***Disclaimer: This is how I like to do things, I hope you don't mind. I don't delete anything, so most of the things I type here are nonsense and scrambling through thoughts + learning.***

Amphibian Dataset
Dataset is small enough that I can do a manual visual scan per each variable, to check for which ones are interesting.

*Added a conditional formatting on presence of amphibians*

- *1: Light green*
- *0: Light red*

*Easier to visually scan*

Observations sorting from largest to small and scanning amphibian columns.

**Overall Trend:**

- **First 3 are "common", and appear with many different types of indicators.**
- **The last 4 have much more variance in their presence based.**
- **Considering clustering the dataset into first 3 labels and last 4 labels.**

**Reservoir Size in $m^2$ combined (Numerical):**

- **Very large reservoirs seem to have almost guaranteed presence of the first 3, more likely to have last 4.**
- **Very small reservoirs typically didn't have last 3, but some had all 7.**
- **Straightforward, or potentially bimodal? Also potentially little impact on the small side, but large size may help indicate presence.**

**Number of Reservoirs (Numerical):**

- **Same general trend as Reservoir Size as expected.**
- **Don't know if this or that is a better indicator. Their comment is that the more reservoirs there are, the more likely one of them will be suitable for breeding in comparison to 1 reservoir habitats.**

**Type of Water Reservoir (Categorical):**

- **This one will require a visual aid of some sort****
- **But there could be potentially interesting data here, they categorized it into 15 different types of water reservoirs. Looks like a good indicator.**

**Presence of Vegetation (Categorical)**

- **This one will require a visual aid of some sort****
- **Doesn't seem to be much of a pattern, potentially bimodal though.**

**SUR1, SUR2, SUR3 Surrounding Land Types (Categorical)**

- **Something funky is going on with this one. The data is showing up to 14, but the data description only gives categorical descriptors up to 9.**
- **Nonetheless, needs a visual aid for this sort of categorical data. From a brief scan, some of the SUR1 look a bit barren, SUR2(1) looks very barren on the last 4 amphibians (doesn't necessarily mean anything), nothing on SUR3**

**Use of Water Reservoirs (Categorical)**

- **Description said unused water reservoirs are very attractive for amphibians, but I don't really see it unless I'm misreading the label.**
- **They only have data over 3 of their 4 categories, but there is a wide variance of amphibian presence within each category.**

**Presence of Fishing (Categorical)**

- **Again, their data is confusing. They have descriptors for 3 categories but give data labels in 0-4. Ugh.**
- **Inconclusive. Doesn't seem to be a pattern. Will need to model.****

**Percentage of Reservoir Edges touching undeveloped areas (Numerical)**

- **This looks like a very promising indicator** (Definitely want to model this one)**
- **High untouched edge % is a good indicator of the last 4 labels.**

**Distance to nearest road in meters (Ordinal/"Categorical")**

- **A bit of a weird data point, but their argument is that the farther away, the amphibians feel safer.**
- **Doesn't seem to be a pattern**

**Distance to nearest building in meters (Ordinal/"Categorical")**

- **Same ordinal scale as distance to nearest road**
- **No pattern once again, but for Great crested Newt (Label 7), there doesn't seem to have presence at both extrema (0 and 10). Interesting.**

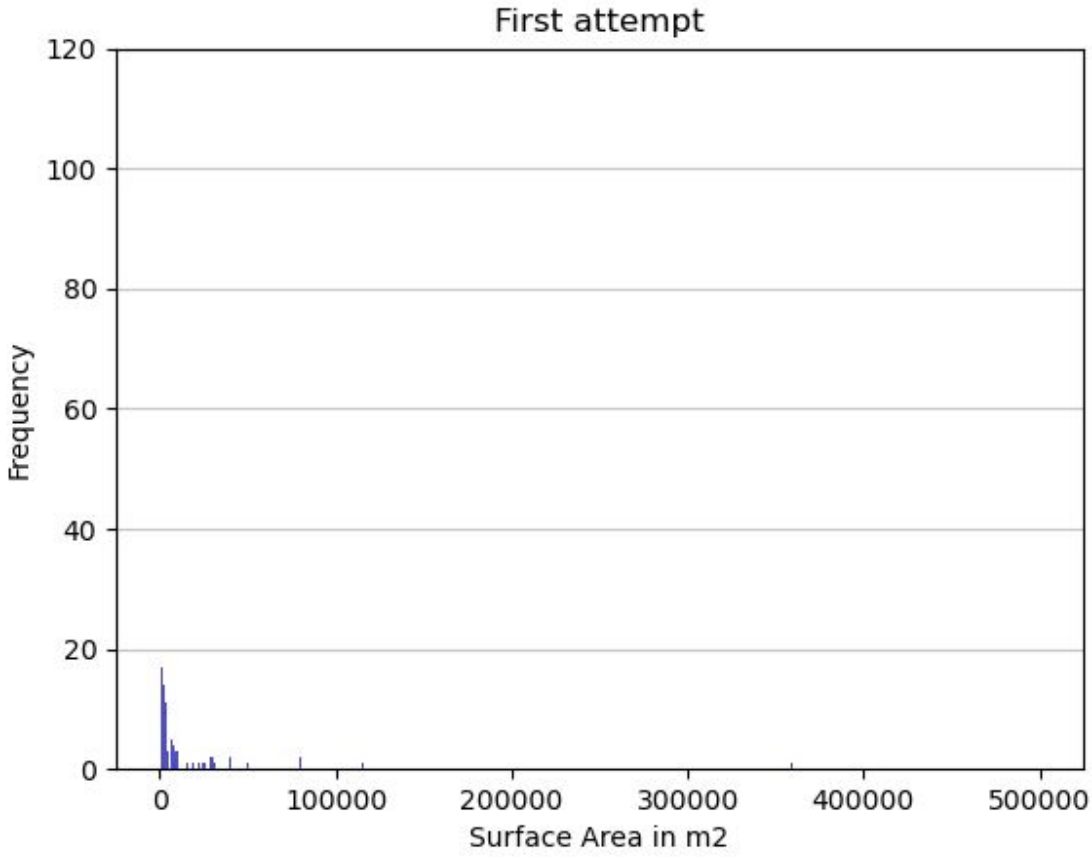
**Maintenance Status of reservoir (Categorical)**

- **From clean, to slightly littered, to trashed.**
- **A fantastic indicator of life in general**
- **Unfortunately (or rather fortunately?), every reservoir is clean except 1 slightly littered, and 3 trashed.**

**After sifting through the variables, I've decided to cluster the data into 2 groups. Labels 1 through 3 are henceforth the common amphibians, and Labels 4 through 7 are the "rare" ones.**

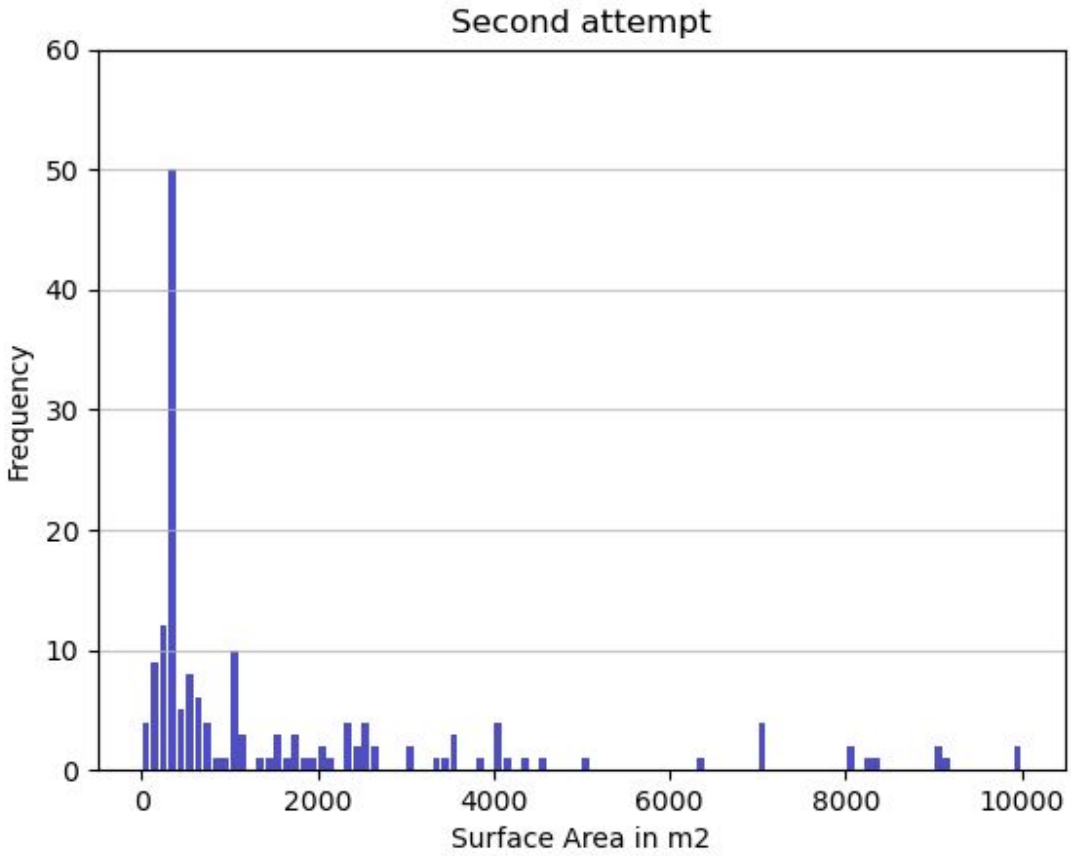
**The decision is justified by the frequency at which the latter 4 appear in comparison to the first 3.**

**Let's start with reservoir size and histogram it**




**That is awful to look at it.**

**Upon further inspection, 20 of these reservoirs range from 15000m$^2$ to 500000m$^2$, while the other 170 range from 0 to 10000. A significant portion of the 170 are 300m$^2$. Let's change the bin width to 100 and go from 0 to 10000. We'll consider this 170 size subset for now and deal with the outliers later.**

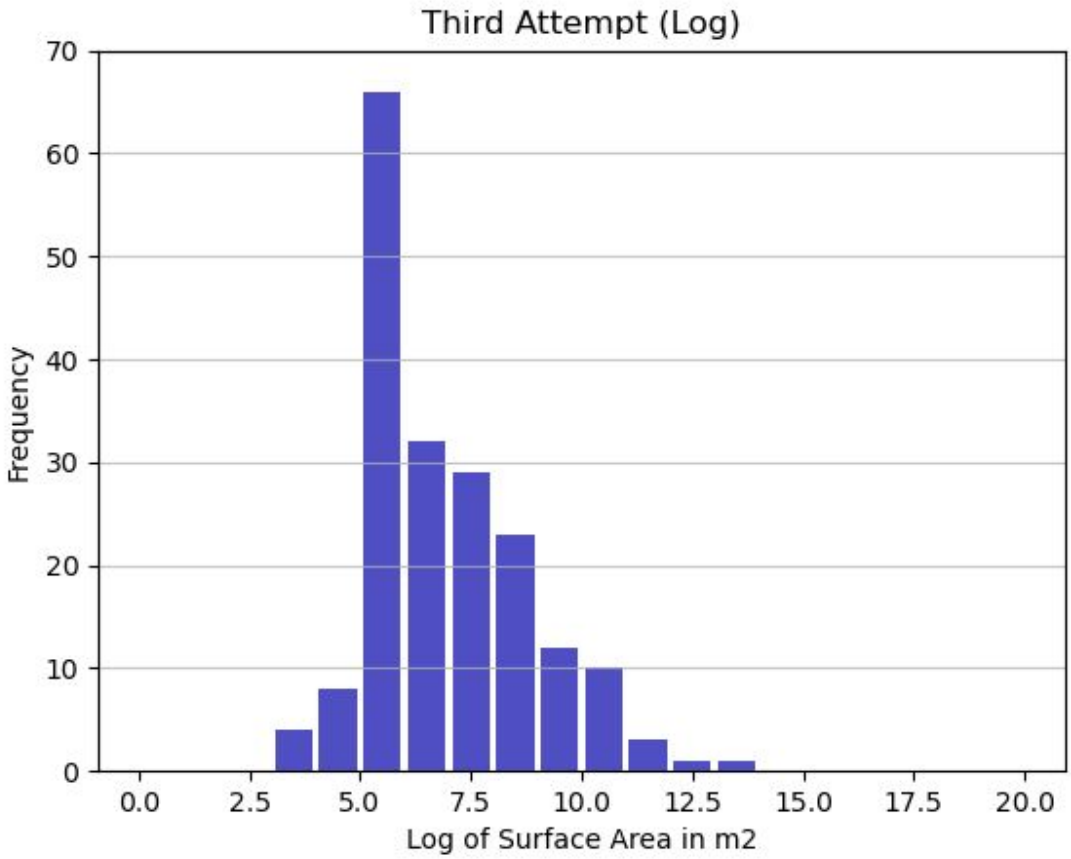



**Better. We're looking at a right skewed histogram. Apparently in 2012, 48% of reservoirs were manmade and 52% were natural. The manmade ones tend to be smaller in size. That being said, I'm incredibly naive right now and just want to normalize everything. So we're going to attempt to normalize this data so we can employ Gaussian models and a Gaussian mixture model, backtracking when finished to look for improvement.**

**The first thing I wanted to try was a log transformation. The skew coefficients dropped dramatically**

```
8.846455010444526
0.8694231652457425
```

**And our new histogram looks a lot nicer too.**




**This is still… very right-skewed. Needs more work. I also transformed the data instead of looking for a distribution to fit the previous data.**

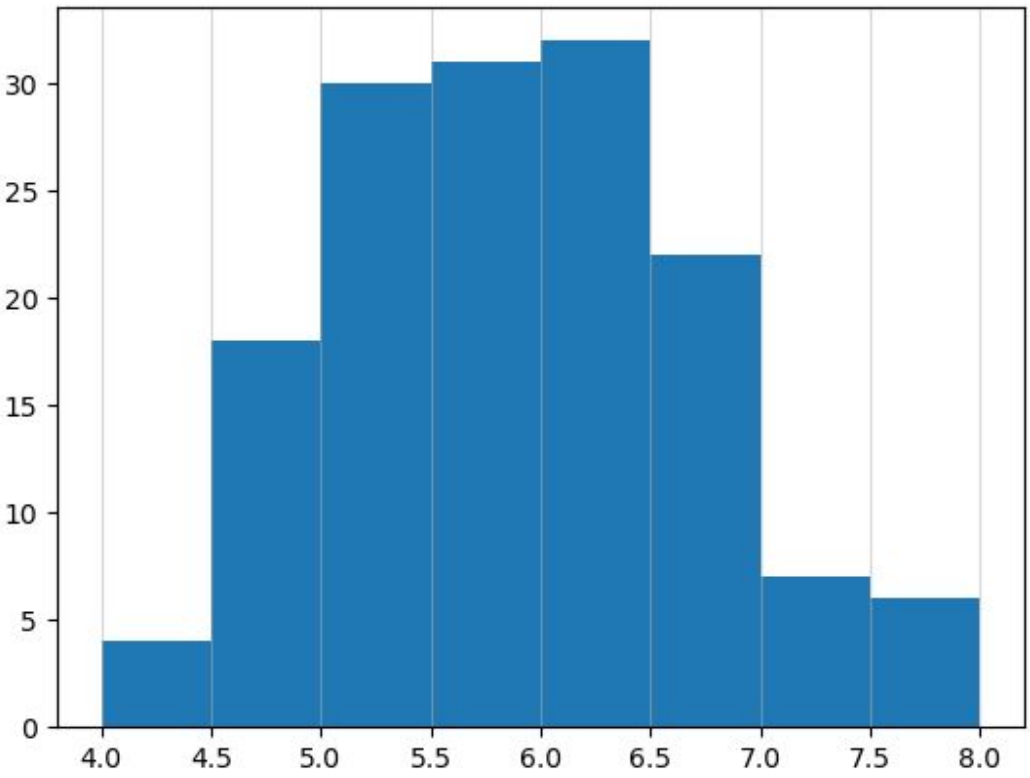
---

**Changing my approach with a slightly better understanding of distributions, so let's start over. First, use a simpler dataset, like Iris. No ordinal data, no multivariate confusion, we'll keep it simple. The challenge for Iris will be a Gaussian mixture model (which I guess is technically multivariate.) We'll come back to Amphibians in a bit.**

**New plan: Check out the distribution of each variable, one at a time. Maybe employ a Gaussian mixture model (if applicable) for at least one of them.**

**Let's start with a Sepal Length (cm) histogram**

**Input looks good, a quick scan of the dataset shows values range from 4.3 to 7.9. Default bin number is usually 10, so let's try 4 to 8 in 0.5 intervals, [4, 4.5, 5. 5.5, 6, 6.5, 7, 7.5, 8]**




**Starting easy, let's just try to fit a Gaussian to this thing. Here's the idea. Since we have the discrete dataset, let's calculate the sample mean and standard deviation to get our probability density function.**
**We'll use the parameters to generate a few hundred samples from a Normal Distribution in Python, run it a few times, and check how well it compares. I'll overlay the generated Gaussian on our histogram here.**

**Mean: Σ sample values / # = 5.843**

**STDev:** $$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

**(I will type out formulas I copy paste to show understanding. Summation of difference between sample and mean squared (to make values absolute) for every sample, averaged by dividing by N, then reversing the squared with a sqrt)**

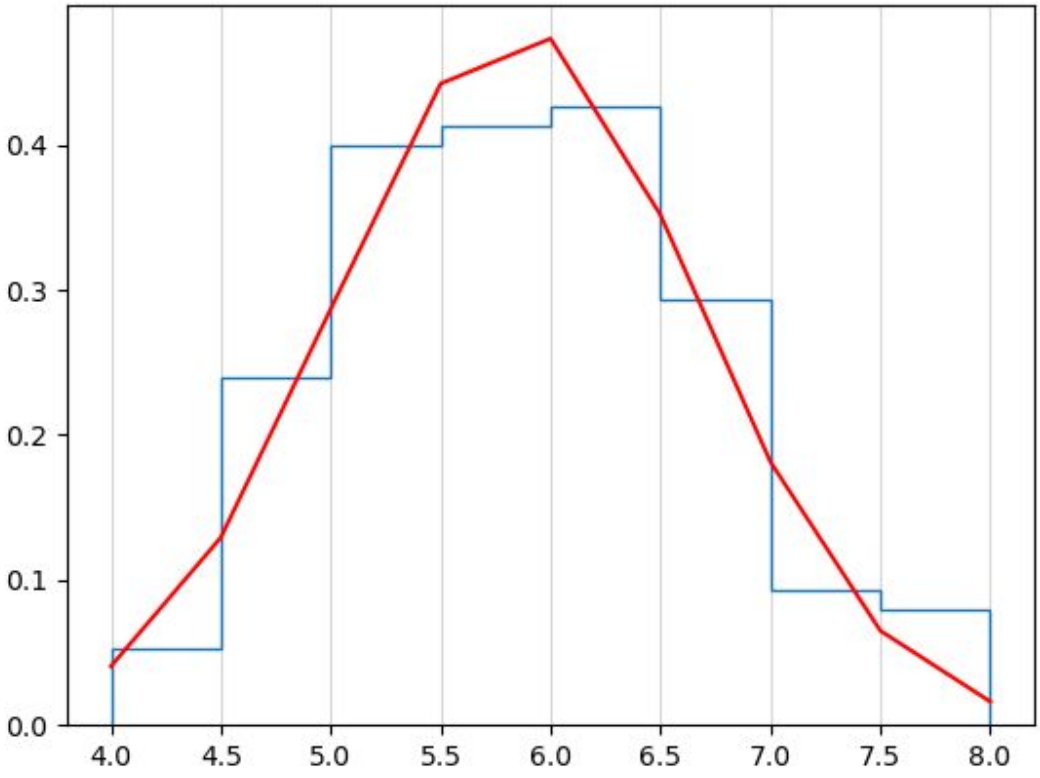
**Note this is the population STDev formula, and since we are using a sample to try and generalize a population, we'll use the N-1 sample STDev instead.**

**So STDev (S): ~0.828**

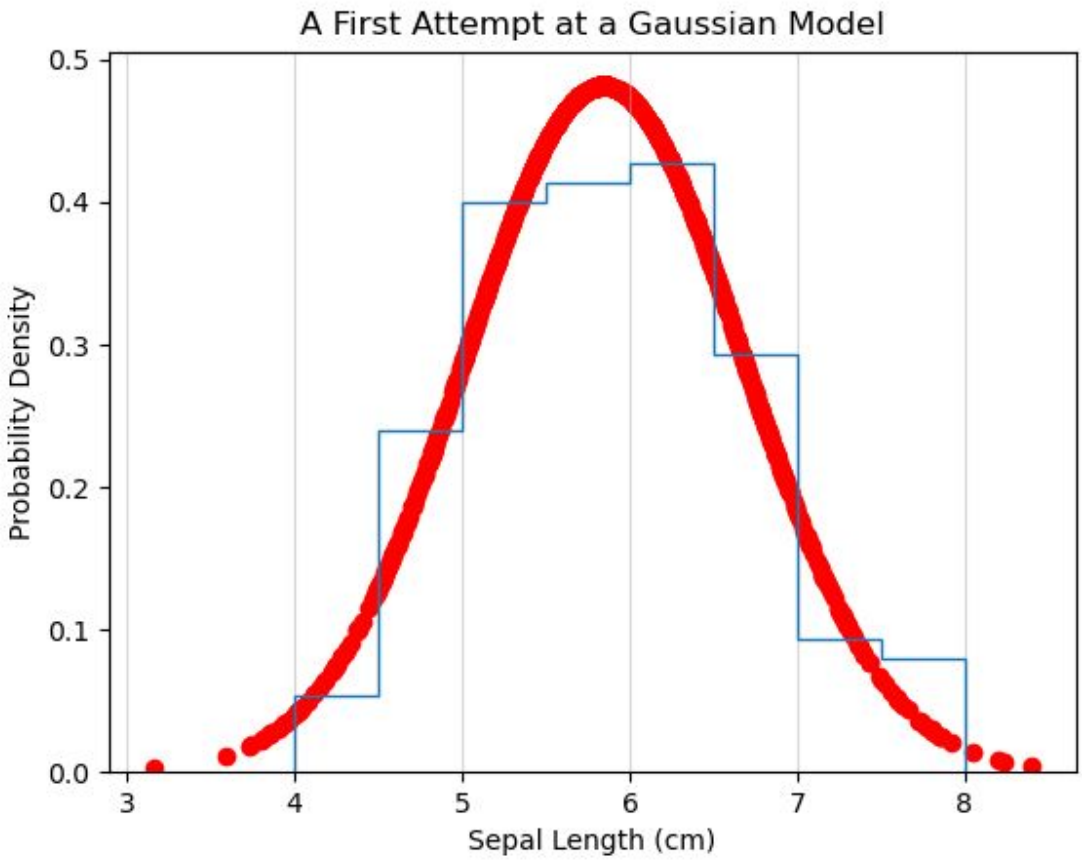
**Armed with (5.843, 0.828), we'll use NumPy.random.normal now, let's do 500 samples. I can verify its working by getting close to a 0 difference for my new/old sample mean/stDEVs. Nice.**

**The probability density function (y-values) for a Normal Distribution is given by:**

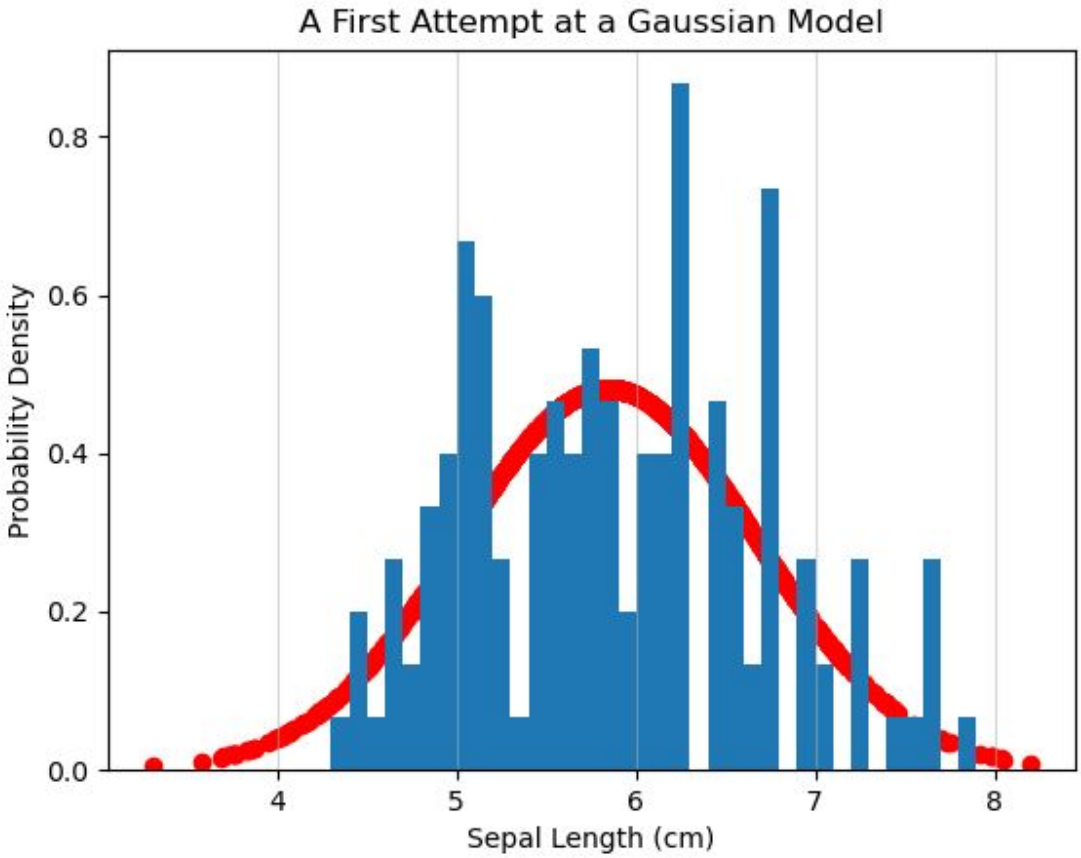
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



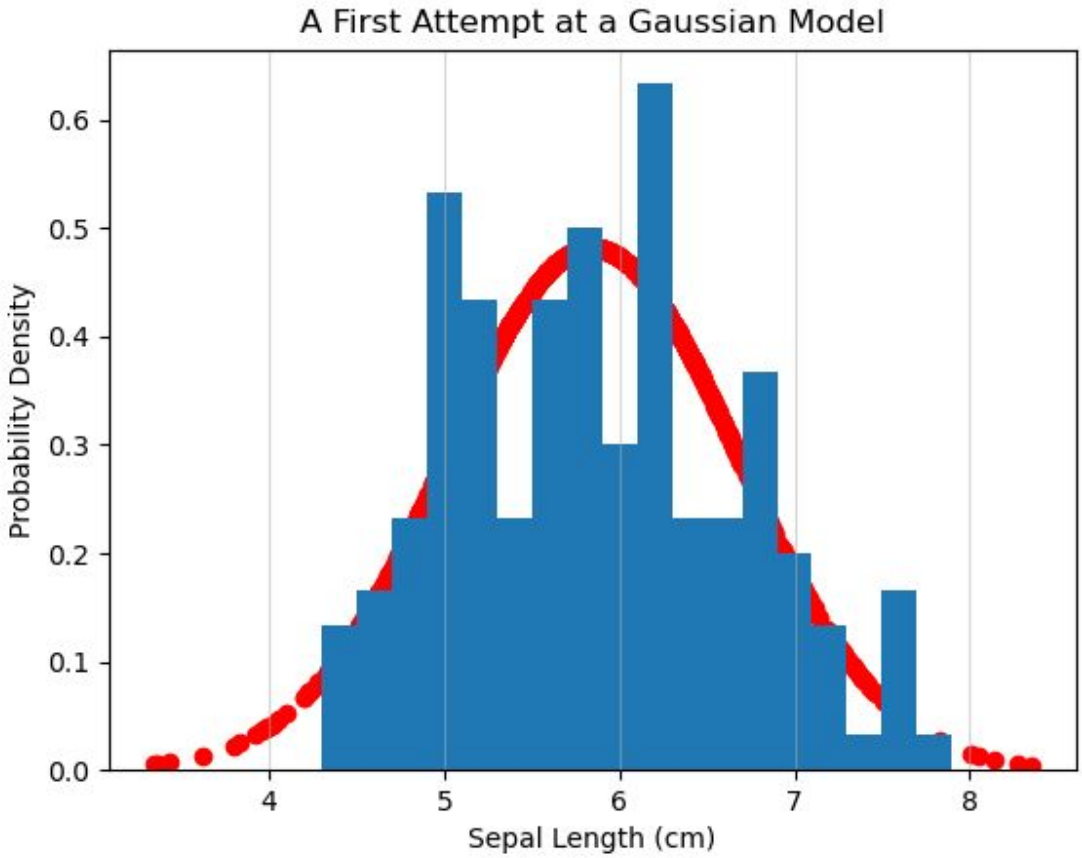
**The blue step histogram is our sample data, the red curve is an attempt to draw a curve without scikit, using the pairs of bin edges and their probability density function values. We can make this better(?) by using every generated sample point and running a scatter plot instead.**




**Here's the scatter version. Simple and reasonable. Before moving onto the next variable, Let's change the bin size for our original sample to 0.1, set the range from our min to max, and take a look. Okay, that's stupid, there's no bin-size constraint. Well if we want go from 4.3 to 7.9 in 0.1's, we'll just make 36 bins.**




**Whoa… It is quite interesting to note since going in such small increments, our distribution choice could be potentially much different. This definitely could be a case of overfitting, but it's possible our original assumption for a Gaussian model was some sort of underfitting. Let's try cutting this bin width in half.**




**Interestingly, this is still not really Gaussian. If anything, this could be a prime opportunity to attempt a mixture of Gaussian models. I see potentially a mixture of 2, one at mean ~6 and the other at mean ~5. Cutting the number of bins in half again gets us back to our original idea, so let's not do that.**

**I was going to move to the next variable, but a Gaussian doesn't seem quite to do justice here. We will attempt a Gaussian mixture on this histogram then.**

## The attempt at a Gaussian Mixture Model (GMM) begins here.

**The prime characteristic of a Gaussian mixture is being able to represent subpopulations within our population without having to explicitly identify which subpopulation a sample falls within. Unlike k-means, where the clustering is hard, we can cluster probabilistically ("soft").**

**I'm very unsure of how to do this, but I am aware of both EM and Markov Chain Monte Carlo. We will see this through with the EM method, and k = 2.**

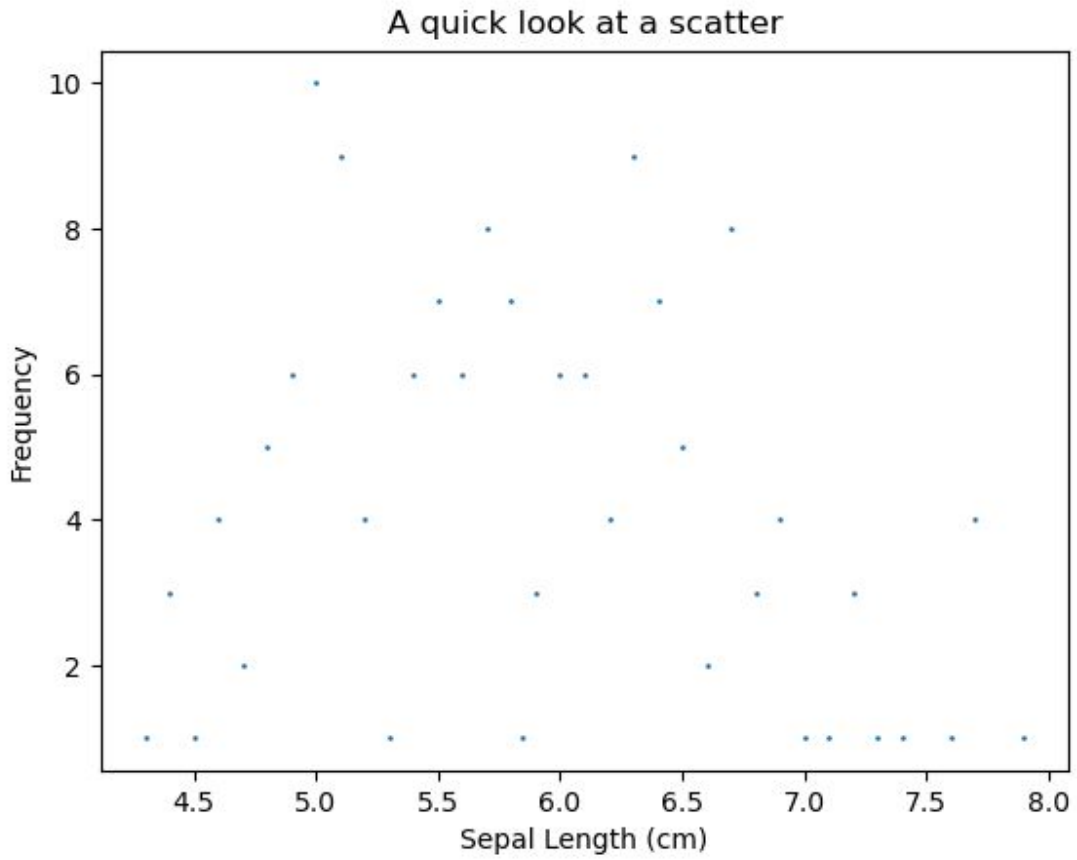
**A wonderful article for GMMs**

https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95

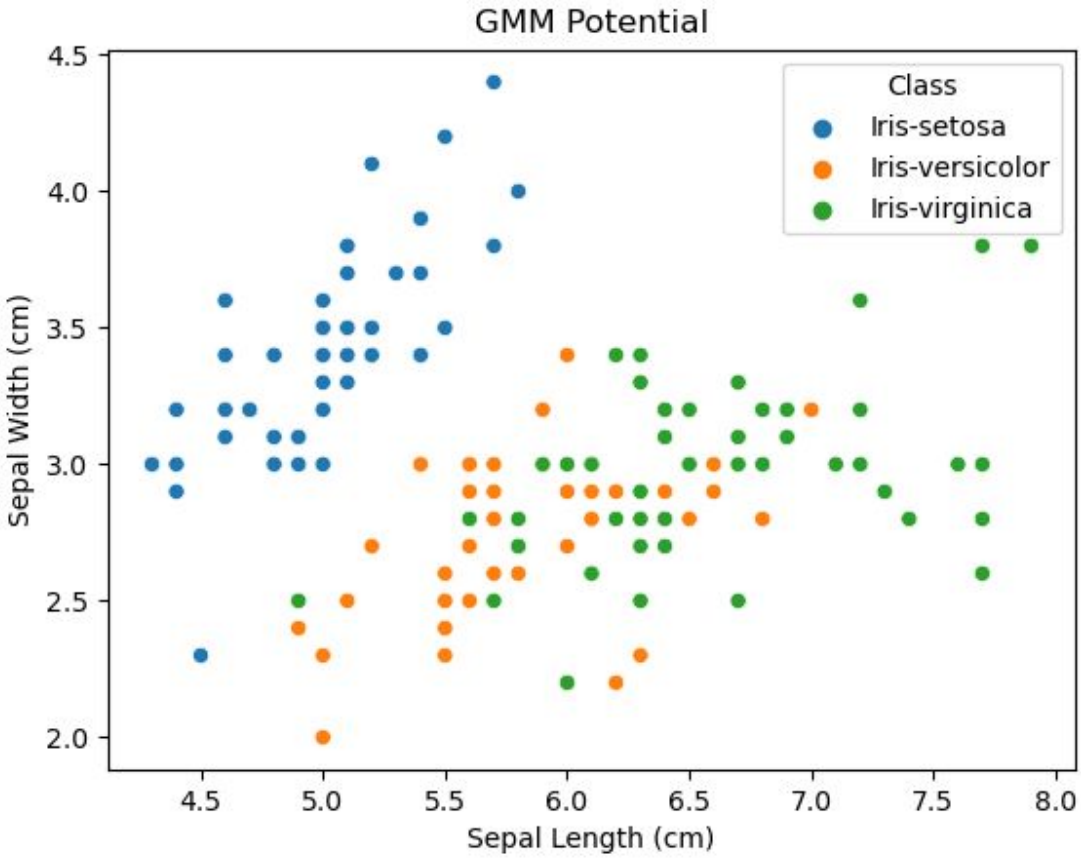
**As explained there, we will have 3 parameters for each Gaussian distribution. 2 of which are familiar, mean and variance, and a 3rd one specific to mixture models, a scaling parameter/probability weight that must sum to 1 across all distributions. A typical rule is to initialize this weight 1/k, so we will, to ½. This weight will eventually converge, a neat property of GMM, as further iterations will not cause any sort of overtraining.**

**For the expectation step, we generate likelihoods (MLE more precisely) for samples and specific clusters. The parameters behave as our prior beliefs, and we retune them in the maximization step, then repeat until convergence, or maximum iterations specified.**

**Since we're only working with 1D data, no covariance matrices are needed, but since I'm looking to scikit-learn's implementation for inspiration, we will attempt it.**




**I just thought this was a cool scatter. Originally, I tried running my GMM on this and was very confused by the non-clustering, but realized frequency is not the way to go. Since I've already built the multivariate GMM class, why not use it? Let's try Sepal Length vs Sepal Width.**



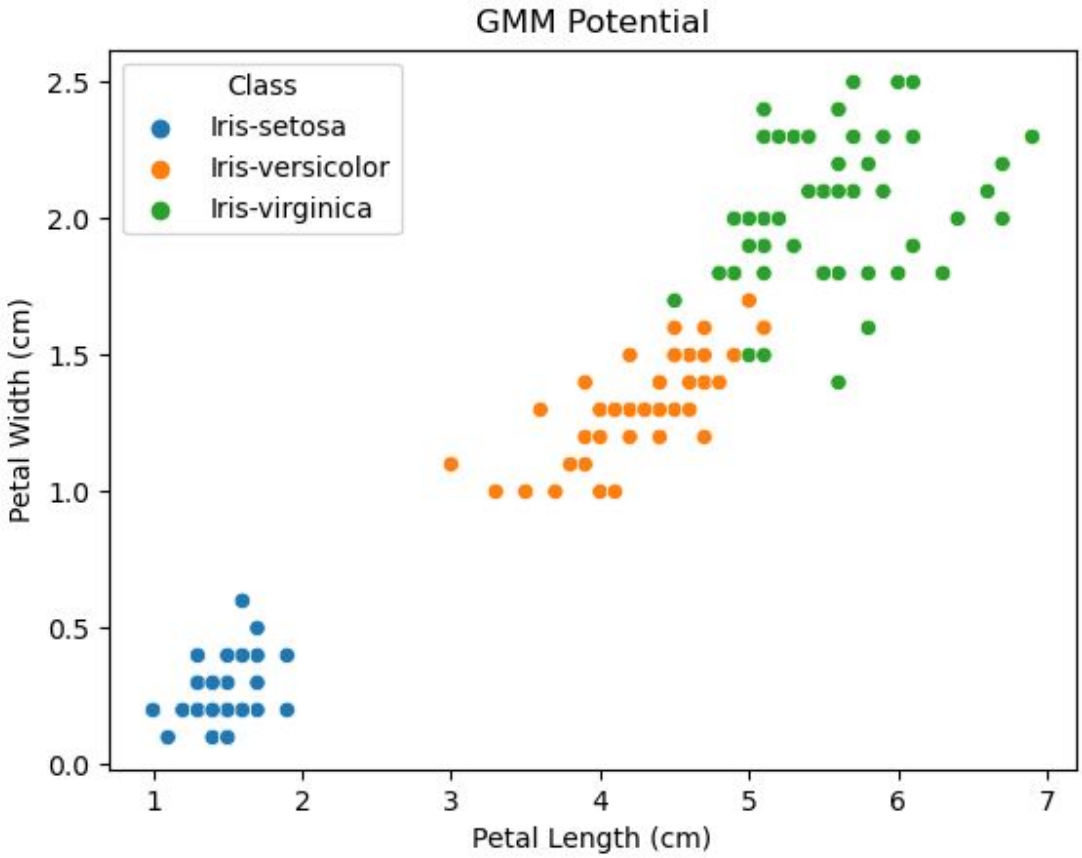

**This looks really promising! In the context of our problem, there are 3 types of irises, and I can see how one might see k=3 clusters here, and why kmeans would be less beneficial than GMM, because in our interpretation, the middle two clusters have quite a bit of overlap.**

---

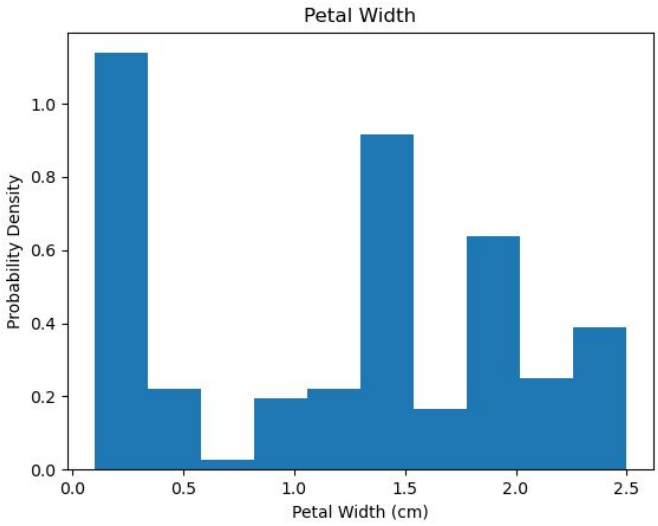
**Unfortunately, I have spent all the time I can afford on trying to get this blasted implementation of GMM to work. I'm sure I will get it eventually, but for now, it's reducing my dataset to k points and creating singularity clusters. I can't figure out the bugs, and I need to move on to the rest of the dataset. I will come back for it.**

```python
class GMM:
    def __init__(self, k, max_iter): #initialization of GMM class
        self.k = k
        self.max_iter = int(max_iter)

    def initialvalues(self, X): #defining values before EM
        self.dimensions = X.shape
        self.row, self.col = self.dimensions
        self.mixingcoeff = np.full(shape=self.k, fill_value = 1/self.k) #using standard 1/k initalization
        self.weights = np.full(shape=self.dimensions, fill_value=1/self.k) #same here
        #Need to split our dataset
        new_X = np.array_split(X, self.k)
        self.mu = [np.mean(x, axis=0) for x in new_X]
        self.sigma = [np.cov(X.T) for x in new_X]

    def e(self, X):
        self.weights = self.bayes(X)
        self.mixingcoeff = self.weights.mean(axis = 0)

    def bayes(self, X):
        likelihood = np.zeros((self.row, self.k)) #contains probabilities for every cluster in its row
        for i in range (self.k):
            distribution = multivariate_normal(
                mean = self.mu[i],
                cov = self.sigma[i],
                allow_singular=True) # actually, this should be removed but I can't figure out the bug
            likelihood[:,i] = distribution.pdf(X)

        numerator = likelihood * self.mixingcoeff
        denominator = numerator.sum(axis=1)[:, np.newaxis]
        weights = numerator / denominator
        return weights

    def m(self, X):
        for i in range (self.k):
            weight = self.weights[:, [i]]
            total_weight = weight.sum()
            self.mu[i] = (X * weight).sum(axis=0) / total_weight
            self.sigma[i] = np.cov(X.T, aweights = (weight/total_weight).flatten(), bias = True)

    def fit(self , X):
        self.initialvalues(X)

        for iteration in range(self.max_iter):
            self.e(X)
            self.m(X)

    def predict(self, X):
        weights = self.bayes(X)
        return np.argmax(weights, axis = 1)
```

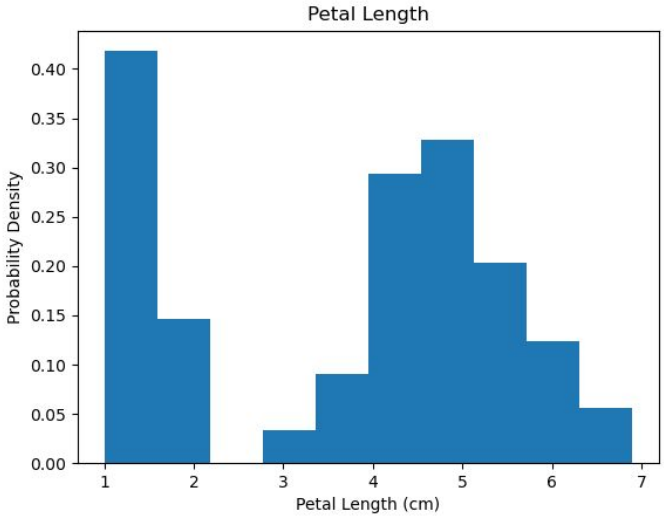
**Let's make a similar graphic above for Petal Length (cm) vs Petal Width (cm), as well as showcase their histograms.**

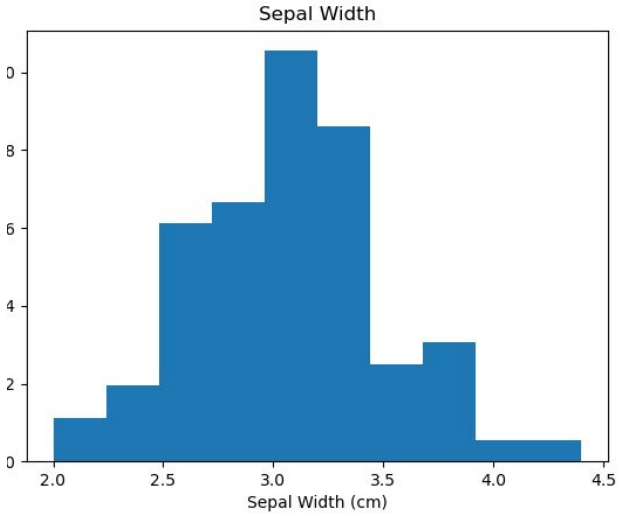



**This one doesn't even need GMM, k-means will do just fine on how distinctly clustered these are. (although the dataset is small enough that a performance issue wouldn't really matter here).**




**10 bins. We can see that multiple near-independent Gaussians idea hinting at good kmeans usage here.**


Petal Length
Probability Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
6
7
Petal Length (cm)


Sepal Width
2.0
2.5
3.0
3.5
4.0
4.5
Sepal Width (cm)