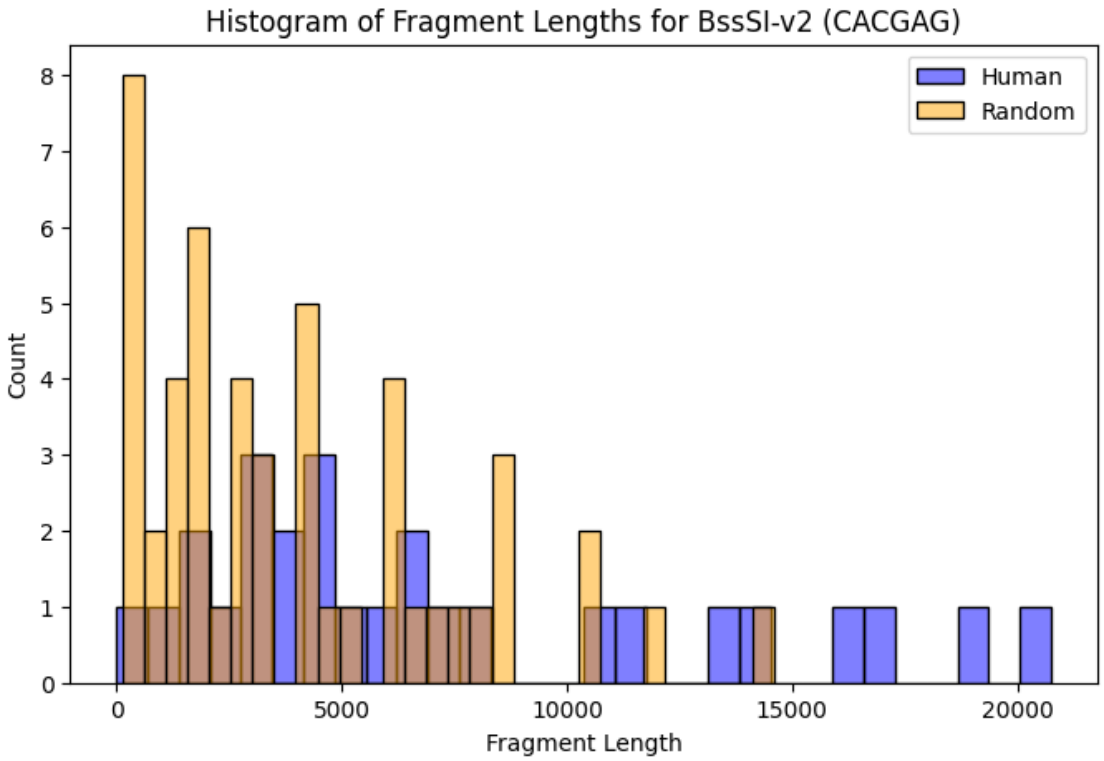

Histogram of Fragment Lengths for HaeIII (GGCC)
Count
Fragment Length
Human
Random
0
100
200
300
400
500
600
700
800
0
500
1000
1500
2000
2500


Histogram of Fragment Lengths for EagI-HF® (CGGCCG)
Human
Random
Count
Fragment Length
0
2
4
6
8
10
0
5000
10000
15000
20000
25000
30000
35000
40000


Histogram of Fragment Lengths for NotI (GCGGCCGC)
Count
Fragment Length
Human
Random
0.0
0.2
0.4
0.6
0.8
1.0
25000
50000
75000
100000
125000
150000
175000


Histogram of Fragment Lengths for HpyCH4IV (ACGT)
Count
Fragment Length
Human
Random
0
20
40
60
80
100
120
140
0
1000
2000
3000
4000
5000


Histogram of Fragment Lengths for ZraI (GACGTC)
Human
Random
Count
Fragment Length
0
1
2
3
4
5
6
0
10000
20000
30000
40000
50000



QUESTION 12: What kinds of different fragment length distributions do you observe in the histograms above? Do any plots show a significantly different distribution from the human versus random DNA? If so, what is a possible biological explanation for those discrepancies?

ANSWER:

MluCI (AATT): Human DNA shows slightly more short fragments than random, suggesting AT-rich regions. This aligns with the abundance of AT-rich repetitive elements (e.g., Alu) in the human genome, leading to more MluCI cut sites.

HaeIII (GGCC): Human DNA shows a stronger peak of short fragments than random, indicating an overrepresentation of GGCC sites. Likely due to GG-rich regions such as CpG islands in promoters.

HpyCH4IV (ACGT): Human DNA shows more long fragments than random, suggesting fewer ACGT cut sites. Likely due to suppression of CG dinucleotides in the human genome (linked to methylation).

MfeI (CAATTG): Human DNA has more long fragments, indicating sparse or clustered MfeI sites. This could reflect regions with conserved functions that avoid disruption from cuts.

EagI-HF® (CGGCCG): More long fragments in human DNA suggest CGGCCG sites are underrepre-

sented. This is likely due to methylation of CpG sequences, which can block recognition.

ZraI (GACGTC): Distributions are similar with some longer fragments in human DNA, suggesting slight underrepresentation of the site. May reflect local sequence bias or methylation.

BssSI-v2 (CACGAG): Slightly longer fragments in human DNA suggest fewer cut sites compared to random. Could be due to GC-bias or avoidance in certain genomic regions.

PacI (TTAATTAA): Both distributions show very long fragments, consistent with the rarity of this long AT-rich recognition sequence. No major difference between human and random DNA.

NotI (GCGGCCGC): Fragment lengths are extremely long, especially in human DNA. Reflects the rarity and methylation of this GC-rich, CpG-dense site in the human genome.

Cas9 Targeting

CRISPR-Cas9, a revolutionary technology for editing genes, operates in a highly targeted manner. During the Cas9 targeting process, the guide RNA and Cas9 protein complex moves along the DNA strand searching for a binding sequence.

As the CRISPR-Cas9 complex moves along the DNA, it looks for the presence of a Protospacer Adjacent Motif (PAM) sequence. This PAM sequence is crucial because it signals the correct location for gene editing. The Cas9 enzyme from Streptococcus pyogenes, for example, recognizes the PAM sequence 'NGG' (where 'N' can be any nucleotide followed by two guanines).

Once the PAM sequence is identified, the Cas9 protein induces a localized unwinding of the DNA helix. This allows the gRNA to bind or anneal to its complementary DNA sequence adjacent to the PAM. The binding is through standard base-pairing rules (A pairs with T, C pairs with G).

After the gRNA successfully pairs with the target DNA sequence next to the PAM, the Cas9 protein activates its nuclease domains. The gRNA base pairing is 20 bp for actual of 23 match base pairs. These domains then cleave the DNA, typically cutting both strands, leading to a double-strand break at a precise location in the genome, allowing for gene editing to occur.

You could explore either of these questions using `human_chr21`,`yeast_chr7` and/or `e_coli_genome`.

QUESTION 13: These series of questions explore the statistical specificity of gRNA targeting based on sequence length, GC content, and PAM constraints, assessing their impact on target site availability and the precision of CRISPR-Cas9 in different genomes. These questions are open ended, but please show your work to prove your answers

QUESTION 13a: What is the stastically expected minimum lenght of a gRNA target sequences needed in order to bind to only one site in 10^6 bp for each species, considering their GC content? How does this compare to the true length (20+3)?

Answer: So we want the expected number of matches in the genome segment of size 10^6 for each species to be 1, therefore our match probability will be $\frac{1}{10^6}$.

If we assume that the bases are equally likely within each pair we get that the probability of:

- $P(\text{A or T}) = \frac{p_{AT}}{2}$ where p_{AT} is the AT proportion
- $P(\text{G or C}) = \frac{p_{GC}}{2}$ where p_{GC} is the GC proportion

So we want to solve for Expected minimum length of gRNA target sequences needed, and let's call this quantity L. Thus we can form the following relation:

$$(\frac{p_{AT}}{2})^{L \cdot p_{AT}} \cdot (\frac{p_{GC}}{2})^{L \cdot p_{GC}} = \frac{1}{10^6}$$

and simply solve for L which can be done by taking ln to both sides of the equation giving us that the expexted min length of a gRNA target L is given by:

$$L = \frac{\ln 10^{-6}}{p_{AT} \cdot \ln \frac{p_{AT}}{2} + p_{GC} \cdot \ln(\frac{p_{GC}}{2})}$$

and we calculate pythonically as follows below

```
In [57]: def get_min_length(bp, gc_content):
             numerator = -np.log(bp)
             at_content = 1 - gc_content
             denominator = at_content * np.log(0.5 * at_content) + gc_content * np.log(0.5 * gc_content)
             return numerator / denominator

         base_pairs = 1e6

         human_min_length = get_min_length(base_pairs, gc_content_human)
         yeast_min_length = get_min_length(base_pairs, gc_df.loc["Yeast"]["GC content"])
         e_coli_min_length = get_min_length(base_pairs, gc_df.loc["E. coli"]["GC content"])

         print(f"Computed minimum length for human DNA: {human_min_length:.2f} bp")
         print(f"Computed minimum length for yeast DNA: {yeast_min_length:.2f} bp")
         print(f"Computed minimum length for E. coli DNA: {e_coli_min_length:.2f} bp")
```

```
Computed minimum length for human DNA: 10.00 bp
Computed minimum length for yeast DNA: 10.18 bp
Computed minimum length for E. coli DNA: 9.97 bp
```

So what this illustrates that while the theoretical minimum length for uniqueness in a small region is around 10 bp, the true CRISPR system employs a much longer sequence to achieve precise targeting in real biological contexts.

QUESTION 13b:How much does the NGG constraint matter for the number of target sites in the 10^6 bp for each species?

Answer: So without the PAM constraints, every possible region in our 10^6 bp region could be a target site. But as you'll see in the work below, if we do have this NGG constraint, it narrows the pool of potential target sites dramatically which ensures that only a subset of sites is eligible for editing

```
In [59]: #use this code cell to show your work
         human_gc = gc_df.loc["Human"]["GC content"]
         yeast_gc = gc_df.loc["Yeast"]["GC content"]
         ecoli_gc = gc_df.loc["E. coli"]["GC content"]

         bp = 1e6

         # now i calculate expected number of valid PAM sites
         human_sites = bp * (human_gc/2)**2
         yeast_sites = bp * (yeast_gc/2)**2
         ecoli_sites = bp * (ecoli_gc/2)**2

         print(f"Expected number of valid PAM sites in human DNA: {human_sites:.2f}")
         print(f"Expected number of valid PAM sites in yeast DNA: {yeast_sites:.2f}")
         print(f"Expected number of valid PAM sites in E. coli DNA: {ecoli_sites:.2f}")
```

```
Expected number of valid PAM sites in human DNA: 50176.00
Expected number of valid PAM sites in yeast DNA: 36100.00
Expected number of valid PAM sites in E. coli DNA: 65792.25
```

So we see how instead of having 1,000,000 positions as target site when we ignore PAM requirements, we see that the above narrows it down a lot!

QUESTION 13c:How many cut sites to you expect to occur by chance in each whole genome? (Hint genome sizes are: Ecoli: 4.6e6 bp, Yeast: 12.2e6 bp, Human 3.2e9 bp). You can assume GC content from above. Is this 'good enough' for precision genome engineering of human cells?

Answer: The expected number of cut sites in the whole genome can be found as such:

$$E(\text{matches}) = N \times P_{\text{match}}$$

And $P_{\text{match}} = P_{\text{target}} \times P_{PAM}$.

$$P_{\text{target}} = (\frac{P_{AT}}{2})^{20 \cdot P_{AT}} \times (\frac{P_{GC}}{2})^{20 \cdot P_{GC}}$$

```
In [ ]: #use this code cell to show your work
```

0.2 Extra Credit

Most assignments in this course will include optional extra credit questions. These questions are designed as starting points for students to explore more free-form mini projects. Therefore, there is no skeleton code and minimal guidance for these questions. Students are welcome to go beyond the scope of the question or adapt the question as necessary to answer their own scientific questions of interest. You are welcome to create as many coding cells as you would like for these mini-projects. In order to get extra credit, students should make a reasonable attempt (as judged by the grader) on at least one question and write a brief report.

Write a summary on your methodology and your findings, highlighting key results and any interesting observations. The length of the report does not matter, as long as it answers all of the following questions: - What was your scientific goal with this project? - What methods did you use and why? - What were the key results you found for each method you implemented? - Were there any limitations in your methods? - What additional observations or comments can you make on your findings? What is the greater biological relevance or implication? - Are there any additional questions you would want to explore?

EC Mini-Project A: There is a major patent dispute between UC Berkeley and the Broad Institute. Learn about this dispute and write 500 essay summarizing the current state of the litigation.

EC Mini-Project B: Eric Lander, the director of the Broad Institute, wrote a high profile review article chronicling the early history of CRISPR during the patent litigation. How might this article have influenced the patent litigation? Do you see any discrepancies between this article and the history published by the Nobel committee? Do you notice any biases? Do you feel any potential conflicts of interest were adequately disclosed? https://www.cell.com/fulltext/S0092-8674%2815%2901705-5 vs https://www.nobelprize.org/uploads/2020/10/popular-chemistryprize2020.pdf

EC Mini-Project C: While Cas9 based CRISPR is currently the most popular tool for modifying genomes, other technologies existed before CRISPR, like TALENs and Zinc finger nucleases. Research one of these alternative methods for genome engineering and write a short paragraph comparing and contrasting that method with CRISPR. What are the advantages of each method? What are the disadvantages? This question is intended as an open ended exploration. You may want to investigate gene therapy dimensions and AAV delivery systems, or explore immunogenicity, describe genome target specificity, or describe ease of reagent validation. Please limit your answer to 500 words discussing one or a few contrasts. After exploring the strengths and weaknesses of these different technologies with the members of your group, what is your favorite method? Why?

EC Mini-Project D: How many unique cut sites can you target in each of your 10^6 bp regions?

```
In [ ]: #YOUR CODE HERE. ADD ADDITIONAL CODE CELLS AS NEEDED.
```

EXTRA CREDIT REPORT: [insert project choice here]

DOUBLE-CLICK TO EDIT THIS CELL AND TYPE YOUR REPORT

0.3 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

Save, then download the zip file and submit it to the Gradescope autograder assignment. Then, extract the written solutions PDF and upload to the written work lab assignment.

In [ ]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export(run_tests=True)
```