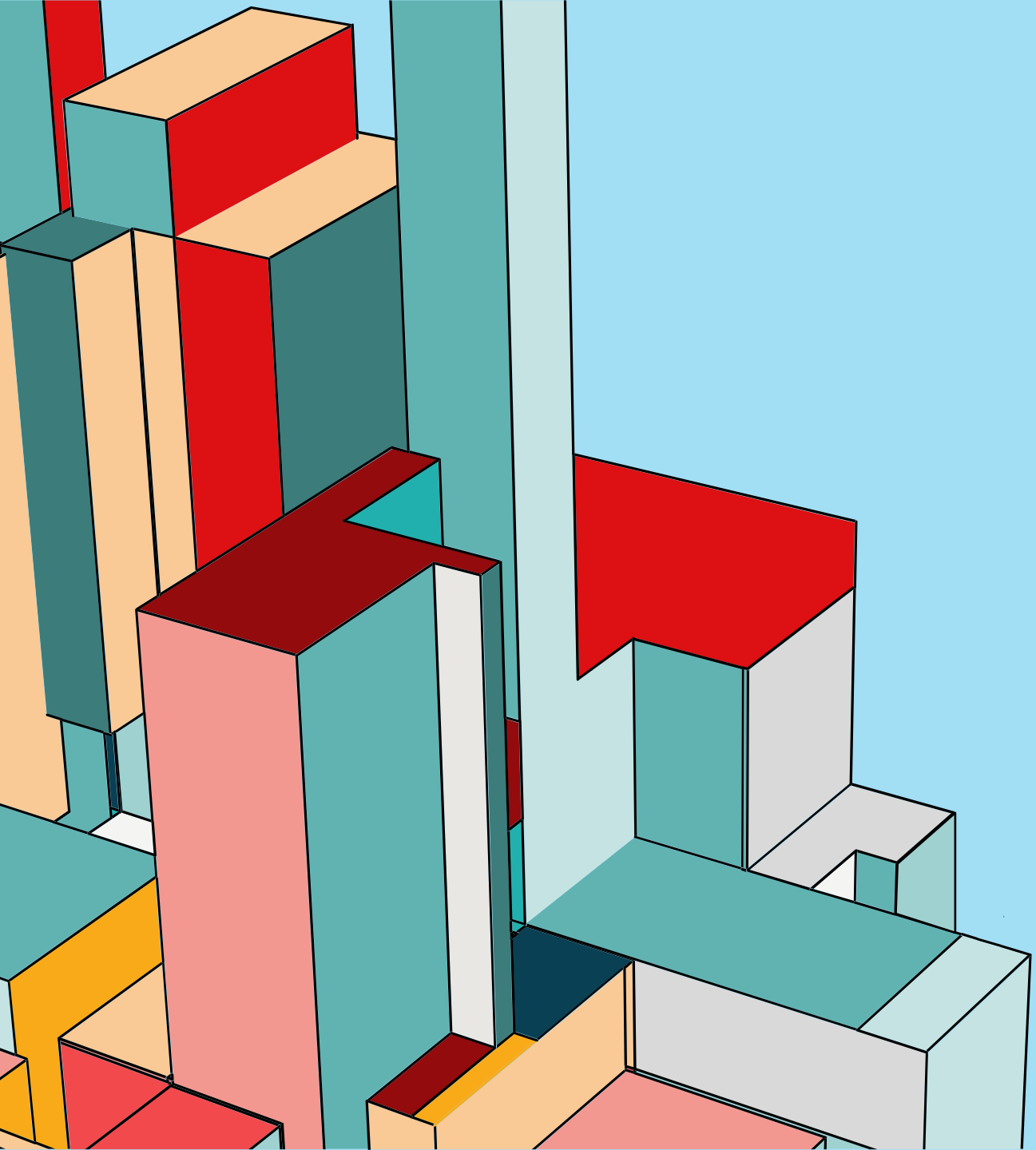
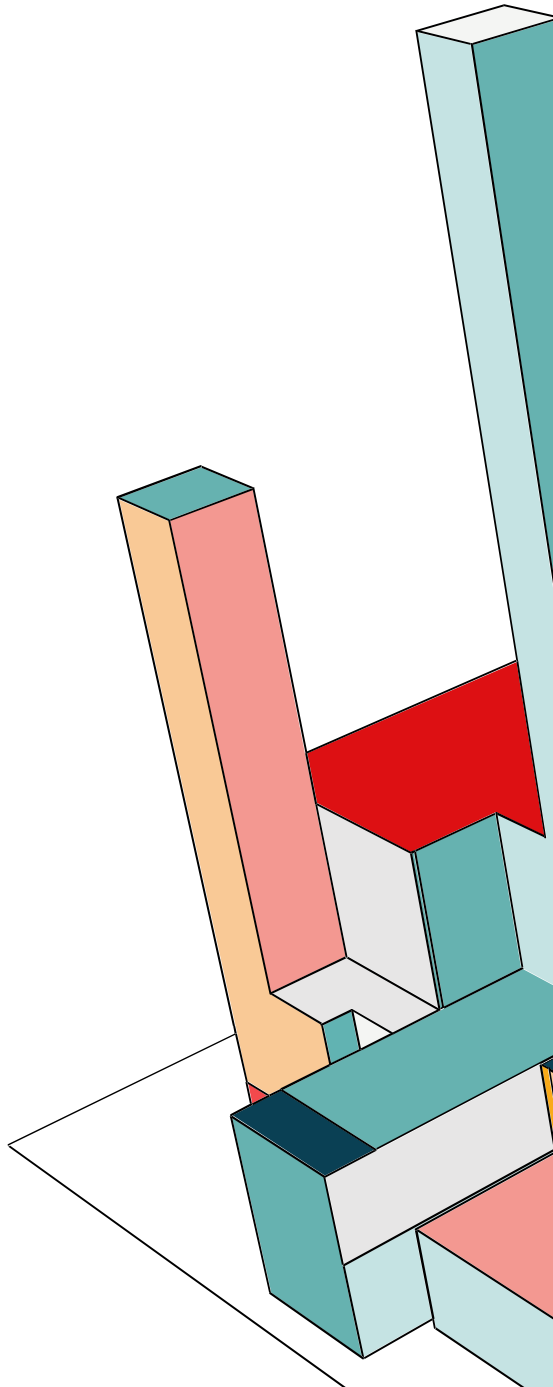


BAYESIAN PARTS

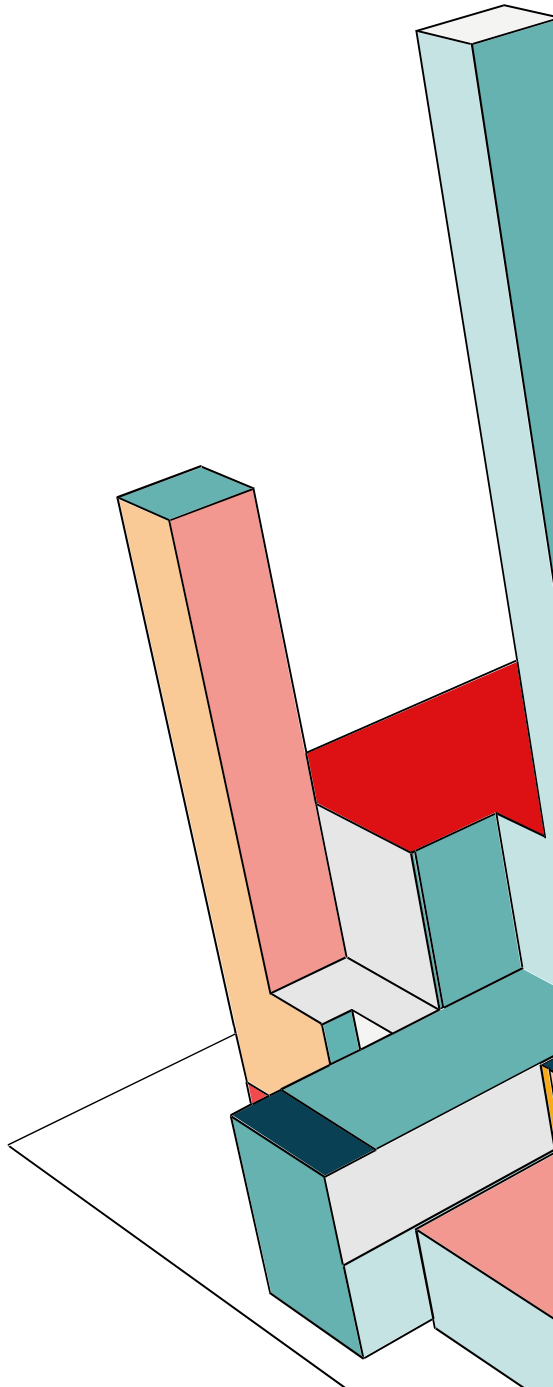
THE PRIOR

- We can see from Bayes' theorem that the prior is a probability: P(θ). First, let's dig into what ‘θ’ means. θ is often expressed as our hypothesis for the model that best describes the variable we are trying to investigate. Let's go back to the example of heights. We infer, from background knowledge and common sense, that height is normally distributed in a class. Formally:
- $h \sim N(\mu,\sigma)$
- Where N denotes the normal distribution, μ denotes the mean and σ denotes the standard deviation.



THE PRIOR

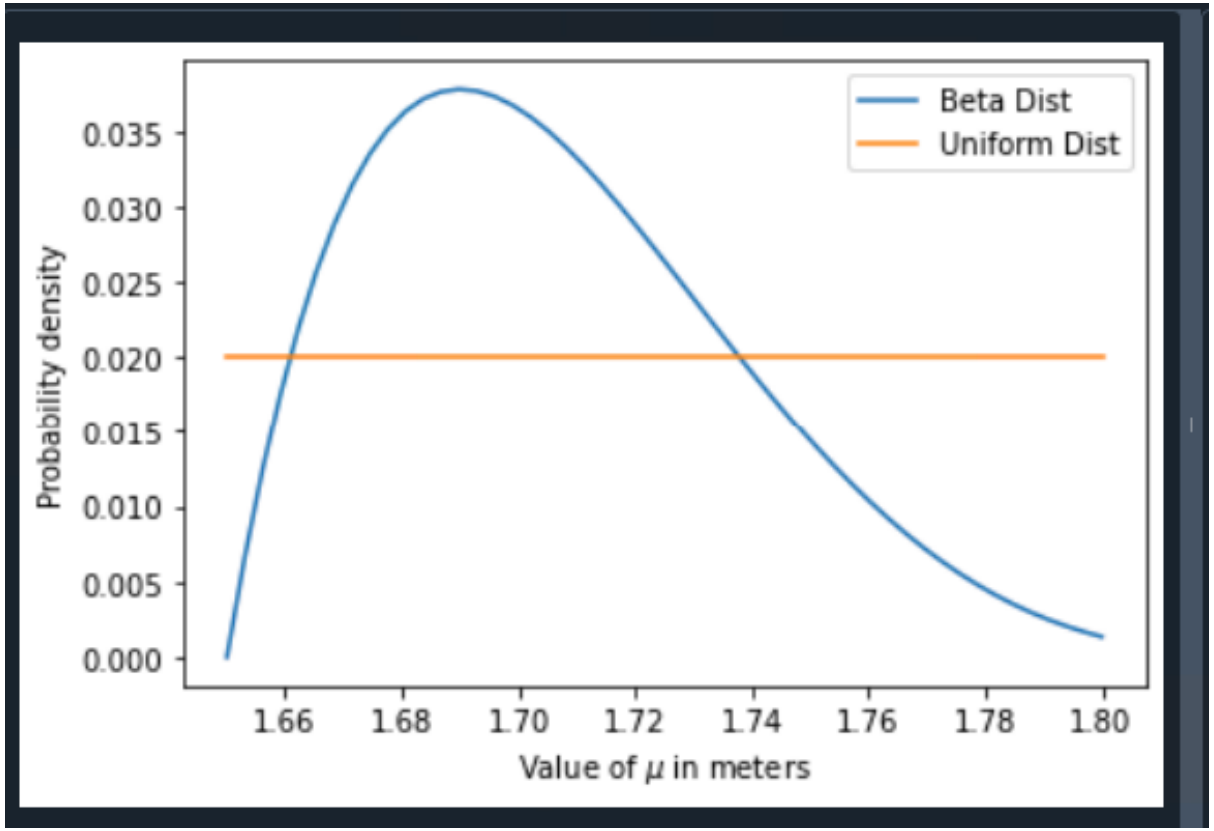
- Now, our prior isn't exactly the expression above. Instead, it is our assumptions for how each of the parameters, μ, and σ, is distributed. Note that this is where the defining characteristic of Bayesian statistics shows up: how do we find the distribution of these parameters? Well, amusingly, we "make them up" based on our prior knowledge. If we have very little prior knowledge, we can choose a very uninformative prior so as not to bias the process at all. For example, we can define that μ, the mean height, is somewhere between 1.65m and 1.8m. If we want to go for an uninformative prior, we can say that μ is uniformly distributed along that interval. If instead, we believe that the mean height is somewhat skewed to a value closer to 1.65m than to 1.8m, we can define that μ is distributed with beta distribution, defined by "hyper" parameters α and β. We can take a look at these options below:



SAMPLE SCRIPT/S

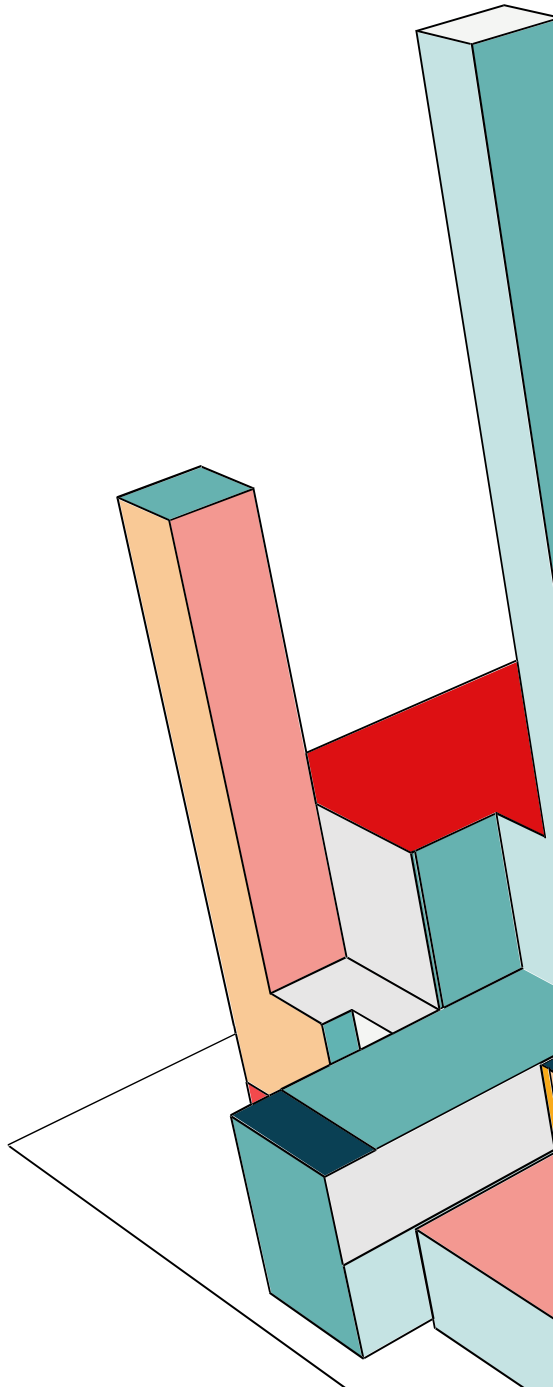
```
"""
Created on Sat Mar  9 10:21:10 2024

@author: jcprado
"""

import scipy.stats as sts
import numpy as np
import matplotlib.pyplot as plt

mu = np.linspace(1.65, 1.8, num = 50)
test = np.linspace(0, 2)
uniform_dist = sts.uniform.pdf(mu) + 1 #sneaky advanced note: I'm using the uniform distribution for clarity,
                                       #but we can also make the beta distribution look completely flat by tweaking alpha and beta!
uniform_dist = uniform_dist/uniform_dist.sum() #Normalizing the distribution to make the probability densities sum into 1
beta_dist = sts.beta.pdf(mu, 2, 5, loc = 1.65, scale = 0.2)
beta_dist = beta_dist/beta_dist.sum()
plt.plot(mu, beta_dist, label = 'Beta Dist')
plt.plot(mu, uniform_dist, label = 'Uniform Dist')
plt.xlabel("Value of $\mu$ in meters")
plt.ylabel("Probability density")
plt.legend()
```

SAMPLE SCRIPT/S



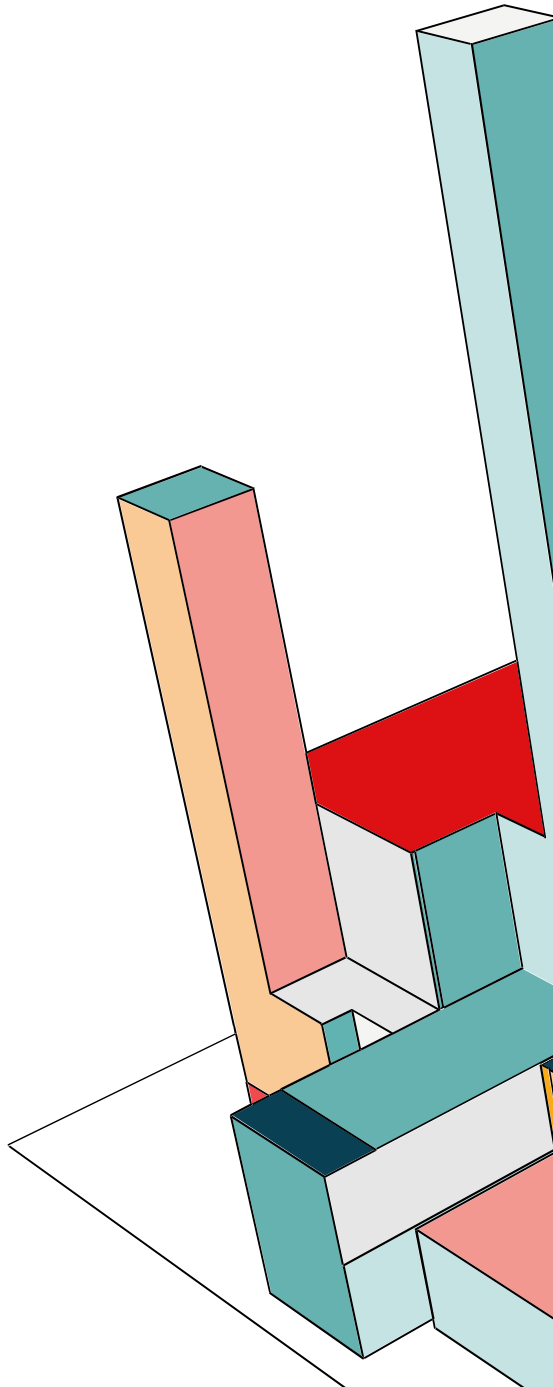
THE LIKELIHOOD

- The likelihood is expressed as P(Data|θ). The ‘data’ in this case would be an observed value for the height. Say we get to measure one student, picked at random, and their height is 1.7m. Consider that with this datum we can now have a sense of how good each option for θ is. We do this by asking: if a particular option for θ, call it θ1, were the true one, how ‘likely’ would it be for us to observe a height of 1.7m? How about θ2: how likely would it be to observe a height of 1.7m if θ2 were the ‘correct’ model?



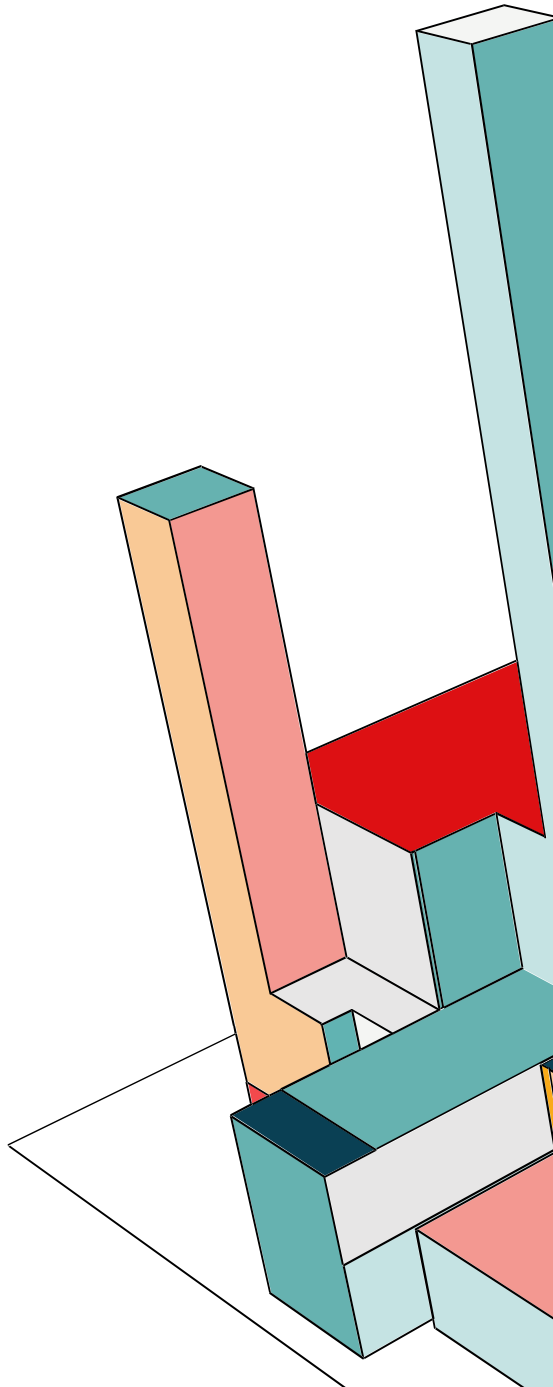
THE LIKELIHOOD

- Indeed what we want at this stage is a function that will systematically look at every possibility for the model θ and find the probability that this model would have ‘produced’ or ‘predicted’ the observed value. Remember that in this case, our model θ is defined as N(μ,σ), i.e. a normal distribution with mean μ and standard deviation σ. We are also keeping σ constant to simplify the process. Therefore our function will take in possible values for μ as our ‘independent’ variable, our observed data point of 1.7m, and it will output the probabilities that each model is the correct model as our ‘dependent’ variable. Note that we run into a slight hiccup here: the ‘probability’ that a particular model is the correct one would technically be zero because, in theory, the possibilities for θ are endless. For this reason, I discretized the possibilities for θ, making it such that there are 50 options for θ between 1.65m and 1.8m. We then use the probability density functions for each proposed model to evaluate the probability density of the observed datum for a particular model. The probability density isn’t the same as the probability, it just gives us a relative measure of how likely it is that the point was observed given each of the model options. To get true probabilities, we would have to ‘normalize’ the probability densities, making it such that adding all of their values would give us 1.



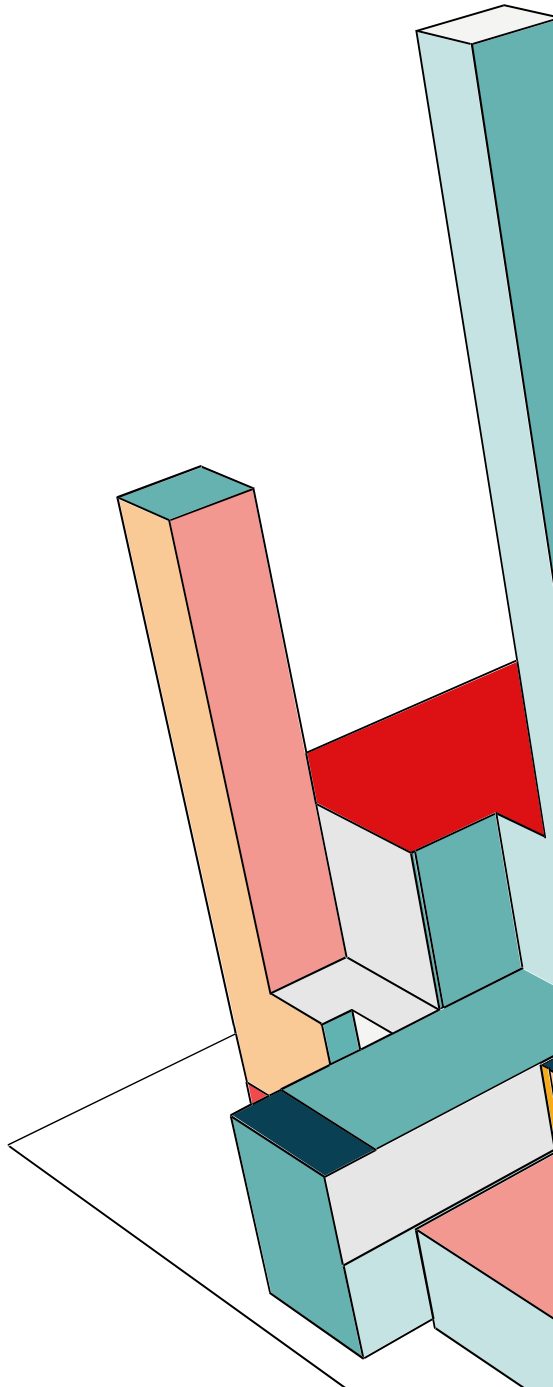
THE LIKELIHOOD

- This function is called the ‘likelihood’ function, and we typically define it by the probability ‘density’ function (PDF) of the model we are proposing, evaluated at the new data point. Thus, we want the “PDF” of the distribution N(μ, 0.1m) for the data point 1.7m.
- Note that for those who have used PDFs before this seems a bit backward. With PDFs, we usually have a fixed model, e.g. N(1.8,0.1), and we use it to evaluate the probability of different values for our variable *h*. This would mean that we would have the variable *h* on the x-axis, and the probability density on the y-axis.



THE LIKELIHOOD

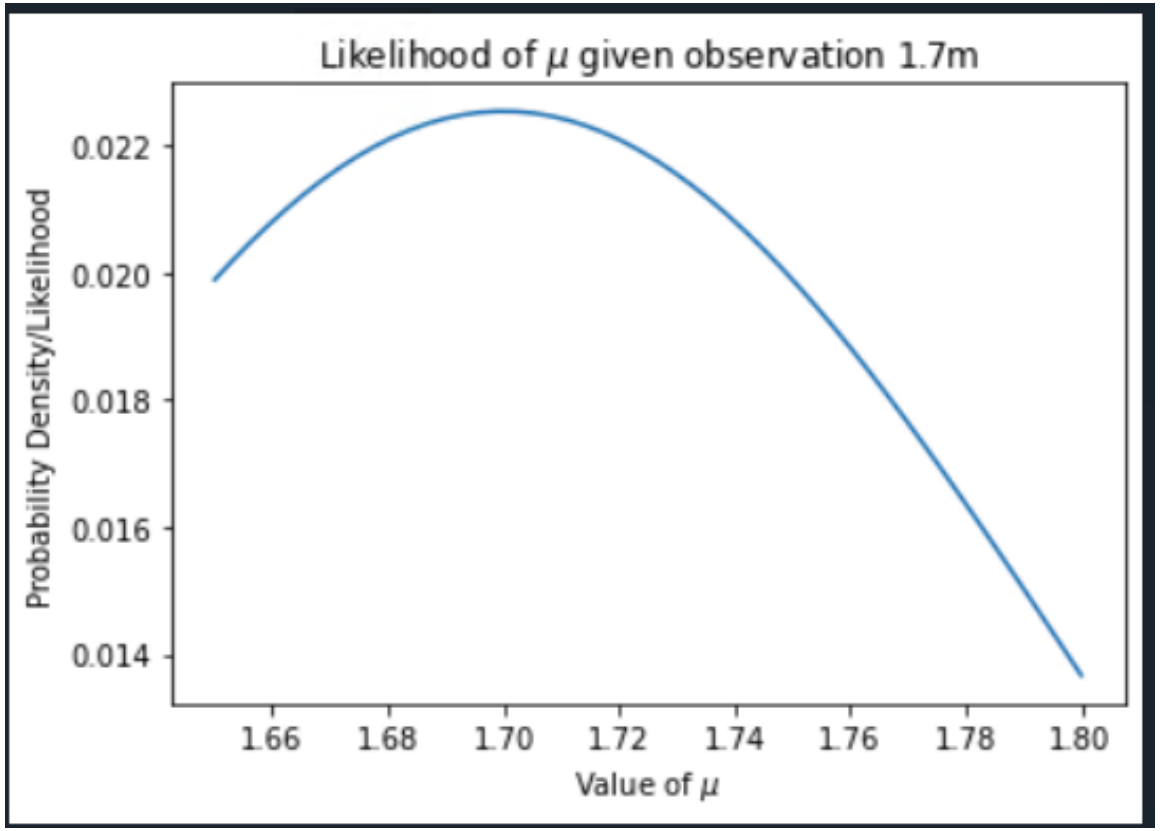
- For our current purposes, however, we are varying the distribution/model itself. This means that our x-axis will actually have the different possibilities for our variable μ, and our y-axis will have the probability density for each of these possibilities. Have a look at the code below, which represents our likelihood function and visualization for it:



SAMPLE SCRIPT/S

```
"""
Created on Sat Mar  9 10:21:10 2024

@author: jcprado
"""

def likelihood_func(datum, mu):
  likelihood_out = sts.norm.pdf(datum, mu, scale = 0.1) #Note that mu here is an array of values, so the output is also an array!
  return likelihood_out/likelihood_out.sum()

likelihood_out = likelihood_func(1.7, mu)

plt.plot(mu, likelihood_out)
plt.title("Likelihood of $\mu$ given observation 1.7m")
plt.ylabel("Probability Density/Likelihood")
plt.xlabel("Value of $\mu$")
plt.show()
```

SAMPLE SCRIPT/S

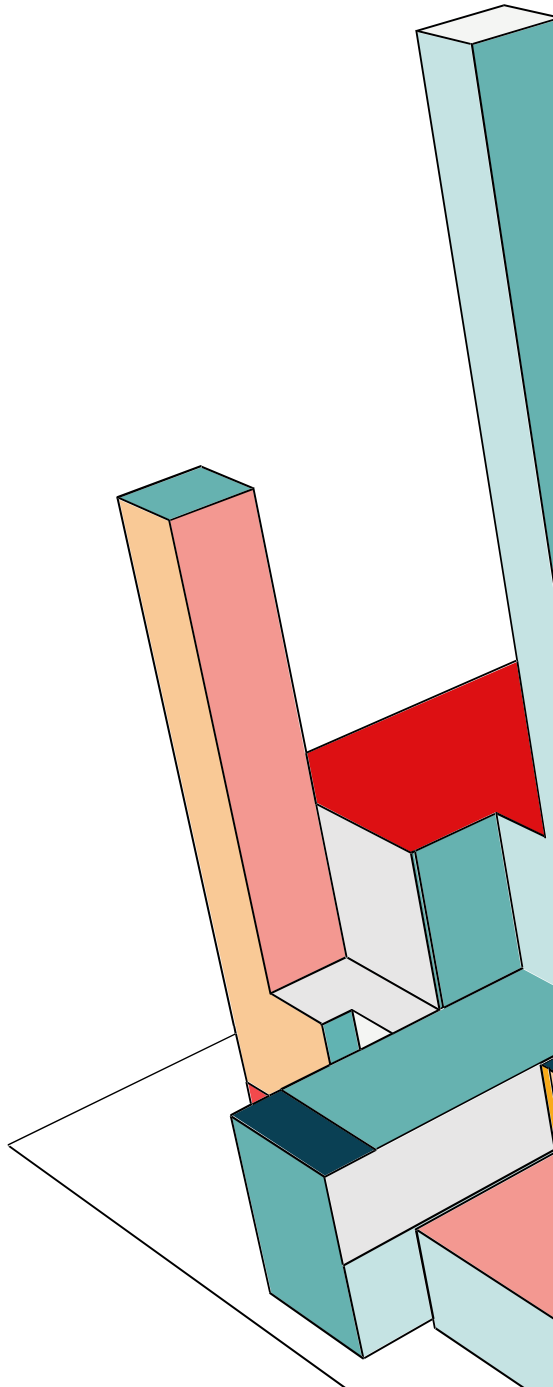


THE EVIDENCE

- Some statisticians call P(Data) the ‘evidence’. The meaning of this variable is pretty straightforward: it is the probability that value Data is produced. However, this is tough to calculate directly. Thankfully, we have a good trick up our sleeves.

- Consider the following expression:

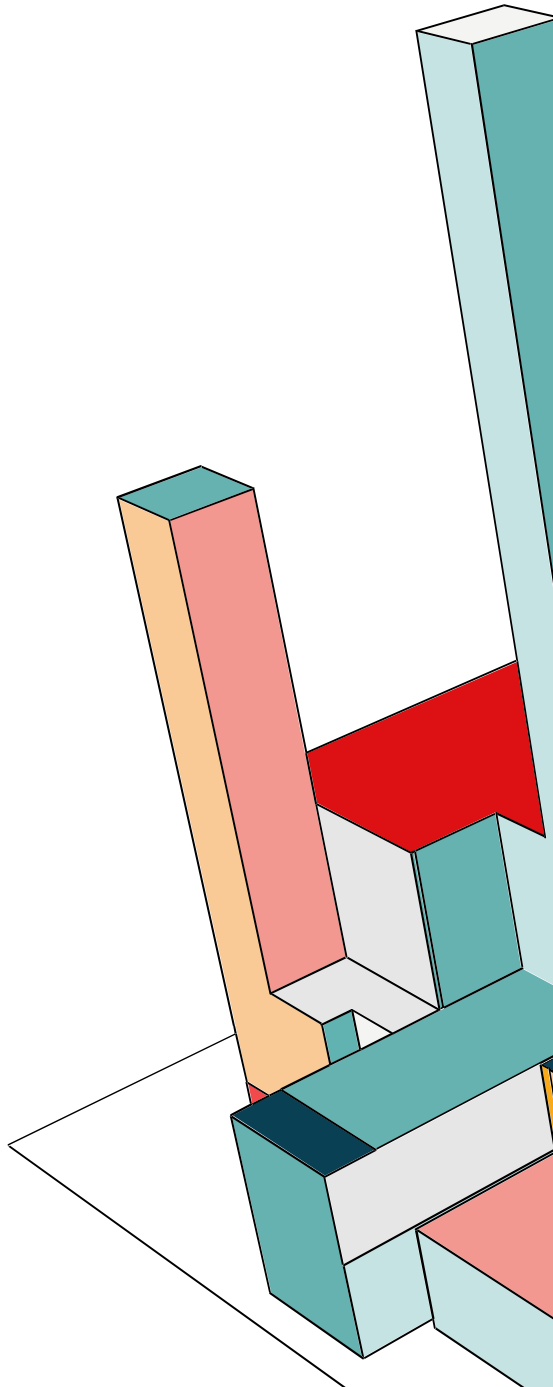
$$\int P(Data|\theta) * P(\theta) d\theta = P(Data)$$



THE POSTERIOR

- The right-hand side of Bayes' theorem, P(θ|Data), is called the 'posterior'. It is our posterior understanding of how the data is distributed given that we witnessed the data, and that we had a prior about it.
- How do we get the posterior? Going back to the equation:

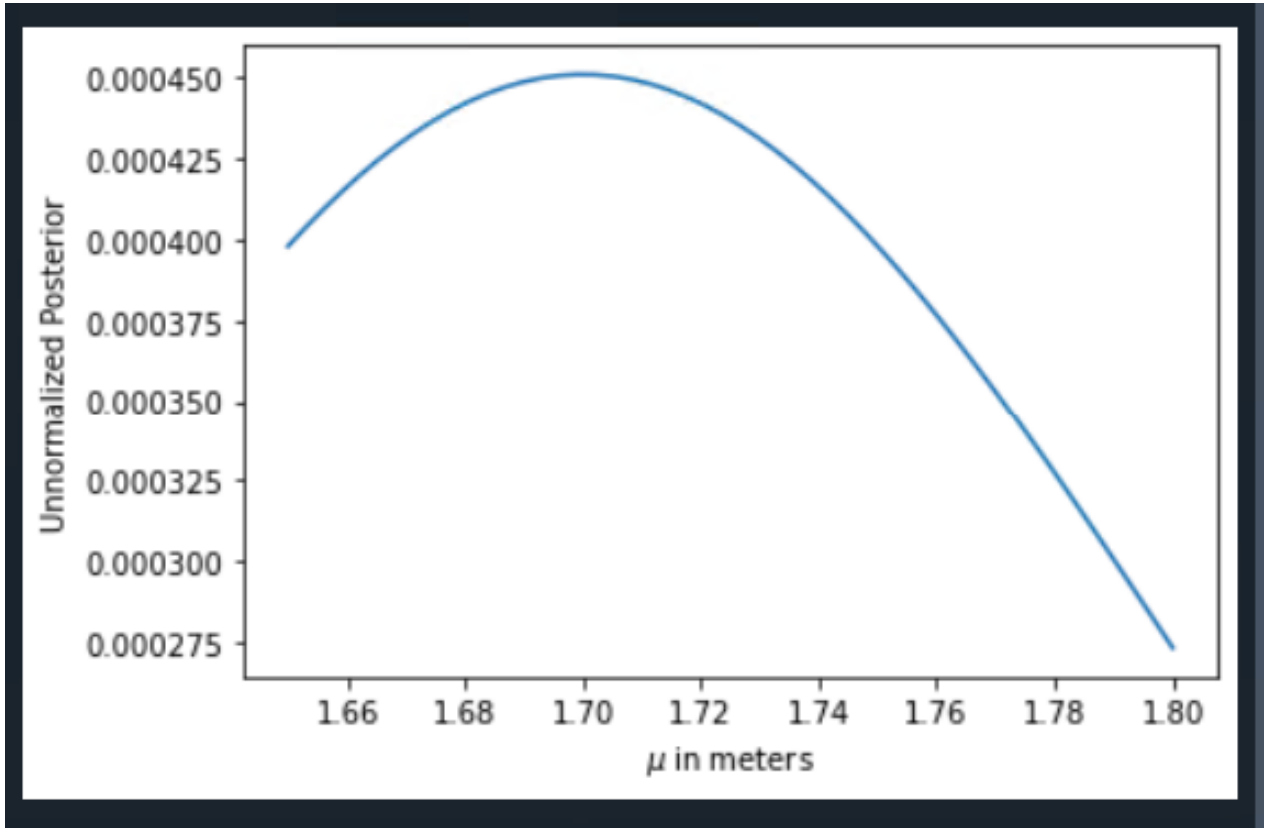
$$P(\theta|Data) = \frac{P(Data|\theta) * P(\theta)}{P(Data)}$$



SAMPLE SCRIPT/S

```
"""
Created on Sat Mar  9 10:21:10 2024

@author: jcprado
"""
import scipy as sp

unnormalized_posterior = likelihood_out * uniform_dist
plt.plot(mu, unnormalized_posterior)
plt.xlabel("$\mu$ in meters")
plt.ylabel("Unnormalized Posterior")
plt.show()
```

SAMPLE SCRIPT/S



THE POSTERIOR

- We can see how, if we were to sum the probability densities in the unnormalized posterior, we would have a value that is different than 1:

```
unnormalized_posterior.sum()
```

```
0.01999999999999997
```

- Note that we still have 'P(Data)' to contend with, hence why I call this the 'unnormalized_posterior'.
- Remember from above that:

$$P(Data) = \int P(Data|\theta) * P(\theta) d\theta$$