EMD Demo

Jeffrey Chandler

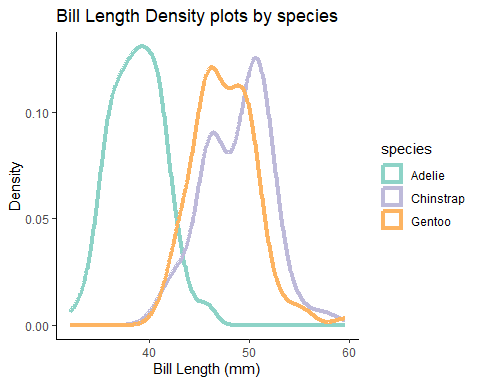
2023-06-29

pkgs <- c("tidyverse", "emdist", "ggpattern", "transport", "FamilyRank",  
 "Compositional", "philentropy", "MASS", "reshape2", "ggExtra", "terra",  
 "viridisLite", "ggpubr", "palmerpenguins", "RColorBrewer")  
invisible(lapply(pkgs, library, character.only = T))  
  
penguin <- penguins %>% drop\_na()  
colors <- brewer.pal(9,"Set3")[c(1,3,6)]

# 1 Problem motivation

We are explorers in the Southern Ocean. We are studying three different species of penguins, Adelie, Gentoo, and Chinstrap penguins. We can identify the species, however we are making a guidebook for visitors. We want to produce a key based on anatomy that requires the minimal amount of information. This also provides scientists some benefit because we may be able infer evolutionary relationships based on similarity (similar to existing work into genetics [Schiebinger et al., 2019]<https://www.cell.com/cell/fulltext/S0092-8674(19)30039-X?_returnURL=https%3A%2F%2Flinkinghub.elsevier.com%2Fretrieve%2Fpii%2FS009286741930039X%3Fshowall%3Dtrue>). First, let’s look at one trait between the penguins, bill length. Let’s look at their density distributions

#setosaXversicolor <- filter(iris, Species == "setosa" | Species == "versicolor")  
#versicolor <- filter(iris, Species == "versicolor")  
#virginica <- filter(iris, Species == "virginica")  
  
  
ggplot(penguin, aes(x = bill\_length\_mm, color = species))+  
 geom\_density(fill = NA, linewidth = 1.5)+  
 scale\_color\_manual(values = colors)+  
 xlab("Bill Length (mm)")+  
 ylab("Density")+  
 ggtitle("Bill Length Density plots by species")+  
 theme\_classic()



This is quite useful. We can see that Adelie penguins have smaller bill lengths than Gentoo and Chinstrap penguins. However, how far away? That information may be useful as we look at other characteristics. The first thing we could do is to simply find the difference between the means

means <- penguin %>%  
 group\_by(species)%>%  
 summarize(mean\_bill\_length = mean(bill\_length\_mm))  
  
print(paste0("Adelie to Chinstrap Distance = ",  
 means$mean\_bill\_length[which(means$species == "Adelie")] - means$mean\_bill\_length[which(means$species == "Chinstrap")]))

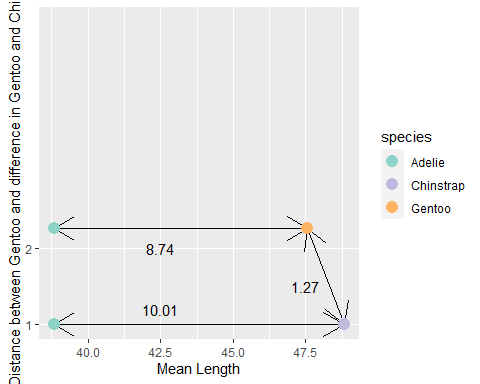
## [1] "Adelie to Chinstrap Distance = -10.009850926672"

print(paste0("Adelie to Gentoo Distance = ",  
 means$mean\_bill\_length[which(means$species == "Adelie")] - means$mean\_bill\_length[which(means$species == "Gentoo")]))

## [1] "Adelie to Gentoo Distance = -8.74409462415102"

print(paste0("Chinstrap to Gentoo Distance = ",  
 means$mean\_bill\_length[which(means$species == "Chinstrap")] - means$mean\_bill\_length[which(means$species == "Gentoo")]))

## [1] "Chinstrap to Gentoo Distance = 1.26575630252101"



We can see that Adelie is quite smaller than Chinstrap and Gentoo, which are quite close to each other. Let’s see what happens when we swap the comparison around

print(paste0("Gentoo to Chinstrap Distance = ",  
 means$mean\_bill\_length[which(means$species == "Gentoo")] - means$mean\_bill\_length[which(means$species == "Chinstrap")]))

## [1] "Gentoo to Chinstrap Distance = -1.26575630252101"

We get the same value, just with the opposite sign. That makes sense.

A problem is, however, we have uncertainty around that mean value. What if our data when we go elsewhere no longer applies because of larger variance? The next step we can take is to measure distance in terms of the variance around the data. We can use Standard deviations to see how far they are while controlling for that variation. However, how do we decide which standard deviation to choose from? We can choose from the “reference” species. Let’s try that

means <- penguin %>%  
 group\_by(species)%>%  
 summarize(mean\_bill\_length = mean(bill\_length\_mm), sd\_bill\_length = sd(bill\_length\_mm))  
  
print(paste0("Adelie to Chinstrap Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Adelie")] - means$mean\_bill\_length[which(means$species == "Chinstrap")])/means$sd\_bill\_length[which(means$species == "Adelie")]))

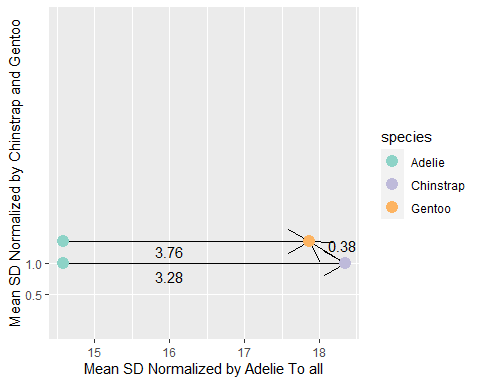
## [1] "Adelie to Chinstrap Distance = -3.75943183990646"

print(paste0("Adelie to Gentoo Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Adelie")] - means$mean\_bill\_length[which(means$species == "Gentoo")])/means$sd\_bill\_length[which(means$species == "Adelie")]))

## [1] "Adelie to Gentoo Distance = -3.28404768282773"

print(paste0("Chinstrap to Gentoo Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Chinstrap")] - means$mean\_bill\_length[which(means$species == "Gentoo")])/means$sd\_bill\_length[which(means$species == "Chinstrap")]))

## [1] "Chinstrap to Gentoo Distance = 0.379053400508038"

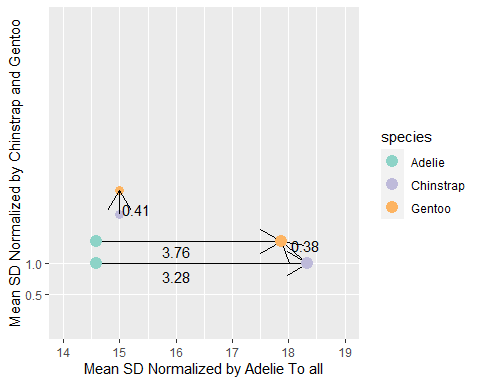


Let’s see what happens when we swap the order though

print(paste0("Gentoo to Chinstrap Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Gentoo")] - means$mean\_bill\_length[which(means$species == "Chinstrap")])/means$sd\_bill\_length[which(means$species == "Gentoo")]))

## [1] "Gentoo to Chinstrap Distance = -0.407504557793128"

adj <- 13.5  
segments <- data.frame(starty = means$mean\_bill\_length[3]/means$sd\_bill\_length[3]-adj, endy = means$mean\_bill\_length[2]/means$sd\_bill\_length[3]-adj,  
 startx = 15, endx = 15)  
plot\_by\_sd +  
 geom\_point(aes(x = c(15,15), y =c(means$mean\_bill\_length[3]/means$sd\_bill\_length[3], means$mean\_bill\_length[2]/means$sd\_bill\_length[3])-adj), color = colors[c(2,3)], size = 3 )+  
 geom\_segment(aes(x = startx, xend = endx, y = starty, yend = endy), data = segments,  
 arrow = arrow(ends = "last"))+  
 annotate("text",x = 15.3, y = 1.9,label = abs(round((means$mean\_bill\_length[which(means$species == "Gentoo")] - means$mean\_bill\_length[which(means$species == "Chinstrap")])/means$sd\_bill\_length[which(means$species == "Gentoo")],2)))+  
 scale\_x\_continuous(limits = c(14,19))



We get different values. That’s frequently not a desirable characteristic in a distance measure because you now must care about the order in which you perform operations. One way we can get around this is by pooling the variance

print(paste0("Adelie to Chinstrap Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Adelie")] - means$mean\_bill\_length[which(means$species == "Chinstrap")])/(means$sd\_bill\_length[which(means$species == "Adelie")]+ means$sd\_bill\_length[which(means$species == "Chinstrap")])))

## [1] "Adelie to Chinstrap Distance = -1.66779352129546"

print(paste0("Adelie to Gentoo Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Adelie")] - means$mean\_bill\_length[which(means$species == "Gentoo")])/(means$sd\_bill\_length[which(means$species == "Adelie")]+ means$sd\_bill\_length[which(means$species == "Gentoo")])))

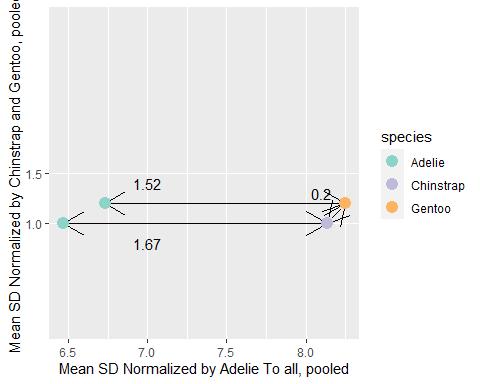
## [1] "Adelie to Gentoo Distance = -1.5157792553238"

print(paste0("Chinstrap to Gentoo Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Chinstrap")] - means$mean\_bill\_length[which(means$species == "Gentoo")])/(means$sd\_bill\_length[which(means$species == "Chinstrap")]+ means$sd\_bill\_length[which(means$species == "Gentoo")])))

## [1] "Chinstrap to Gentoo Distance = 0.196382207723929"

print(paste0("Gentoo to Chinstrap Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Gentoo")] - means$mean\_bill\_length[which(means$species == "Chinstrap")])/(means$sd\_bill\_length[which(means$species == "Gentoo")] + means$sd\_bill\_length[which(means$species == "Chinstrap")])))

## [1] "Gentoo to Chinstrap Distance = -0.196382207723929"



This corrects the problem of directionality while still incorporating the variance of the underlying distributions. The next step we can make is by controlling for the variance in the means. Right now we are just controlling for variance in the distributions. We can use our sample size of each penguin to calculate “standard error”, the expected variation within our mean value. When we normalize by standard error, we need to square the standard error around each mean, then take the square root of that sum.

means <- penguin %>%  
 group\_by(species)%>%  
 summarize(counts = n(), mean\_bill\_length = mean(bill\_length\_mm), sd\_bill\_length = sd(bill\_length\_mm) )%>%  
 mutate(se\_bill\_length = (sd\_bill\_length/sqrt(counts))^2)  
  
print(paste0("Adelie to Chinstrap Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Adelie")] - means$mean\_bill\_length[which(means$species == "Chinstrap")])/sqrt(means$se\_bill\_length[which(means$species == "Adelie")]+ means$se\_bill\_length[which(means$species == "Chinstrap")])))

## [1] "Adelie to Chinstrap Distance = -21.7124980566359"

print(paste0("Adelie to Gentoo Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Adelie")] - means$mean\_bill\_length[which(means$species == "Gentoo")])/sqrt(means$se\_bill\_length[which(means$species == "Adelie")]+ means$se\_bill\_length[which(means$species == "Gentoo")])))

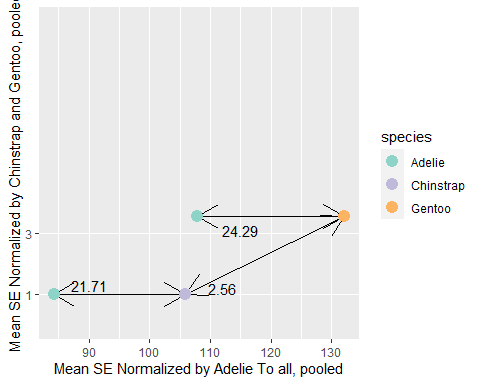
## [1] "Adelie to Gentoo Distance = -24.2860665004714"

print(paste0("Chinstrap to Gentoo Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Chinstrap")] - means$mean\_bill\_length[which(means$species == "Gentoo")])/sqrt(means$se\_bill\_length[which(means$species == "Chinstrap")]+ means$se\_bill\_length[which(means$species == "Gentoo")])))

## [1] "Chinstrap to Gentoo Distance = 2.55692638319214"

print(paste0("Gentoo to Chinstrap Distance = ",  
 (means$mean\_bill\_length[which(means$species == "Gentoo")] - means$mean\_bill\_length[which(means$species == "Chinstrap")])/sqrt(means$se\_bill\_length[which(means$species == "Gentoo")] + means$se\_bill\_length[which(means$species == "Chinstrap")])))

## [1] "Gentoo to Chinstrap Distance = -2.55692638319214"



Now we get distances that don’t care about direction and incorporate the underlying variance of our known means. We can see that Adelie penguins are very different from Chinstrap and Gentoo penguins, which are quite similar to each other. In fact, we stumbled upon the formulation for the two sample T-statistic. We can actually see this by doing a t test in base R. Let’s do Adelie to Chinstrap Penguins.

ad <- penguin%>%filter(species == "Adelie")  
chi <- penguin %>%filter(species == "Chinstrap")  
ge <- penguin %>%filter(species == "Gentoo")  
test <- t.test(ad$bill\_length\_mm,  
 chi$bill\_length\_mm, var.equal = F)  
  
print(paste0("the test statistic/ distance is : ",test$statistic))

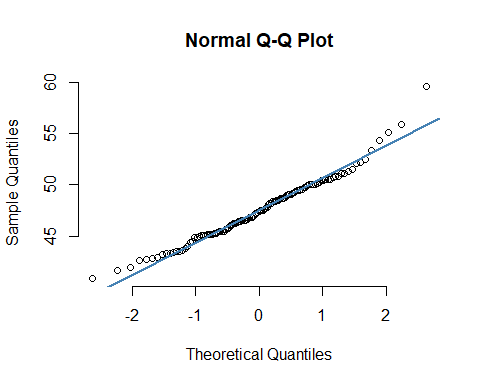
## [1] "the test statistic/ distance is : -21.7124980566359"

This works well. However, implicit within the definitions of standard deviation and standard error are that the data are normally distributed. Not all data sets are normally distributed. In fact, Gentoo penguins appear to be non-normal according to the Shapiro-Wilk normality test at .05 significance level. I will admit, Gentoo is normal enough that we shouldn’t throw out the idea that it approximates a normal distribution.

shapiro.test(ge$bill\_length\_mm)

##   
## Shapiro-Wilk normality test  
##   
## data: ge$bill\_length\_mm  
## W = 0.97379, p-value = 0.01989

qqnorm(ge$bill\_length\_mm, pch = 1, frame = FALSE)  
qqline(ge$bill\_length\_mm, col = "steelblue", lwd = 2)



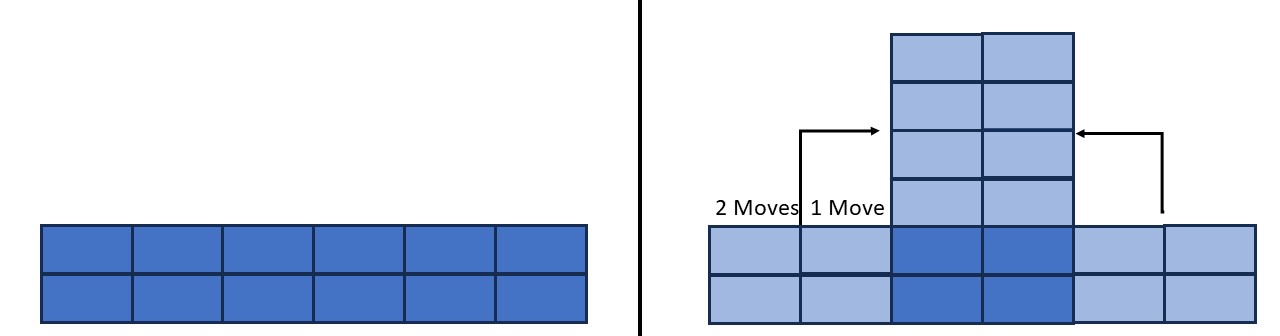
How can we measure distance outside of the framework of simple distance, and the various forms of standardized distances?

# 2 Intuition and Calculation of Earth Mover’s Distance.

Let’s take our problem and recontextualize it slightly. Instead of measuring from mean to mean, potentially adjusting by variance, lets “move” one distribution to the next, as if we are moving earth from one pile to another. We can think of the actual computation for this using children’s blocks. We have a wall with 12 blocks, 6 x 2. Let’s build a 2 x 6 tower from that wall moving the fewest blocks in the most efficient way. Moving vertically has no cost, but moving laterally does. The cost is 1 for a block moved one block over.

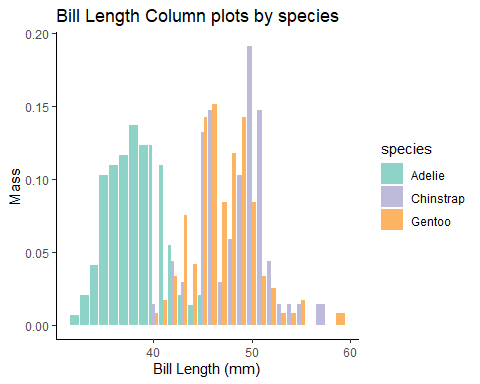
What is the effort required to make these moves using minimum effort?

knitr::include\_graphics("Children\_blocks\_conceptual\_model.jpg")



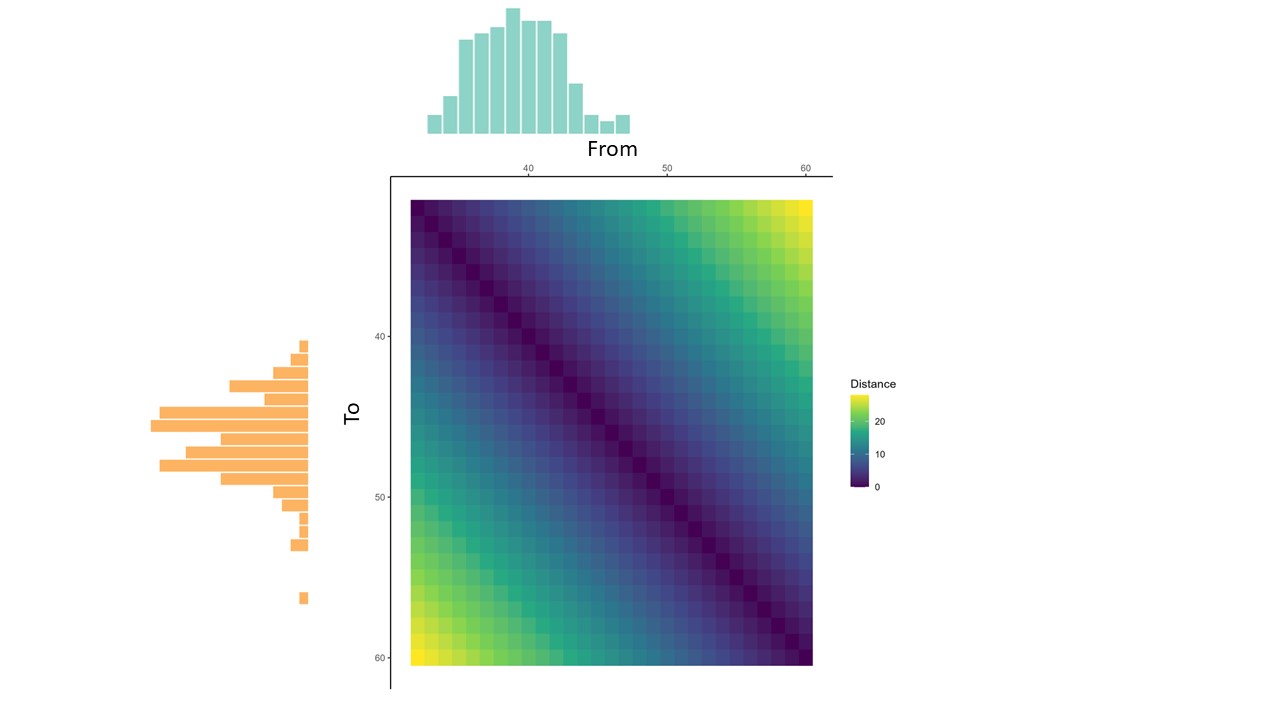
From this simple diagram we can deduce logically that we make eight moves. Four with a cost of one, and four with a cost of two. If each block is a “mass” of 1, then we can say our effort was 12. If we reframe this to a probability mass function, then the sum of blocks is = 1, that means each block has a mass of 1/12. Our effort is now 1. Let’s apply this logic to the penguins dataset. First we need to discretize our data. I’m going to choose a binwidth of 0.05 (aka dx = 0.05).

## `summarise()` has grouped output by 'species'. You can override using the  
## `.groups` argument.  
## Adding missing grouping variables: `species`



This shows similar information as the density plots above, but now the data is discrete and we can track how much “mass” we move and where. Next lets model the costs of moving. This can be be performed by making a matrix of costs (C) where columns indicate “from” and rows indicate “to”. so the cell index [4,2] would be the cost of moving any mass from 2 to 4. In this case, let’s choose the standard euclidean distance (e.g. . See “Cost Matrix Changes” in the supplementary information for more information). For illustrative simplicity, let’s remove Chinstrap penguins so we only focus on one computation.

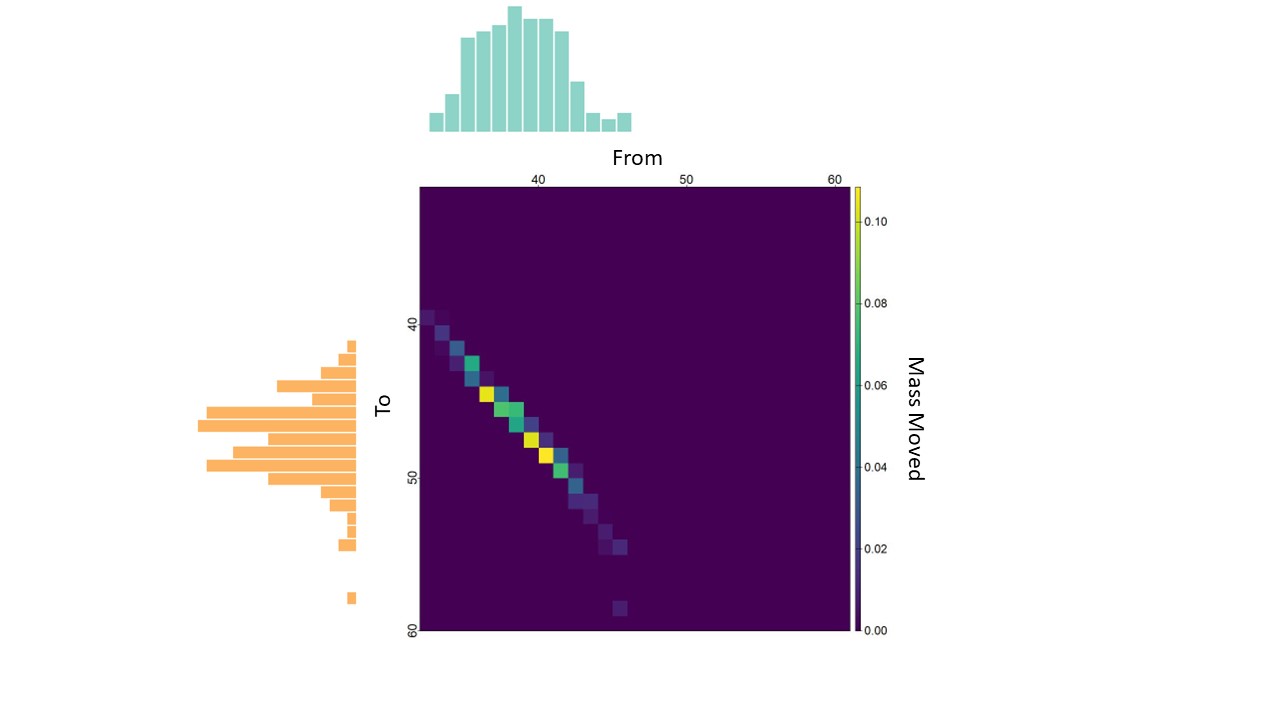
n <- length(bins)  
C <- matrix(nrow = n, ncol = n)  
for(i in 1:n){  
 for(j in 1:n){  
 C[i,j] = sqrt((bins[i]-bins[j])^2)  
 }  
}  
dimnames(C) <- list(x = bins, y = bins)  
C\_df <- melt(C, value.name = "intensity")  
  
plot <- ggplot(C\_df) +  
 geom\_raster(aes(x = x, y = -y, fill = intensity))+  
 scale\_fill\_viridis\_c()+  
 scale\_x\_continuous(position = "top")+  
 scale\_y\_continuous(breaks = c(-40,-50,-60),  
 labels = c(40,50,60))+  
 labs(x = NULL,  
 y = NULL,  
 fill="Distance")+  
 theme\_classic()  
  
# ggarrange(  
# empty, pen\_disc\_ad,   
# pen\_disc\_ge, plot,  
# nrow = 2, ncol = 2,  
# widths = c(1,3),  
# heights = c(1,2)  
#   
# )



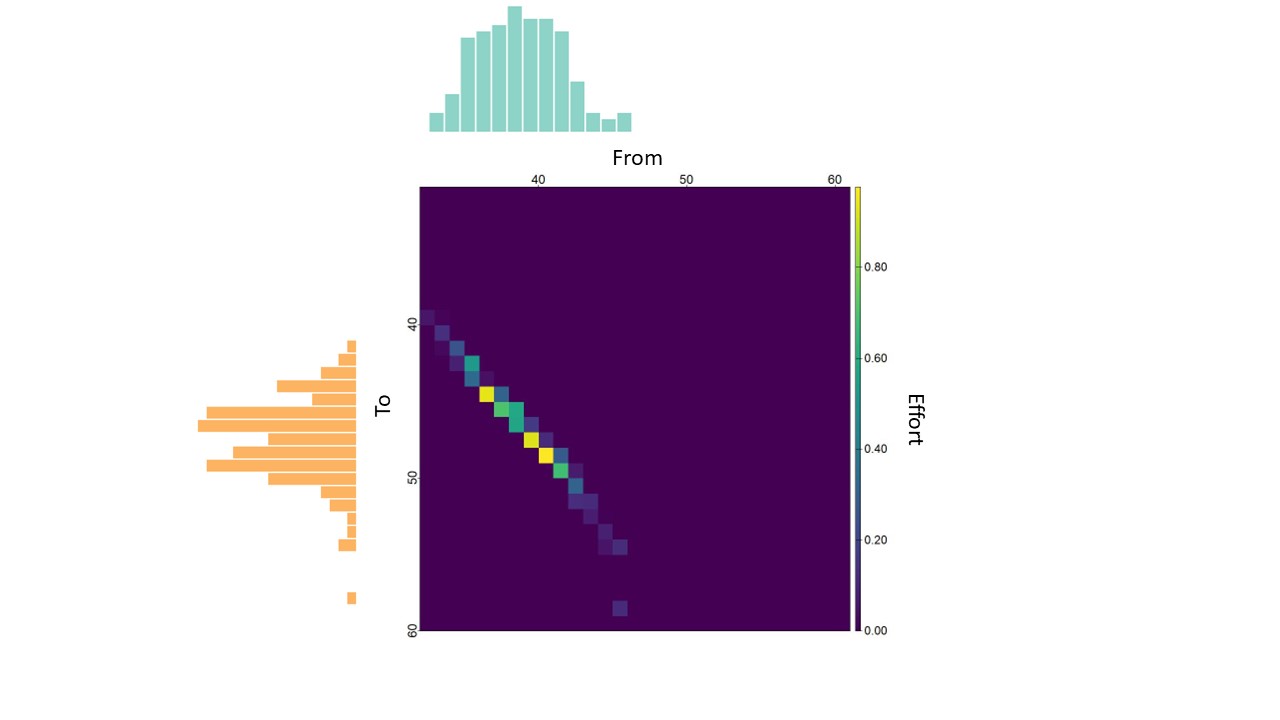
Here we can see that the 1:1 line is 0. That makes sense because if we move from a place to the same place, we don’t move at all. The highest costs are at the corners, where you move from the beginning of the range to the end of the range. Also note, this matrix is symmetric. Moving from 2 to 4 is equal to moving from 4 to 2. We can see from the marginal plots and the histograms above that most of the Adelie penguins are smaller than the Gentoo penguins.

The next step is to actually find the minimum effort to move from Adelie to Gentoo. While there are multiple ways of doing this (a few I will discuss in the supplementary information) we’re going to use the standard and general approach. We are going to use a linear program solver to solve for the “transport map” that minimizes “effort”. Effort is the cost multiplied by the transport map. Linear programming is a field of mathematics where the objective is to solve systems of linear equations given a minimization or maximization problem and constraints. Linear programming is used frequently in business for maximizing profits or minimizing costs given a real world constraint on resources. What we want to do is to find the transport map that minimizes the effort given the constraits that all of our first distribution moves to the second distribution (i.e. conserving mass) and that all mass moved is positive (so we don’t create negative masses). In math formulation this looks like

T is our transport map, C is the cost matrix, and i and j are the row and column indecies. 1 is a vector of 1s that form our contraints that all of Adelie () must move to all of Gentoo (). Let’s construct the linear system of equations and compute the transport map using the lpsolve::lp function (code is documented in the RMD file)

 Here we can see that we are moving mass from the 30-45 range of Adelie to the 40-60 range of Gentoo penguins. We can see where most of our mass is moved to, and that it corresponds to the peaks of Gentoo. Now we can calculate the total effort by multiplying C by T to generate an effort map, then adding every cell together to calculate the total effort

Effort Map



Effort

calcTmap <- sum(solved$solution\*c(C))  
paste0("This is equal to sum(transport\_map \* C): ", calcTmap)

## [1] "This is equal to sum(transport\_map \* C): 8.7727063428111"

As a sanity check, lets check our manual calculation the lpsolve:lp function’s “objective function” attribut that should be equivalent.

calcObj <- solved$objval  
paste0("The calculated solution is: ", calcObj)

## [1] "The calculated solution is: 8.7727063428111"

We can see that lpsolve::lp objval provides the same value as calculating a transport map, multiplying by C, then summing every cell. With that information we can say that the effort from Adelie to Gentoo bill length is $8.77 $. If we compare that to our prior euclidean distance measure of 8.74, 8.77 is quite close but slightly different because the shapes of the distributions are slightly different. What we have just calculated is called the “Earth Mover’s Distance” (EMD).

one nice thing about this metric is that packages for calculating EMD already exist. “transport” and “emdist” both can be used to calculate the EMD.

dist\_ad\_df <- as.matrix(data.frame(weights = dist1, locations = bins))  
dist\_ge\_df <- as.matrix(data.frame(weights = dist2, locations = bins))  
dist\_chi\_df <- as.matrix(data.frame(weights = dist3, locations = bins))  
emdist\_emd <-emdist::emd(dist\_ad\_df,dist\_ge\_df)  
paste0("emdist package = ", emdist\_emd)

## [1] "emdist package = 8.77270603179932"

transport\_emd <-transport::wasserstein1d(bins,bins,1,dist1,dist2)  
paste0("transport package = ", transport\_emd)

## [1] "transport package = 8.7727063428111"

Let’s check the distances from Adelie to Chinstrap, and Chinstrap to Gentoo

## [1] "Adelie to Chinstrap = 9.95547945205479"

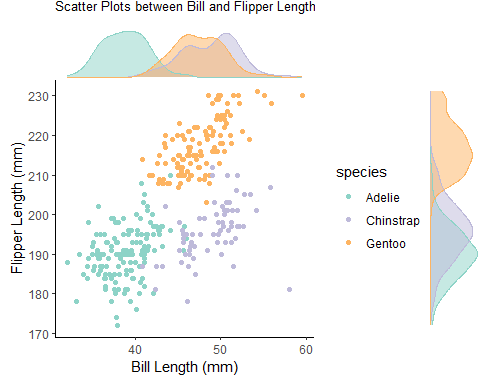
## [1] "Chinstrap to Gentoo = 1.22899159663866"

Again, pretty similar to the Euclidean distances, but not exactly because the distributions are not perfectly normal. These distance quite easily can be used to compare different groups in a more robust way. There is also room to further develop the metric to be more robust. The metric also works in multiple dimensions. A note on EMD in multiple dimensions, data should be converted to a gridded probability map or using the Compositional::mkde function to normalize the data by density.

if you want to see what the transport cost looks like in 2D, or you want another resource to learn about Optimal transport, the resource at this link: [A Short Introduction to Optimal Transport and Wasserstein Distance]<http://alexhwilliams.info/itsneuronalblog/2020/10/09/optimal-transport/>

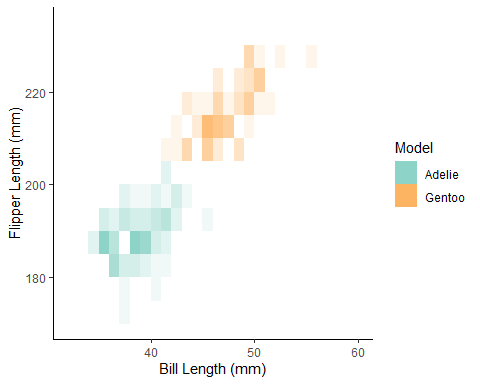
Bill and Flipper Length Scatter plots.

pen\_2D <- penguin %>%  
 ggplot(aes(x=bill\_length\_mm,y = flipper\_length\_mm, color = species))+  
 geom\_point()+  
 scale\_color\_manual(values = colors)+  
 xlab("Bill Length (mm)")+  
 ylab("Flipper Length (mm)")+  
 ggtitle("Scatter Plots between Bill and Flipper Length")+  
 theme\_classic()+  
 theme(plot.title = element\_text(size = 10))  
ggMarginal(pen\_2D,groupColour = T, groupFill = T, type = "density")

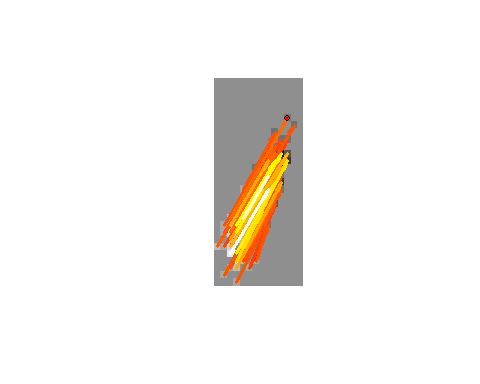


We will focus on Adelie to Gentoo for 2D EMD

Gridded density plots using a dx of 1 and dy of 5



Heatmap of moving Adelie to Gentoo in 2D with EMD values using different implementations



## [1] "EMD using wasserstein() = 21.8755206489158"

## [1] "EMD using emd2d() = 28.0444965362549"

## [1] "EMD using emd() = 26.4789047241211"

## [1] "The emdist calculated EMD with points set to weight = 1 is: 26.4595813751221"

# 3 Supplemental

## 3.1 EMD definitions

Earth Mover’s Distance (EMD; A.K.A. 1-Wasserstein metric, Kantorovich distance, Mallows Distance, Ocean Mover’s distance, Fortet–Mourier distance, or minimal L distance) is the minimum energy required to move mass (or “earth”) from a distribution P to a distribution Q.

EMD is a special case of “Optimal Transport”, specifically the “wasserstein metric”. The Wasserstein metric is defined as

The function is defined over a dimension space, so if the two distributions are 1D, the function space is 2D. and are probability density functions. is an exponential scaling factor (A.K.A. a finite-moment). EMD is when p = 1. inf can be read as the minimum (its technically an infimum). is all couplings of and . is a joint probability measure whose marginals are and on the first and second factors, in effect, constraining our joint density between the two distributions to be equivalent. is the expected value of of all derivatives from P to Q. This problem can be set up as a linear programming problem for computation.

Linear programming is solving linear systems of equations with constraints. It’s usage in Optimal Transport is to constrain to real world limits, whether that be cost, distance, or other things we want to try to minimize/maximize

$$\ $$

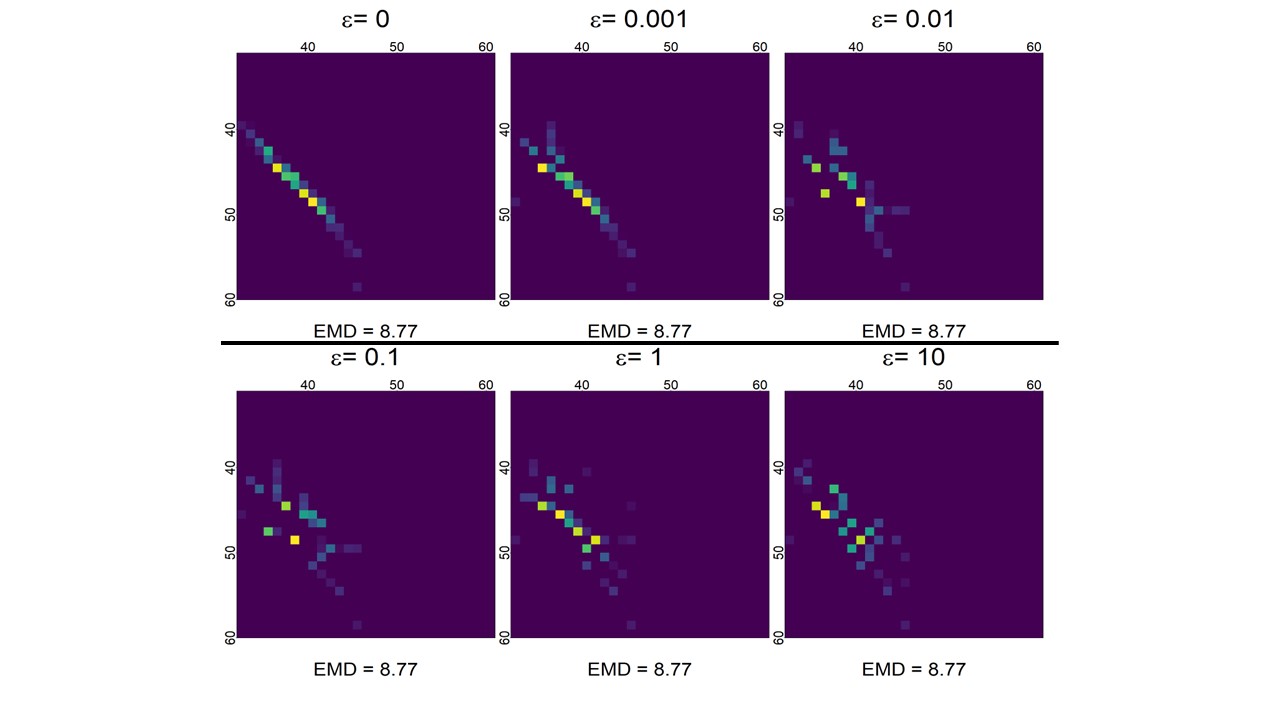
Where T is a transport Matrix, C is the Cost Matrix of movement from to . and are the values within each matrix corresponding to a cost in C and a “move” in T. We minimize The transport plan by cost such that we don’t add mass, which is what the “subject to” terms are performing. We can use a linear program solver to solve for the transport plan, and EMD is the sum of T\*C.

Optimal transport has been referred to as “the natural geometry for probability measures” ([Cuturi, 2019]<https://www.youtube.com/watch?v=6iR1E6t1MMQ>)

## 3.2 Entropy Regularlization

The calculation time for EMD scales poorly, however [Altschuler et al., 2019]<https://arxiv.org/abs/1705.09634>; [Dvurechensky et al., 2019]<https://arxiv.org/abs/1802.04367>, and [CH 4 Peyré & Cuturi (2019)]<http://dx.doi.org/10.1561/2200000073> have introduced the usage of regularizing by shannon entropy of T

where $>0 $ and is a regularization penalty. which is the shannon entropy. As $$ the Transport plan approaches . This logically makes the plan easier to solve for. The way we implement this is by subtracting off from C in our implementation. EMD is calculated by where is our transport plan calculated from regularized data.



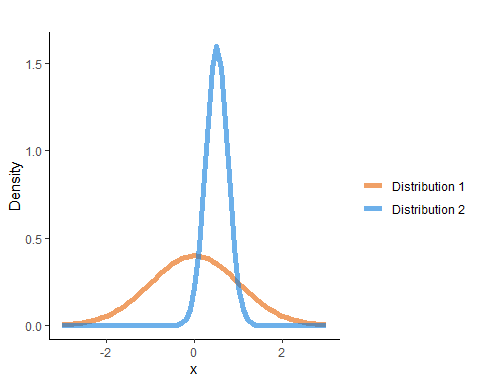
## 3.3 Analytical solutions

### 3.3.1 Univariate case

When working with univariate distributions you can compute analytic solutions. The proof is beyond the scope of this document however the formulation is let be the inverse CDF of our pdfs where u is a quantile from 0 to 1. We can define the wasserstein metric to be . If the formula can further reduce to where are the CDF’s of pdfs which looks like

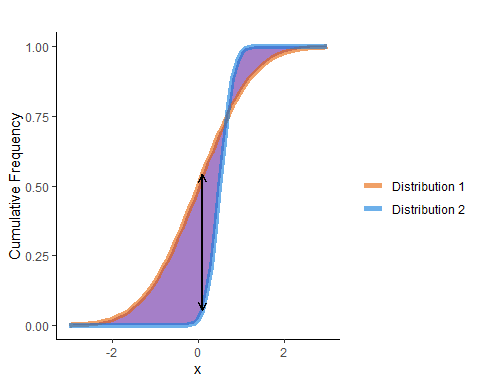
x <- seq(-3,3,.1)  
dx <- x[2]-x[1]  
dist1 <- dnorm(x)  
dist2 <- dnorm(x, .5, .25)

Normal PDFs

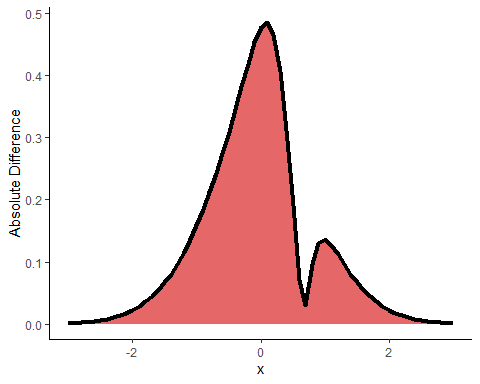


Normal CDFs with shaded absolute differences

dist1\_cdf <- pnorm(x)  
dist2\_cdf <- pnorm(x, .5, .25)  
df\_1 <- data.frame(factors = x, distribution1 = dist1\_cdf, distribution2 = dist2\_cdf)  
  
df\_1 <- df\_1 %>%  
 mutate(difference = distribution2-distribution1, absoluteDifference = abs(difference))  
df\_1\_ribbon <- df\_1 %>%  
 mutate(  
 ymax = pmax(distribution1, distribution2),  
 ymin = pmin(distribution1, distribution2),  
 fill = TRUE  
 )  
df\_max\_diff <- df\_1\_ribbon[which.max(df\_1\_ribbon$absoluteDifference),1:7]  
colors <- c("#E66100","#0C7BDC")  
df\_1\_long <- df\_1 %>%  
 pivot\_longer(c(distribution1, distribution2,), names\_to = "distribution", values\_to = "val")  
df\_1\_long$distribution <- as.factor(df\_1\_long$distribution)  
distribution1ToDistribution2Plot <- ggplot()+  
 geom\_ribbon(data = df\_1\_ribbon, aes(factors, ymin = ymin, ymax = ymax),  
 fill = "#4B0092", alpha = .5)+  
 geom\_line(data = df\_1\_long, aes(x=factors, y = val, color = distribution ),  
 alpha = .6, linewidth = 2)+  
 geom\_segment(aes(x = factors, xend= factors,  
 y = ymin, yend = ymax),  
 data = df\_max\_diff,linewidth = 1, color = "black",  
 arrow = arrow(length = unit(0.03, "npc"),ends = "both"), show.legend = F)+  
 labs(x="x", y = "Cumulative Frequency")+  
 guides(segment = "none")+  
 scale\_color\_manual(name='',  
 labels = c("Distribution 1", "Distribution 2"),  
 values=colors)+  
   
 theme\_classic() +  
 ggtitle("")  
distribution1ToDistribution2Plot



The function of absolute differences



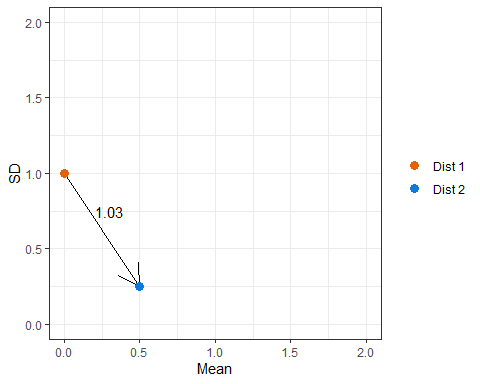
EMD is equal to the indefinite integral of the absolute difference function (in this case, EMD is equal to 0.7217111)

### 3.3.2 Gaussian Case

If we are given two 2D multivariate normal distributions with means and covariances we can compute and analytical second order (p = 2) Wasserstein distance with where is the Bures metric on positive-definite matrices. The Bures metric is defined as . is the [Fidelity Function]<https://en.wikipedia.org/wiki/Fidelity_of_quantum_states> defined as

in the Univariate normal case, we can simplify the equation to . in the above example that would be $((0-1)2+(.5-.25)2) = $ 1.0307764. let’s check with the transport::wasserstein1D package: 0.893843. That is close enough because the inverse CDF method is numerically approximated and we are computing it on a discretized space, which both introduce some level of approximation. We can actually interpret this calculation as the euclidean distance of the paramaeters in 2D, axis 1 = mean and axis 2 = standard deviation

df\_params <- data.frame(mean = c(0,.5), sd = c(1,.25), colors = as.factor(rev(colors)))  
  
segments <- data.frame(startx = df\_params$mean[1],endx = df\_params$mean[2],  
 starty = df\_params$sd[1], endy = df\_params$sd[2])  
ggplot()+  
 geom\_segment(aes(x = startx, xend = endx, y = starty, yend = endy), data = segments,  
 arrow = arrow(ends = "last"))+  
 geom\_point( aes(x = mean, y = sd,color = colors),df\_params, size = 3)+  
   
 scale\_x\_continuous(limits = c(0,2))+  
 scale\_y\_continuous(limits = c(0,2))+  
 scale\_color\_manual(values = colors,  
 labels = c("Dist 1", "Dist 2"))+  
 annotate("text",x=.3,y=.75,label = 1.03)+  
 labs(x="Mean",  
 y="SD")+  
   
 theme\_bw()+  
theme(legend.title = element\_blank())



## 3.4 Cost Matrix Changes

In the Wasserstein formulation above, we find the pairwise distances from to proportional to p. In the linear programming step we make two changes.

1. We modify our cost matrix C. C is generally defined as which is the p-th vector norm.
2. We modify our to be

These two changes give us the general computational formula for the wasserstein metric to the p-th order.

## 3.5 Kantorovich-Rubenstein duality

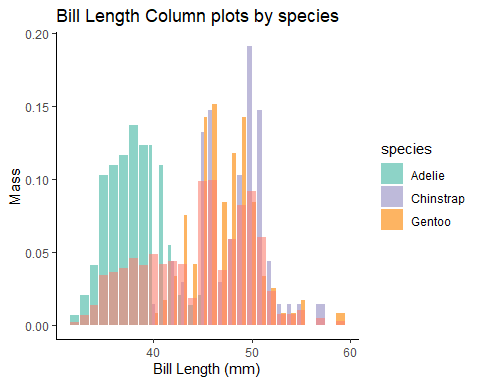
A property of the wasserstein definition is that because we are optimizing over a minimum, there can be a “dual” that optimizes over a complementary optimum. For the wasserstein metric, the “dual” is known as the “Kantrorovich-Rubenstein duality” and is mostly used in Wasserstein General Adversarial Networks (WGAN). The formula is

where and are a coupled joint probability, and are equal to P and Q from our wasserstein definition above, and c is your cost over (x,y). This effectively swaps the constraints and objective in our linear program. The reason this is used in GANs is because they are commonly operating within a generated cost (from a prompt, or input) and need to map that onto a distribution to generate an image, rather than our example where we try to map a distribution onto a cost. \*note, they train WGANs, in part, using the standard formulation.

## 3.6 Wasserstein Barycenter

Let’s reframe the initial problem. Instead of asking “how far is Adelie penguin bill length from Gentoo penguin bill length?” to “what distribution best represents all three penguins”. This could be useful for assessing the average conditions, resource benefits, phenology, etc of the sampled penguins. The standard way of performing this would involve taking a simple average. The average (in black) is added to our discrete graph below.

colors <- brewer.pal(9,"Set3")[c(1,3,6, 4)]  
ave\_pen\_disc <- pen\_disc %>%  
 group\_by(bin)%>%  
 summarize(avrel = sum(rel)/3)  
 discrete\_plot+  
 geom\_col(aes(x = bin, y = avrel), data = ave\_pen\_disc, alpha = .6, fill = colors[4])+  
 theme\_classic()

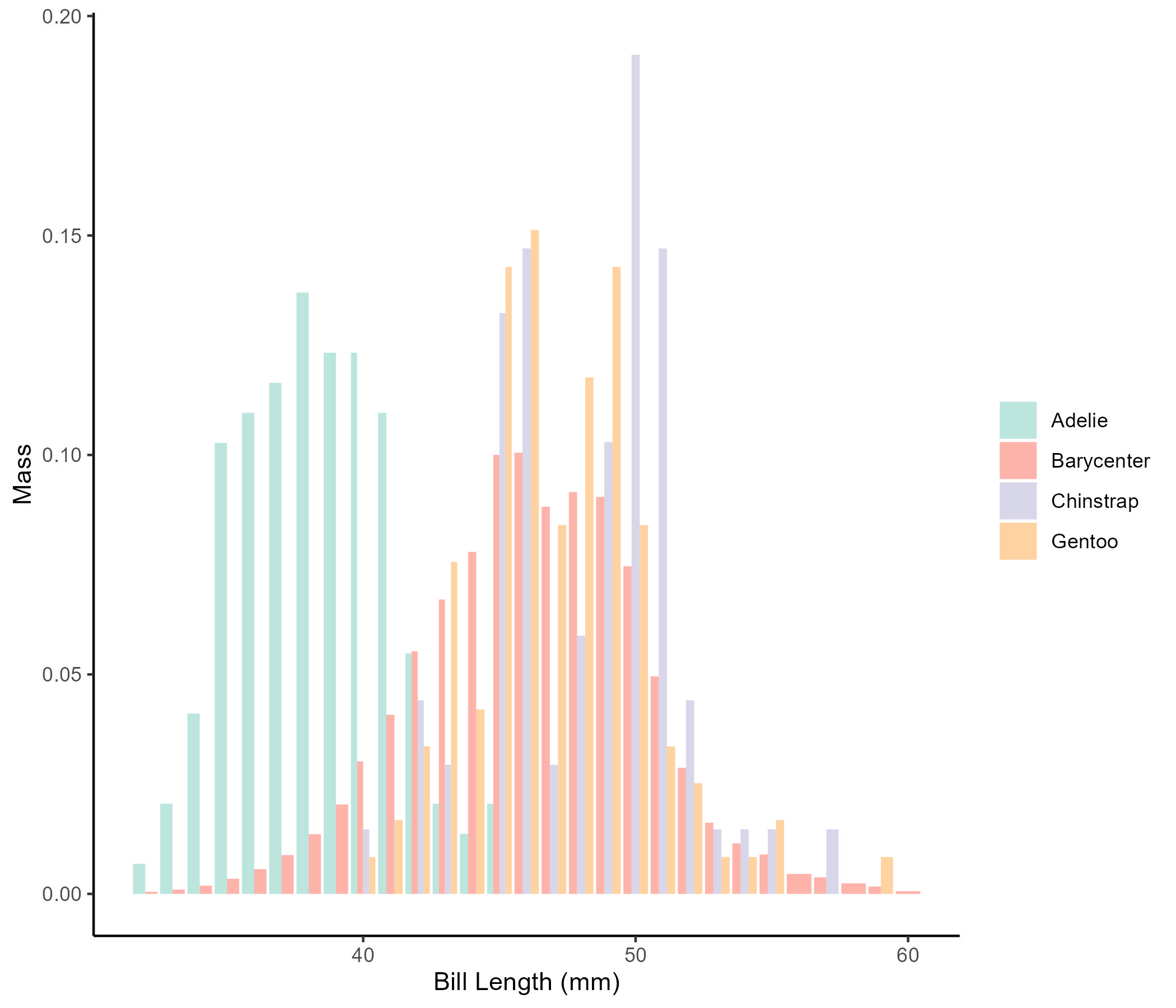


As we can see, it’s a simple average. However, it’s safe for us to qualitatively say that the simple average distribution does not represent the individual distributions it’s trying to describe. It’s only smoothing the total distribution. This means that we are going to integrate the effects of species or islands as unaccounted for random effects. The wasserstein barycenter attempts to correct this problem by computing the distribution that minimizes the wasserstein metric to all of the target distributions. This computes a representative distribution that requires the least amount of effort to transform into the target distributions.

Here is what an updated plot looks like with the wasserstein barycenter

## Warning: package 'einsum' was built under R version 4.3.1

knitr::include\_graphics("Wass\_barycenter.jpg")



## 3.7 Wasserstein Geodesic Principal Components Analysis (GPCA)

Understanding Wasserstein GPCA we need to understand multiple layers of advanced mathematics. I will refer to [Geodesic Regression]<https://towardsdatascience.com/geodesic-regression-d0334de2d9d8> and [Principal Geodesic Analysis]<https://towardsdatascience.com/principal-geodesic-analysis-2ec7ad1b2679> for you to read and get up to speed on how to conceptualize the foundations.