BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Distributions, Parameters and Likelihood

- Univariate
  - Normal
  - Binomial
  - Beta
  - Skew Normal
- Multivariate
  - Normal

Distributions are not data, they're models of data, *(like regression and classification)*. And remember George Box: "all models are wrong, but some are useful".

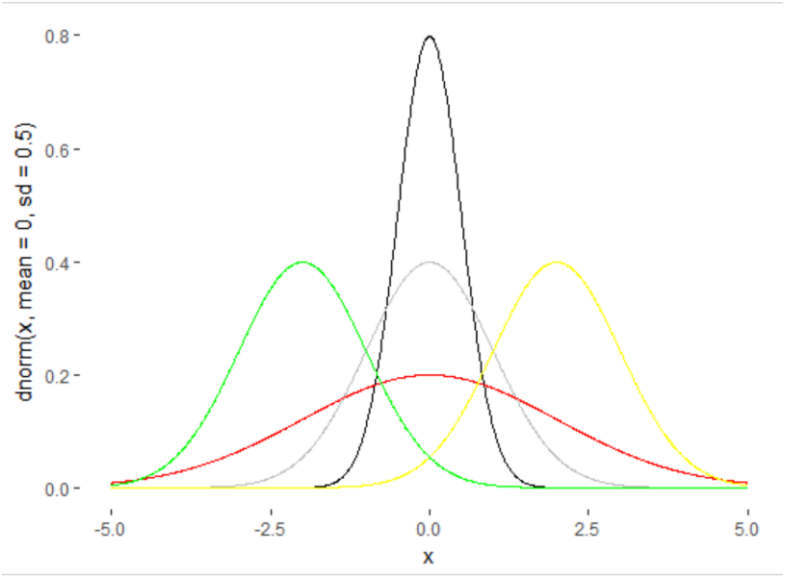
Distributions are foundational for higher level models. The quality of your models *(even the non-parametric – NP, Machine Learning - ML ones)* depend on distributions.

We need to know the distributions of model ***input, parameters and output***. This is where model quality and assurance about that quality begins.

We'll review a few of the more useful distributions here, but there are MANY choices to fitting distributions to your data and parameters, from simple uniform to Dirichlet *(level 3 projects course: Bayesian modeling of NP data)*

Normal Distribution

$N(\mu, \sigma^2)$



Density Function *(PDF)*

dnorm

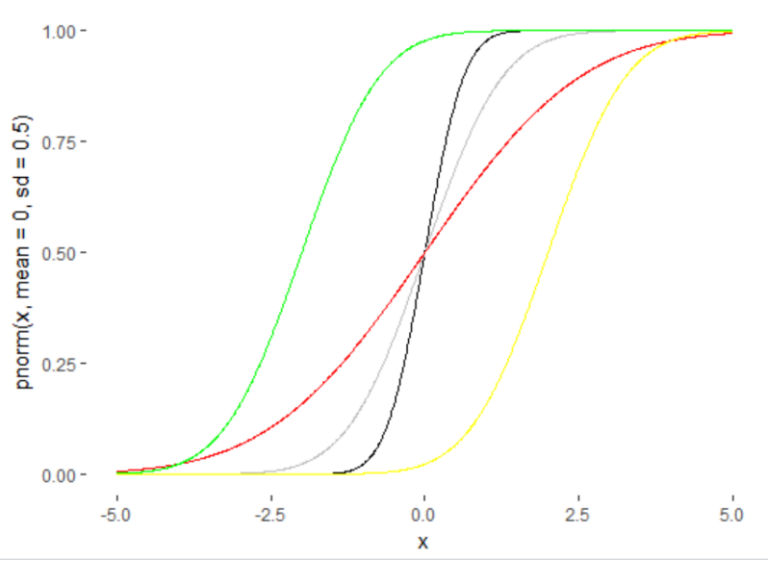
$$\frac{1}{\sqrt{2\pi} * \sigma} \; e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

```
funNormPDF <- function(x, mean, sd) {
1/(sqrt(2*pi)*sd)*exp((-(xmean)^2)/(2*(sd^2)))
}
```

moments

mean(x)
sd(x)
skewness(x)
kurtosis(x)

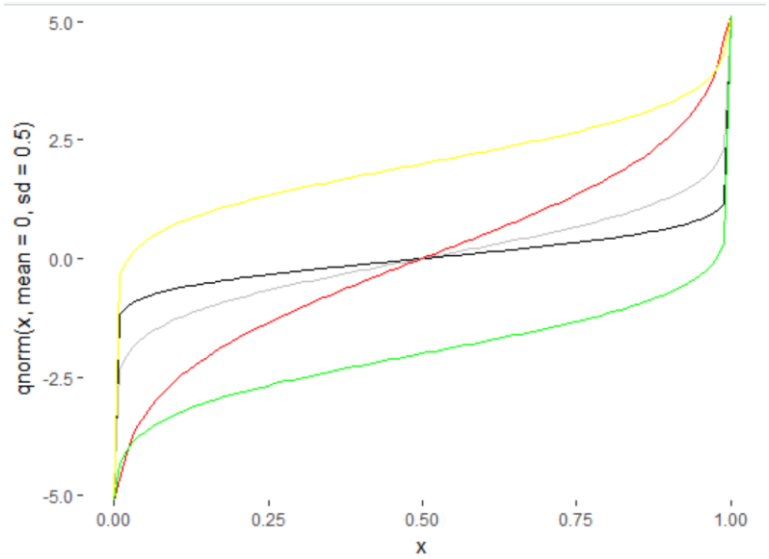
```
> dfDenNorm <- data.frame(x = seq(-5, 5, by = .01))
> p1 <- ggplot(dfDenNorm) +
+   geom_line(aes(x,y= dnorm(x, mean = 0, sd = .5))) +
+   geom_line(aes(x,y= dnorm(x, mean = 0, sd = 1)), color = "gray") +
+   geom_line(aes(x,y= dnorm(x, mean = 0, sd = 2)), color = "red") +
+   geom_line(aes(x,y= dnorm(x, mean = 2, sd = 1)), color = "yellow") +
+   geom_line(aes(x,y= dnorm(x, mean = -2, sd = 1)), color = "green") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```



Cumulative Density *(CDF)*

pnorm

```
> dfPNorm <- data.frame(x = seq(-5, 5, by = .01))
> # x axis becomes x values - y is cummulative prob
> p1 <- ggplot(dfPNorm) +
+   geom_line(aes(x,y= pnorm(x, mean = 0, sd = .5))) +
+   geom_line(aes(x,y= pnorm(x, mean = 0, sd = 1)), color = "gray") +
+   geom_line(aes(x,y= pnorm(x, mean = 0, sd = 2)), color = "red") +
+   geom_line(aes(x,y= pnorm(x, mean = 2, sd = 1)), color = "yellow") +
+   geom_line(aes(x,y= pnorm(x, mean = -2, sd = 1)), color = "green") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```



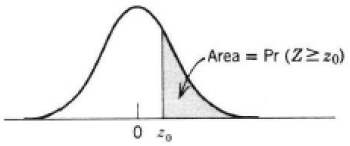
Quantile

qnorm

```
> dfQNorm <- data.frame(x = seq(0, 1, by = .01))
> # x axis becomes x quantiles
> p1 <- ggplot(dfQNorm) +
+   geom_line(aes(x,y= qnorm(x, mean = 0, sd = .5))) +
+   geom_line(aes(x,y= qnorm(x, mean = 0, sd = 1)), color = "gray") +
+   geom_line(aes(x,y= qnorm(x, mean = 0, sd = 2)), color = "red") +
+   geom_line(aes(x,y= qnorm(x, mean = 2, sd = 1)), color = "yellow") +
+   geom_line(aes(x,y= qnorm(x, mean = -2, sd = 1)), color = "green") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```

BAUER
COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

Looking back: Remember the z tables?



Standard Normal, Cumulative Probability in Right-Hand Tail (For Negative Values of z, Areas are Found by Symmetry)

	NEXT DECIMAL PLACE OF z_0									
z_0	0	1	2	3	4	5	6	7	8	9
0.0	.500	.496	.492	.488	.484	.480	.476	.472	.468	.464
0.1	.460	.456	.452	.448	.444	.440	.436	.433	.429	.425
0.2	.421	.417	.413	.409	.405	.401	.397	.394	.390	.386
0.3	.382	.378	.374	.371	.367	.363	.359	.356	.352	.348
0.4	.345	.341	.337	.334	.330	.326	.323	.319	.316	.312
0.5	.309	.305	.302	.298	.295	.291	.288	.284	.281	.278
0.6	.274	.271	.268	.264	.261	.258	.255	.251	.248	.245
0.7	.242	.239	.236	.233	.230	.227	.224	.221	.218	.215
0.8	.212	.209	.206	.203	.200	.198	.195	.192	.189	.187
0.9	.184	.181	.179	.176	.174	.171	.169	.166	.164	.161
1.0	.159	.156	.154	.152	.149	.147	.145	.142	.140	.138
1.1	.136	.133	.131	.129	.127	.125	.123	.121	.119	.117
1.2	.115	.113	.111	.109	.107	.106	.104	.102	.100	.099
1.3	.097	.095	.093	.092	.090	.089	.087	.085	.084	.082
1.4	.081	.079	.078	.076	.075	.074	.072	.071	.069	.068
1.5	.067	.066	.064	.063	.062	.061	.059	.058	.057	.056
1.6	.055	.054	.053	.052	.051	.049	.048	.047	.046	.046
1.7	.045	.044	.043	.042	.041	.040	.039	.038	.038	.037
1.8	.036	.035	.034	.034	.033	.032	.031	.031	.030	.029
1.9	.029	.028	.027	.027	.026	.026	.025	.024	.024	.023
2.0	.023	.022	.022	.021	.021	.020	.020	.019	.019	.018
2.1	.018	.017	.017	.017	.016	.016	.015	.015	.015	.014
2.2	.014	.014	.013	.013	.013	.012	.012	.012	.011	.011
2.3	.011	.010	.010	.010	.010	.009	.009	.009	.009	.008
2.4	.008	.008	.008	.008	.007	.007	.007	.007	.007	.006
2.5	.006	.006	.006	.006	.006	.005	.005	.005	.005	.005
2.6	.005	.005	.004	.004	.004	.004	.004	.004	.004	.004
2.7	.003	.003	.003	.003	.003	.003	.003	.003	.003	.003
2.8	.003	.002	.002	.002	.002	.002	.002	.002	.002	.002
2.9	.002	.002	.002	.002	.002	.002	.002	.001	.001	.001

The inside cover of your statistics book usually has a standard normal distribution table where each z value is the number of standard deviations away from the mean.

For general normal distributions, we converted the values to z values by standardizing: $Z = \frac{X - \mu}{\sigma}$

To get the probability a random values exceeding a standardized z value (standard deviation) you can look it up in the table, or just type pnorm.

```
> round(1 - pnorm(q = 1.5, mean = 0, sd = 1),3)
[1] 0.067
```

pnorm returns the probability from -∞ to q *(q = quantile – i.e., the values are sorted)*

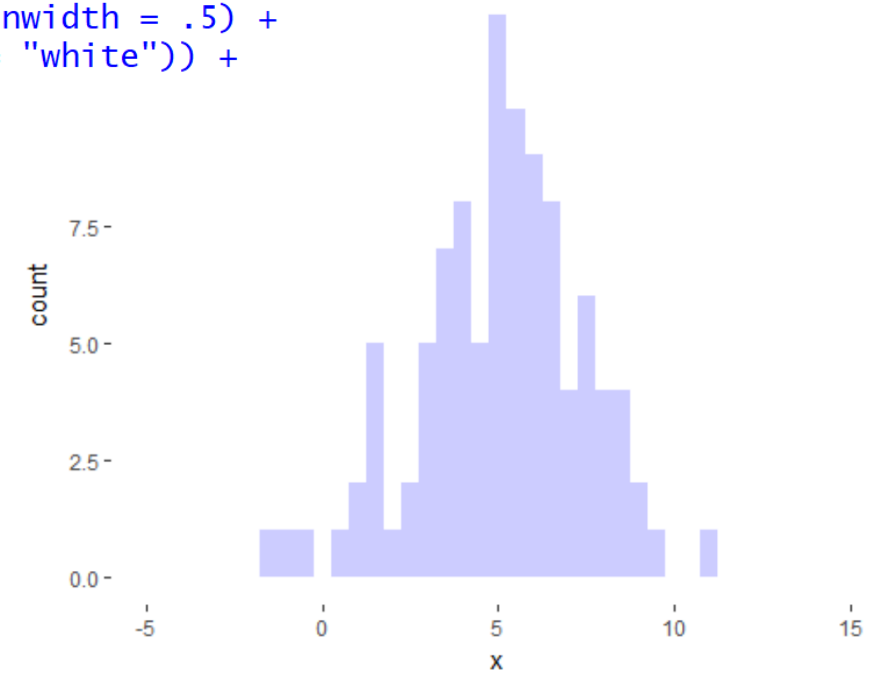
dnorm(x, mean = 0, sd = 1, log = FALSE) returns density function
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE) returns probability
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE) returns quantile
rnorm(n, mean = 0, sd = 1) returns n random values based on distribution parameters

We put this is a dataframe called dfRandomNorm and name the column x

generates 100 random values following a distribution with mean of 5, and standard deviation of 2

```
> dfRandNorm <- data.frame(x = rnorm(100, mean = 5, sd = 2))
> p <- ggplot(dfRandNorm, aes(x)) +
+   geom_histogram(alpha = .2, fill = 'blue', binwidth = .5) +
+   theme(panel.background = element_rect(fill = "white")) +
+   xlim( -5, 15)
> p
```

We're plotting out the counts from rnorm using a histogram

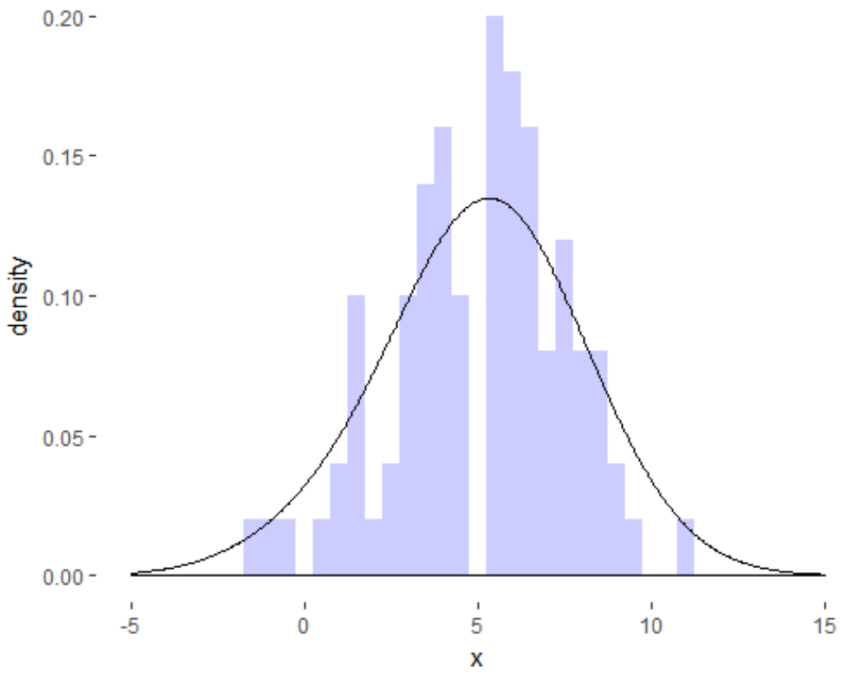


Or, we can just use a ..density.. function built into ggplot. This is a kernel smoothing function and doesn't return a true density function.

```
> p <- ggplot(dfRandNorm, aes(x, ..density..)) + geom_histogram(alpha = .2,
+     fill = 'blue', binwidth = .5) +
+     geom_density(bw = 2) +
+     xlim( -5, 15) + ylim(0, .20) +
+   theme(panel.background = element_rect(fill = "white"))
> p
```

And we can also add a smoothed density line. Again, this is not a true normal density function, it's a kernel smoothing technique. Kernel density estimation is a non-parametric approach, with bandwidth (bw) a parameter.

So, this just estimated a normal probability density function (pdf).



Many ways to get probability estimate on a normal:

1. Calculate z-score
2. Integrate the density function using dnorm
3. Integrate using your own density function.

```
> x <- 8
> # manually calcuate z score
> zScore <- (x - 5)/2
> round(1 - pnorm(q = zScore, mean = 0, sd = 1),3)
[1] 0.067
>
> # or you can use a density fucntion and integrate the area
> f <- integrate(dnorm, mean = 5, sd = 2, lower = 8, upper = Inf)
>
> # check
> round(f$value,3)
[1] 0.067
>
> # or you can just calculate using the formula
> manNorm <- function(x, mean, sd) {1/(sqrt(2*pi)*sd)*exp((-(x-mean)^2)/(2*(sd^2)))}
> f2 <- integrate(manNorm, lower = 8, upper = Inf, sd=2, mean=5)
> # check
> round(f2$value, 3)
[1] 0.067
```

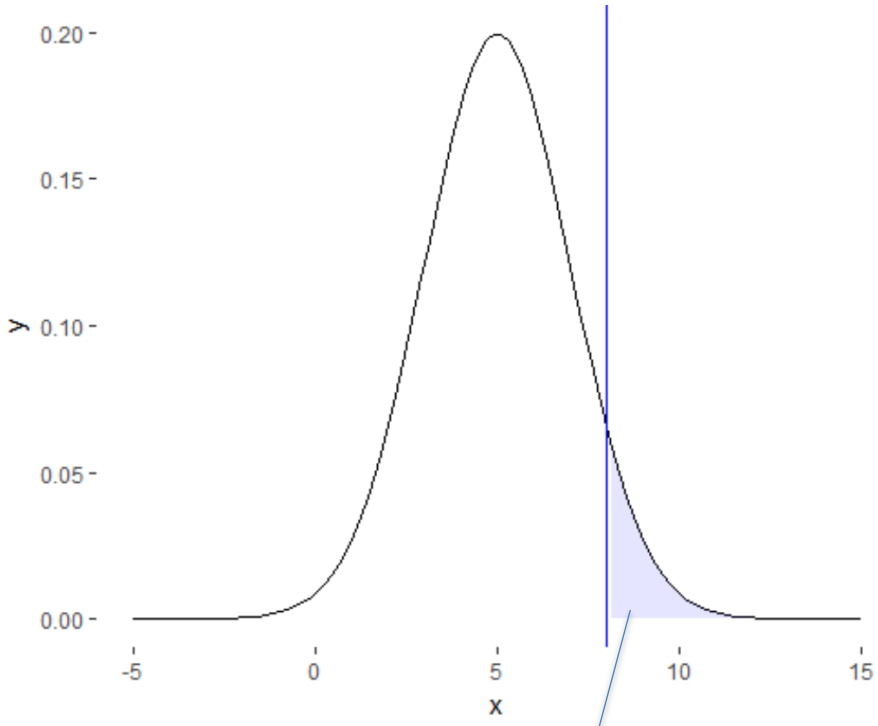
$$\frac{1}{\sqrt{2\pi} * \sigma} \; e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

This is the normal density function and we'll be using this extensively

```
> p
> # ------------------ let's take a more visual approach using a true density function
> # note again that the density values will be different from a kernel density estimate
>
>
> x = seq(from = -5, to  = 15, length.out = 100)
> dfNormX <- data.frame(x, y = dnorm(x, mean = 5, sd = 2))
> p <- ggplot(dfNormX, aes(x,y)) + geom_line() +
+   theme(panel.background = element_rect(fill = "white")) +
+   xlim( -5, 15) + ylim(0, .2)
> p
>
> p <- p + geom_vline(xintercept = 8, color = "blue")
> # set up data for mapping and show probability
> ce <- dplyr::filter(dfNormX, x > 8) %>% arrange(x)
> ce2 <- rbind( c( min( ce$x), 0), + ce, + c( max( ce$x), 0))
> p <- p + geom_polygon( data = ce2, aes(x = x, y = y),
+                        fill = 'blue', alpha = 0.1)
> p
```

You don't have to know how to do this polygon stuff, just for visual reinforcement.

Here, we use the dnorm function and then plot the values using geom_line.

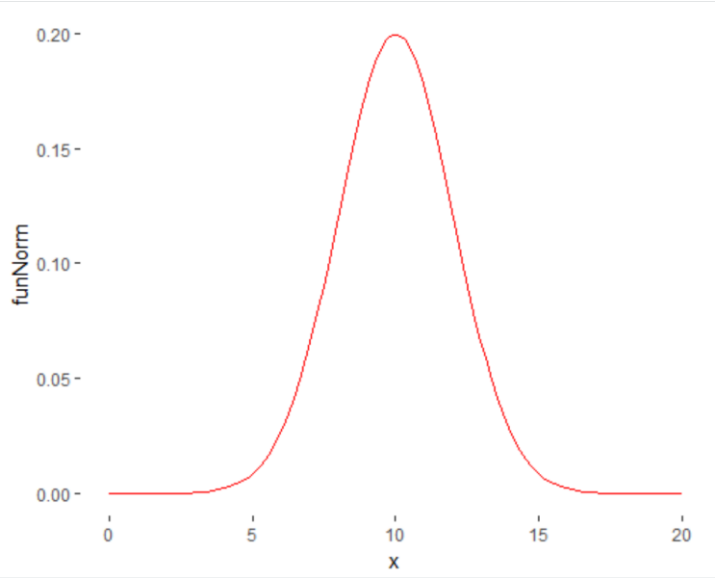


$$\int_{8}^{\infty} \frac{1}{\sqrt{2\pi} * \sigma} e - \frac{(x-\mu)^2}{2\sigma^2} =\sim .067$$

Normal PDF = $\frac{1}{\sqrt{2\pi} * \sigma} \; e - \frac{(x-\mu)^2}{2\sigma^2}$

$= \frac{n}{\sqrt{2\pi} * \sigma} \; exp\left[-\frac{\sum_1^n (x-\mu)^2}{2\sigma^2}\right]$

1/(sqrt(2*pi)*sd)*exp((-(x-mean)^2)/(2*(sd^2)))




```
> funNormPDF <- function(x, mean, sd) {1/(sqrt(2*pi)*sd)*exp((-(x-mean)^2)/(2*(sd^2)))}
>
> x = seq(from = 0, to = 20, length.out = 100)
> dfNormPDF <- data.frame(x=x, funNorm = funNormPDF(x, 10 ,2), dnormPDF = dnorm(x, 10 ,2))
>
> p <- ggplot(data = dfNormPDF, aes(x, funNorm)) + geom_line()
> p <- p +   theme(panel.background = element_rect(fill = "white"))
> p
> p <- p + geom_line(data = dfNormPDF, aes(x, dnormPDF), color = "red")
> p
```

verifying manual results to dnorm, which does the same thing

Normal Likelihood Function

Normal PDF $(x|\mu, \sigma^2) = \frac{n}{\sqrt{2\pi} * \sigma} \; exp\left[-\frac{\sum_1^n (x-\mu)^2}{2\sigma^2}\right]$

Likelihood $L(\mu, \sigma^2 \mid x_{1..n}) = \frac{n}{\sqrt{2\pi} * \sigma} \; exp\left[-\frac{\sum_1^n (x-\mu)^2}{2\sigma^2}\right]$

the PDF becomes a likelihood function when we hold the data constant and vary the parameters (theta$_x$ values). So, instead of telling us the most likely data value, it tells us the most likely parameter value – given the data.

We can simplify this equation (and also improve it's computing performance) by taking the log (see your handouts on log transformations.

We also want to 'turn it around" by multiplying by -1, so we can use r optimization functions (which search for minimums).

Negative Log Likelihood $L(\mu, \sigma^2 \mid x_{1..n}) = -\frac{n}{2}\ln(\sqrt{2\pi} * \sigma) - \frac{1}{2\sigma^2}\sum_{j=1}^n (x_j - \mu)^2$

```
-sum(log(1/(sqrt(2*pi)*sd))-((data-mu)^2)/(2*(sd^2)))
```

```
> # generate data and test theta values
> y <- rnorm(100, mean = 10, sd = 2)
> mean(y)
[1] 9.842579
> sd(y)
[1] 1.979163
>
> NLL <- function(theta, data) {
+   mu <- theta[1]
+   sd <- theta[2]
+   -sum(log(1/(sqrt(2*pi)*sd))-((data-mu)^2)/(2*(sd^2)))
+ }
> out = optim(par=c(9,1), fn=NLL, data = y)
> out$par
[1] 9.842868 1.969398
```

Here, we:

- Generate a random normal distribution,
- Test the mean and sd *(remember it's random, so you don't know what you'll get)*
- Then we call a custom NLL function that "wraps" our negative log likelihood function.
- We use optim to find the minimum value *(it runs through a grid - starting with 9 and 1, until convergence)*.

… as you can see, we get very close

```
> mu <- seq(from = 8, to = 12, by = 1)
> m<- length(mu)
> sd <- seq(from = 1, to = 5, by = 1)
> n<- length(sd)
> LikeMatrix <- matrix( nrow = 0, ncol = 3)
>
> for (i in 1:m) {
+   for(j in 1:n){
+     LikeVal <-  NLL(c(mu[i], sd[j]), y)
+     LikeMatrix <<- rbind(LikeMatrix, c(LikeVal, mu[i], sd[j]))
+   }
+ }
>
> dfLike <- data.frame(LikeMatrix)
> colnames(dfLike)[1] <- 'Like'
> colnames(dfLike)[2] <- 'mu'
> colnames(dfLike)[3] <- 'sigma'
> max.row <- row.names(dfLike)[(which(dfLike$Like ==min(dfLike$Like)))]
> maxRow <- dfLike[max.row,]
> round(maxRow$mu,0)
[1] 10
> round(maxRow$sigma,0)
[1] 2
```

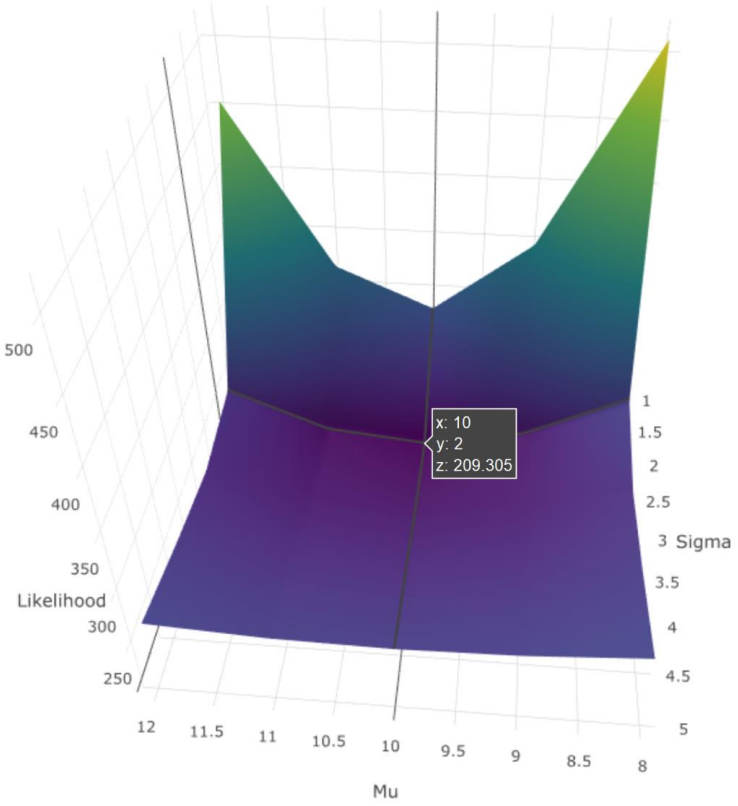
We can do this without optim:

- We set up a grid of mu values and sd values.
- Create a matrix to hold all the values for each combination of parameter values *(mu, sd and likelihood for each combination)*
- Iterate through each parameter combination, using NLL to compute the negative likelihood value (then, we clean up the matrix for later).
- And select the row with the smallest likelihood value, and pull the mu and sigma (theta) values

```
> library(plotly)
>
> tst <- spread(dfLike, sigma, Like)
> rownames(tst) <- tst$mu
> tst$mu <- NULL
> tst2 <- as.matrix(tst)
> tst2 <- t(tst2)
>
>
> p <- plot_ly(x = mu, y = sd, z = tst2) %>% add_surface() %>%
+   layout(
+     title = "Normal Distribution Likelihood",
+     scene = list(
+       xaxis = list(title = "Mu"),
+       yaxis = list(title = "Sigma"),
+       zaxis = list(title = "Likelihood")
+     ))
> p
```

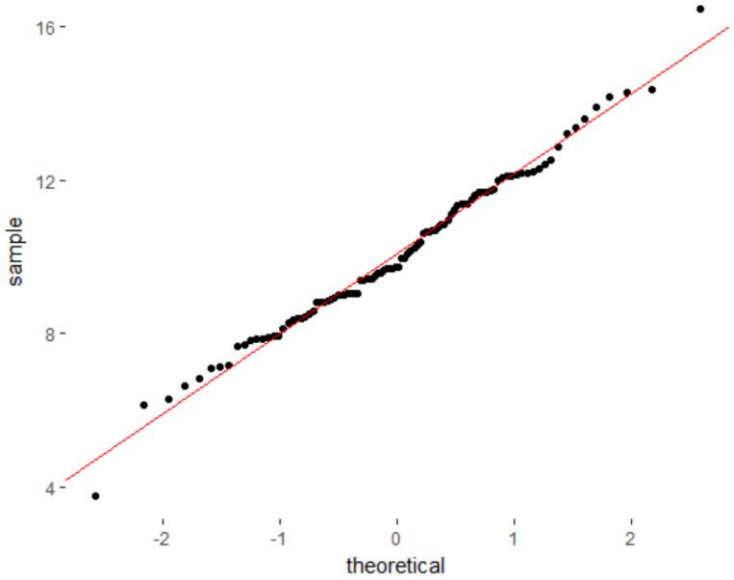
We can also visualize this the likelihood values with mu and sigma on the axes.

Hopefully, this helps build some intuition.



Normal QQ

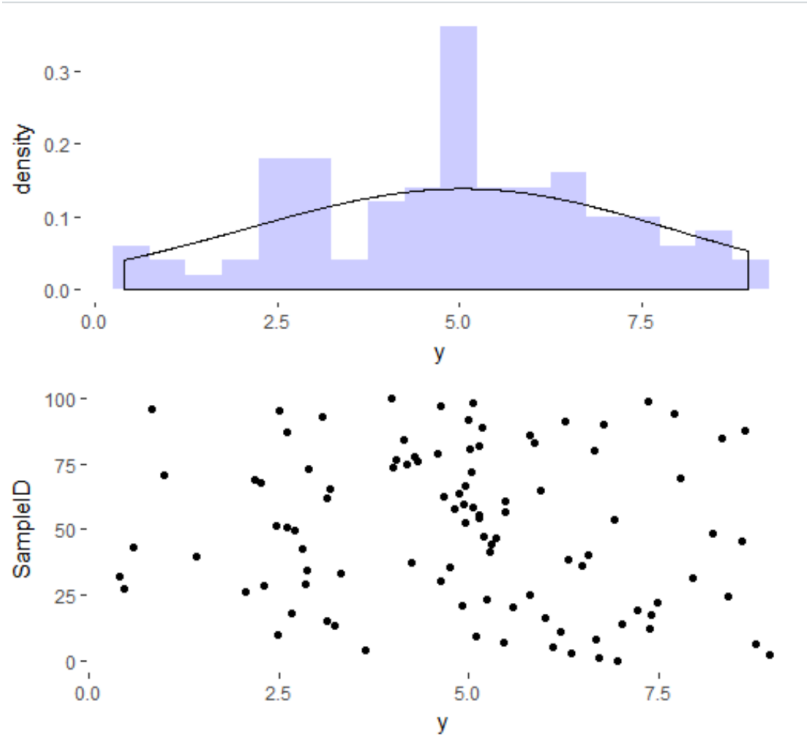
```
> mean(y)
[1] 10.08566
> sd(y)
[1] 2.084907
> skewness(y)
[1] 0.1518809
> kurtosis(y)
[1] 3.391001
>
>
> p1 <- ggplot(data = data.frame(y), aes(sample = y)) +
+       geom_qq() +
+       geom_abline(intercept = mean(y), slope = sd(y), color = "red") +
+       theme(panel.background = element_rect(fill = "white"))
> p1
```



Finally, we add a few metrics. We'll get into skewness and kurtosis more later – for now, be aware that a normal distribution will be around 0 for skewness and around 3 for kurtosis.

A QQ plot sorts your sample data into quantiles *(ranked in ascending order)*, and compares that to a theoretical distribution – in this a standardized normal *(back to the z tables☺)*. A normal distribution should follow a straight line *($y = z\sigma + \mu$)*. The sample data is mapped and you want this to be close to the line *(like the above)*.

Exercise: Normal



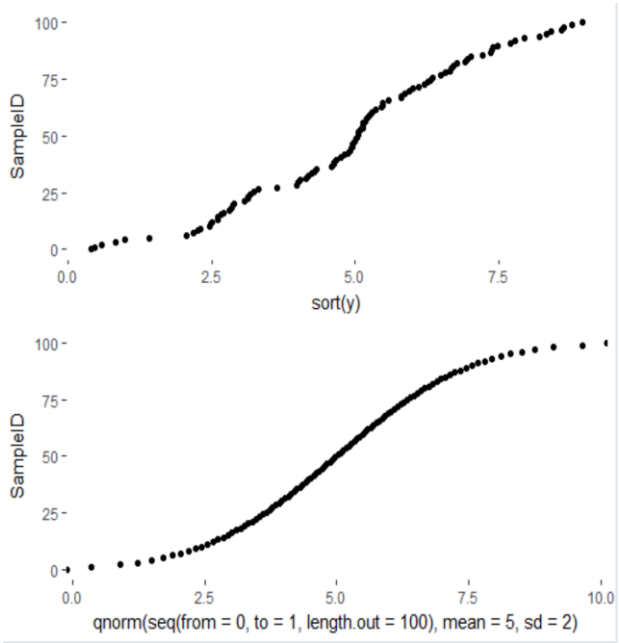
Create a dataframe with a column for SampleID (use seq to number the rows 1:100) and a column with random variables (use rnorm with a mean = 5 and sd = 2).

Check mean, standard deviation, skewness and kurtosis to make sure it fits a normal distribution.

Now, create a plot with a density histogram (use the ..density.. function in ggplot) and a density line (use geom_density).

Create a scatter plot of the data using geom_point and compare.

```
> library(gridExtra)
>
> dfClassExercise1 <- data.frame(
+   SampleID = seq(from = 0, to= 100,length.out = 100),
+   y = rnorm(100, mean = 5, sd = 2))
>
> mean(dfClassExercise1$y)
[1] 4.921297
> sd(dfClassExercise1$y)
[1] 2.027002
> skewness(dfClassExercise1$y)
[1] -0.145546
> kurtosis(dfClassExercise1$y)
[1] 2.508552
>
> p1 <- ggplot(dfClassExercise1, aes(y, ..density..)) +
+   geom_histogram(alpha = .2,
+   fill = 'blue', binwidth = .5) +
+   geom_density(bw = 2) +
+   theme(panel.background = element_rect(fill = "white"))
> p2 <- ggplot(data = dfClassExercise1, aes(y, SampleID)) +
+   geom_point() +
+   theme(panel.background = element_rect(fill = "white"))
> grid.arrange(p1, p2, ncol=1)
```

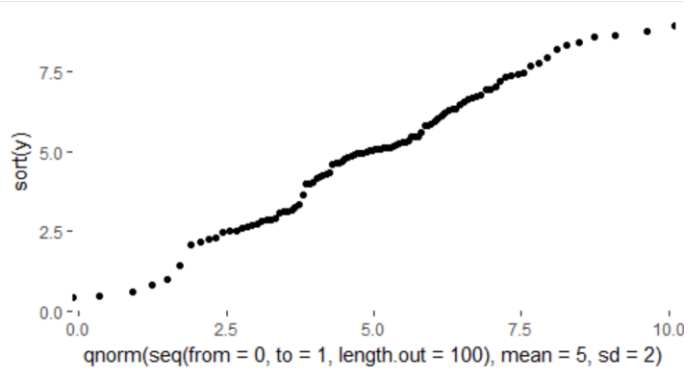


Now take your scatter plot and replot it using sorted data – *the sort(y) function will be easiest.* Notice the shape of the data takes the symmetric form.

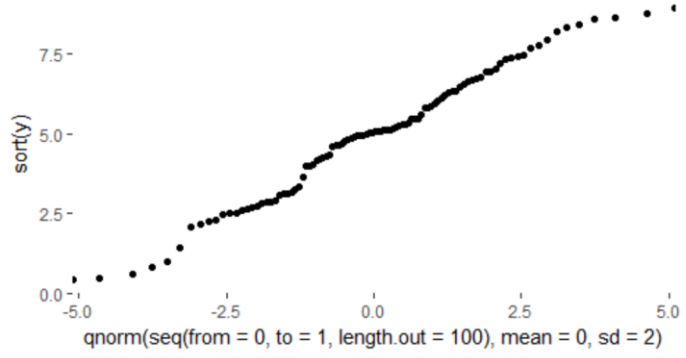
Create another plot using qnorm with the same number of samples (seq 1:100), and the same mean and sd.

Compare the plots.

```
> p3 <- ggplot(data = dfClassExercise1, aes(sort(y), SampleID)) +
+   geom_point() +
+   theme(panel.background = element_rect(fill = "white"))
> p4 <- ggplot(dfClassExercise1) +
+   geom_point(aes(x = qnorm(seq(from = 0, to= 1,length.out = 100),
+                            mean = 5, sd = 2), y = SampleID)) +
+   theme(panel.background = element_rect(fill = "white"))
> grid.arrange(p3, p4, ncol=1)
```

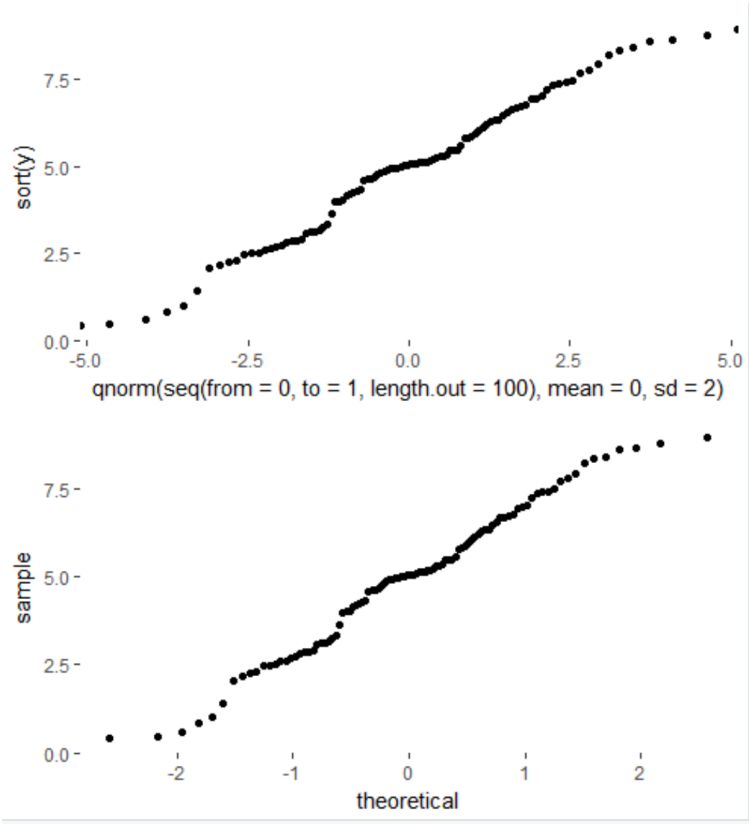


Now combine the plots with your sorted y values on the y axis and your qnorm values on the x axis.



Now standardize the x axis by changing the mean value to 0.

```
> p5 <- ggplot(dfClassExercise1) +
+   geom_point(aes(x = qnorm(seq(from = 0, to= 1,length.out = 100),
+                            mean = 5, sd = 2), y = sort(y))) +
+   theme(panel.background = element_rect(fill = "white"))
> p6 <- ggplot(dfClassExercise1) +
+   geom_point(aes(x = qnorm(seq(from = 0, to= 1,length.out = 100),
+                            mean = 0, sd = 2), y = sort(y))) +
+   theme(panel.background = element_rect(fill = "white"))
> grid.arrange(p5, p6, ncol=1)
```



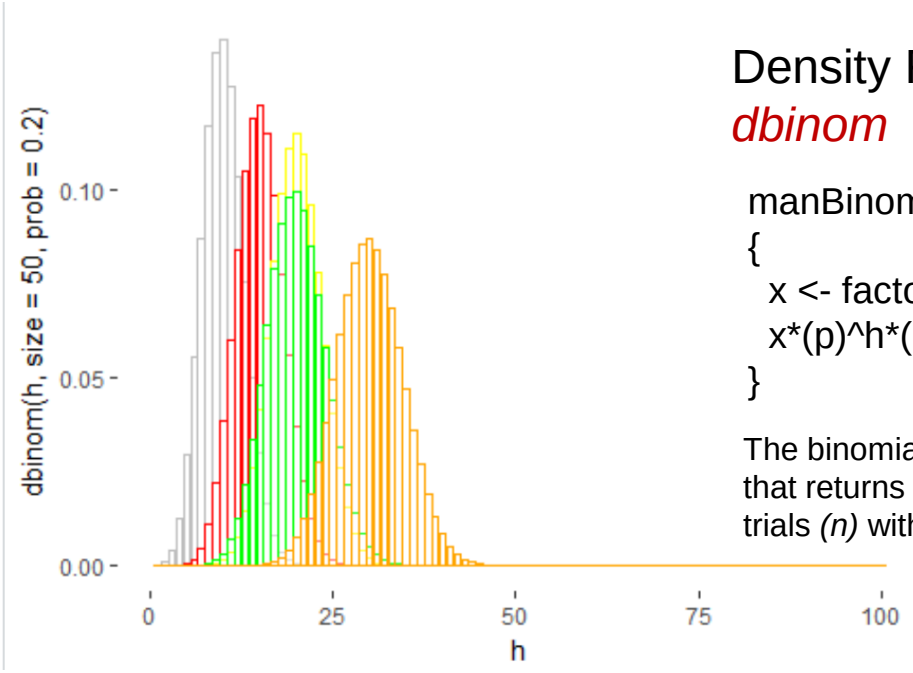
Now create one more plot (the easy one) and just use the geom_qq function and your y values – aes(sample = y).

How do these plots compare?

```
> p7 <- ggplot(dfClassExercise1) +
+   geom_point(aes(x = qnorm(seq(from = 0, to= 1,length.out = 100),
+                            mean = 0, sd = 2), y = sort(y))) +
+   theme(panel.background = element_rect(fill = "white"))
> p8 <- ggplot(dfClassExercise1, aes(sample = y)) +
+   geom_qq() +
+   theme(panel.background = element_rect(fill = "white"))
> grid.arrange(p7, p8, ncol=1)
```

Binomial Distribution

$B(n, p, h)$



Density Function *(PDF)*

$$\frac{n!}{h!(n-h)!} p^h (1-p)^{n-h}$$

dbinom

```
manBinom <- function(n, p, h)
{
  x <- factorial(n)/(factorial(h)*factorial(n-h))
  x*(p)^h*(1-p)^(n-h)
}
```

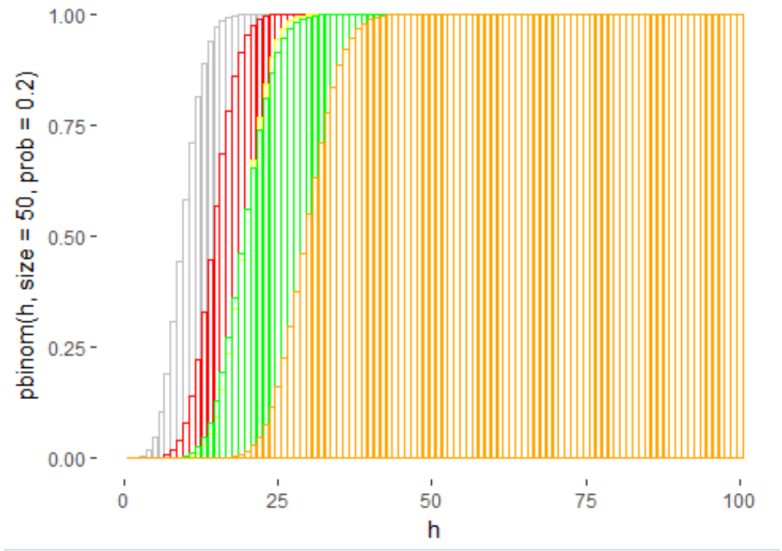
The binomial is a discrete function. The density is a PMF *(not a PDF)*, so it that returns the probability of a number of successes *(h)*, for a given set of trials *(n)* with a probability of success *(p)*.

Intuition. Note how the distribution moves to the right when either:

- *The probability of success is increased, or*
- *The number of trials is increased*

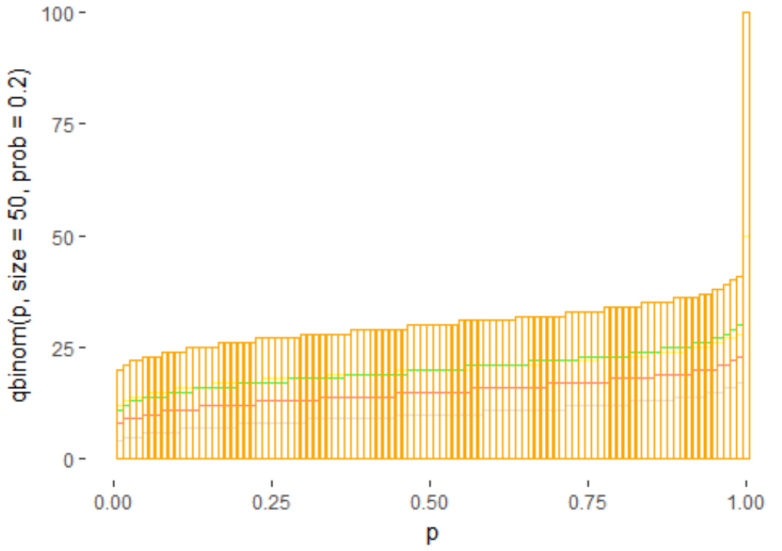
Compare the yellow (40% success prob and for 50 trials aligns with 20% success for 100 trials)

```
> # the binomial distribution - intro
> # h number of successes
> h <- seq(from = 1, to = 100, by = 1)
> # n number of trials
> n <- seq(from = 1, to = 100, by = 1)
> # p probability of success
> p <- seq(from = .01, to = 1, by = .01)
>
> dfData <- data.frame(h, n, p)
>
> # dbinom gives us the probablity of a # of successes
> # based on the number of trials and probablity
>
> p1 <- ggplot( data = dfData) +
+   geom_bar(mapping = aes(x = h, y = dbinom(h, size = 50, prob = .2)),
+            stat = 'identity', alpha = .2, color = "gray", fill = "white") +
+   geom_bar(mapping = aes(x = h, y = dbinom(h, size = 50, prob = .3)),
+            stat = 'identity', alpha = .2, color = "red", fill = "white") +
+   geom_bar(mapping = aes(x = h, y = dbinom(h, size = 50, prob = .4)),
+            stat = 'identity', alpha = .2, color = "yellow", fill = "white") +
+   geom_bar(mapping = aes(x = h, y = dbinom(h, size = 100, prob = .2)),
+            stat = 'identity', alpha = .2, color = "green", fill = "white") +
+   geom_bar(mapping = aes(x = h, y = dbinom(h, size = 100, prob = .3)),
+            stat = 'identity', alpha = .2, color = "orange", fill = "white") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```



Cumulative Density *(CDF)*
pbinom

```
> # pbinom gives us the probablity of a # of successes or less
> # based on the number of trials and probablity
>
> p1 <- ggplot( data = dfData) +
+   geom_bar(mapping = aes(x = h, y = pbinom(h, size = 50, prob = .2)),
+            stat = 'identity', alpha = .2, color = "gray", fill = "white") +
+   geom_bar(mapping = aes(x = h, y = pbinom(h, size = 50, prob = .3)),
+            stat = 'identity', alpha = .2, color = "red", fill = "white") +
+   geom_bar(mapping = aes(x = h, y = pbinom(h, size = 50, prob = .4)),
+            stat = 'identity', alpha = .2, color = "yellow", fill = "white") +
+   geom_bar(mapping = aes(x = h, y = pbinom(h, size = 100, prob = .2)),
+            stat = 'identity', alpha = .2, color = "green", fill = "white") +
+   geom_bar(mapping = aes(x = h, y = pbinom(h, size = 100, prob = .3)),
+            stat = 'identity', alpha = .2, color = "orange", fill = "white") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```



Quantile
qbinom

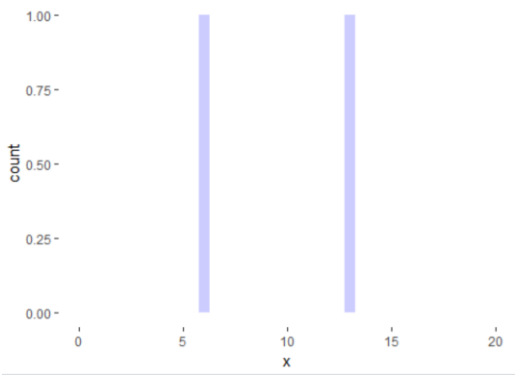
```
> p1 <- ggplot( data = dfData) +
+   geom_bar(mapping = aes(x = p, y = qbinom(p, size = 50, prob = .2)),
+            stat = 'identity', alpha = .2, color = "gray", fill = "white") +
+   geom_bar(mapping = aes(x = p, y = qbinom(p, size = 50, prob = .3)),
+            stat = 'identity', alpha = .2, color = "red", fill = "white") +
+   geom_bar(mapping = aes(x = p, y = qbinom(p, size = 50, prob = .4)),
+            stat = 'identity', alpha = .2, color = "yellow", fill = "white") +
+   geom_bar(mapping = aes(x = p, y = qbinom(p, size = 100, prob = .2)),
+            stat = 'identity', alpha = .2, color = "green", fill = "white") +
+   geom_bar(mapping = aes(x = p, y = qbinom(p, size = 100, prob = .3)),
+            stat = 'identity', alpha = .2, color = "orange", fill = "white") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```

Binomial *(functions same as normal)*:

dbinom(x, size, prob, log = FALSE) returns density function
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE) returns probability
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE) returns quantile
rbinom(n, size, prob) returns n random values based on distribution parameters

Looking at rbinom, let's conduct 2 random trials of 20 flips of a fair coin. (bias of .5).

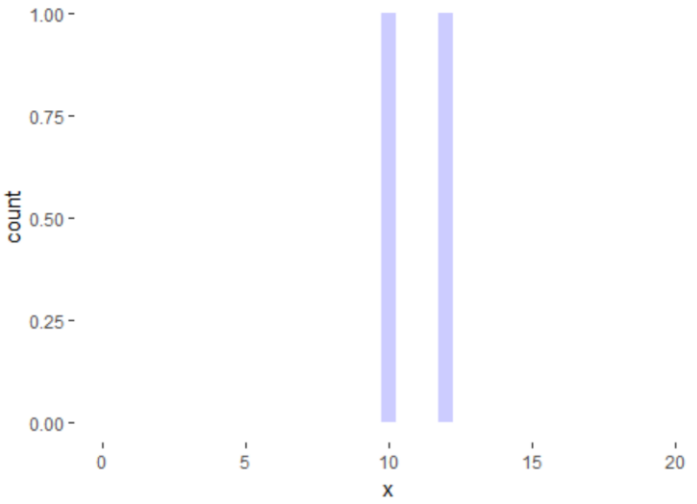
```
> # looking at the binomail functions:
> dfRandBinom <- data.frame(x = rbinom(n=2,size = 20 , prob = .5))
> p1 <- ggplot(dfRandBinom, aes(x)) +
+   geom_histogram(alpha = .2, fill = 'blue', binwidth = .5) +
+   theme(panel.background = element_rect(fill = "white")) +
+   xlim(0, 20)
> p1
```



So, here we got 6 heads in one trial, and 13 head in another.

What if the coin was biased (weighted to land heads 60% of the time)

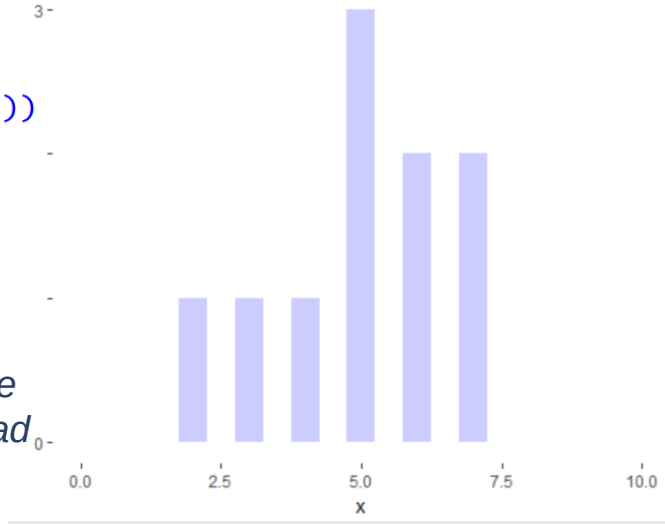
```
> # what if the coin was biased (trick coin) and came up heads 60% of the tim
> dfRandBinom <- data.frame(x = rbinom(n=2,size = 20 , prob = .6))
> p1 <- ggplot(dfRandBinom, aes(x)) +
+   geom_histogram(alpha = .2, fill = 'blue', binwidth = .5) +
+   theme(panel.background = element_rect(fill = "white")) +
+   xlim(0, 20)
> p1
```



This time, we got 10 once and 11 once…

Next, we generate 10 trials of 10 flips (just keeping the numbers easy to track as we walk through the distribution)

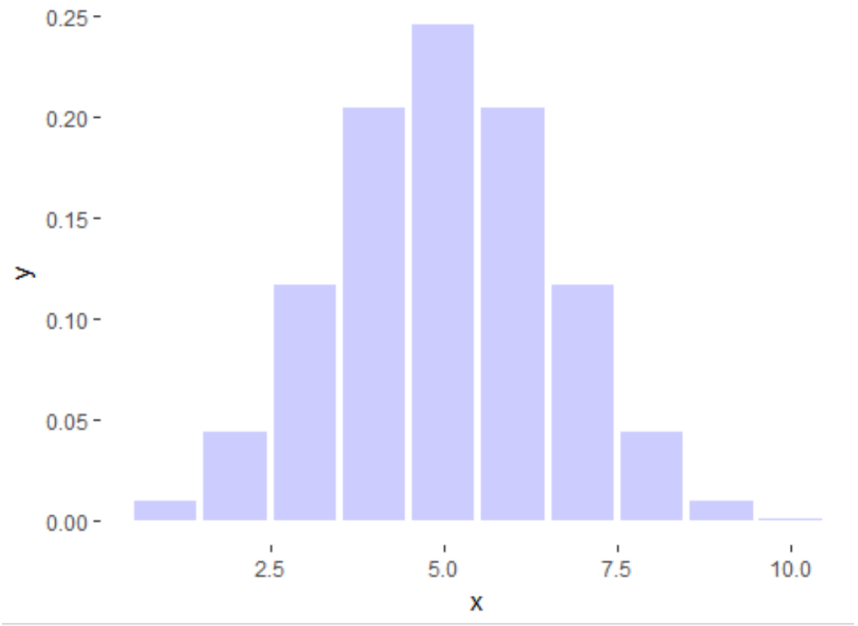
```
> dfRandBinom <- data.frame(x = rbinom(n=10,size = 10 , prob = .5))
>
> p1 <- ggplot(dfRandBinom, aes(x)) +
+   geom_histogram(alpha = .2, fill = 'blue', binwidth = .5) +
+   theme(panel.background = element_rect(fill = "white")) +
+   xlim(0, 10)
>   p1
```



This time, we got 2, 3 and 4 once each, 5 three times, and 6 and 7 twice each (n = 10). Notice how increasing the number of trials starts to spread out the distribution around the expected value (5 out of 10)

dbinom gives the density (probability since this is discrete - PMF) at each point in a supplied vector of values, assuming 10 trials with a probability of .5)

```
> dfDensityBinom <- data.frame(x, y = dbinom(x,size = 10 , prob = .5))
> p2 <- ggplot(dfDensityBinom, aes(x, y)) + geom_bar(alpha = .2,
+   fill = 'blue', stat = 'identity') +
+   theme(panel.background = element_rect(fill = "white"))
> p2
```



Plotting tip: use geom_bar for discrete data and geom_histogram for continuous

```
> manBinom <- function(n, p, h)
+ {
+   x <- factorial(n)/(factorial(h)*factorial(n-h))
+   x*(p)^h*(1-p)^(n-h)
+ }
> n <- 10 # number of trials
> p <- .5 # probability
> h <- seq(1,n,by=1) # number of successes
> dfManBinom <- data.frame(x=h, y = manBinom(n, p, h))
> ggplot(data = dfManBinom)+ geom_bar(aes(x, y),
+     stat = 'identity') + xlab("Successes") + ylab("Probability")
```

The Binomial distribution function is a pmf – a discrete probability *(so the pmf value = probability and you don't need to integrate, just add)*

$$\frac{n!}{\mathrm{h!(n-h)!}} p^h(1-p)^{\mathrm{n-h}}$$

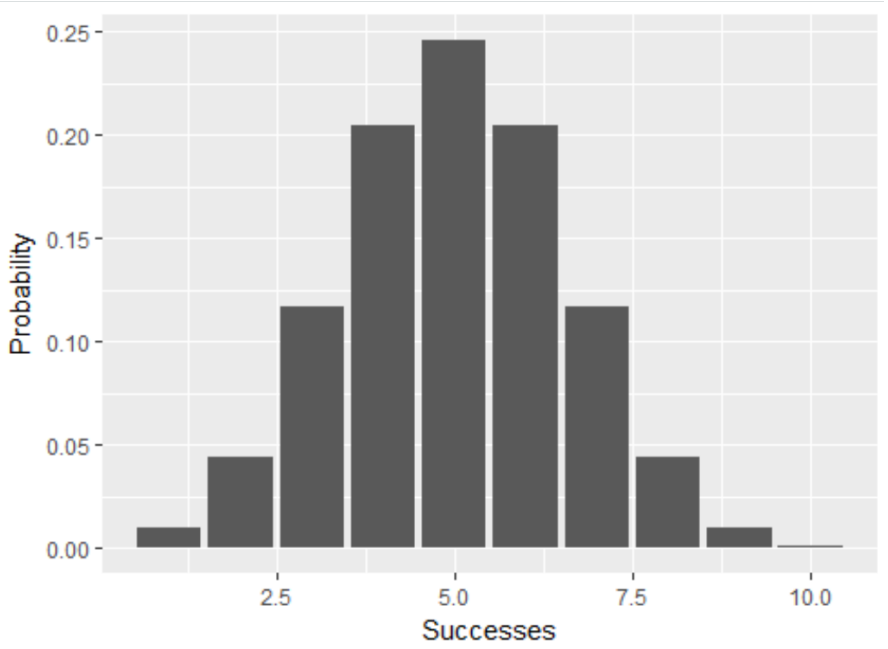
```
> sum(dfManBinom$y)
[1] 0.9990234
```

The binomial function computes probability of successes given n number of trials.

Here we compute the expected number of successes, given 10 trials with a .5 probability of success.

As you can see, we expect 5 successes with a little less than a 24% probability.

$$\frac{10!}{5!(10-5)!} .55(1-.5)^{10-5} = .24$$



Binomial Reparameterization $B(n,p) => N(\mu, \sigma^2)$

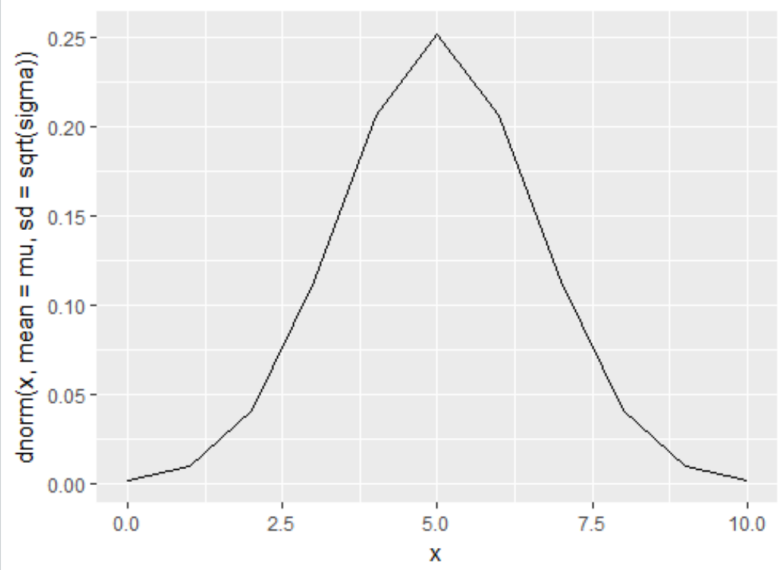
You might notice that the binomial, while discrete, is shaped like a normal distribution.

$$\frac{n!}{h!(n-h)!} p^h (1-p)^{n-h}$$

We can often *re-parameterized* a binomial as a normal with the following functions:

$\mu = n * p$
$\sigma^2 = np(1-p)$

then you have a continuous distribution that will predict # of successes (of course, you'll have to use integration vs addition – keep in mind, integration is really addition of infinitely small intervals)



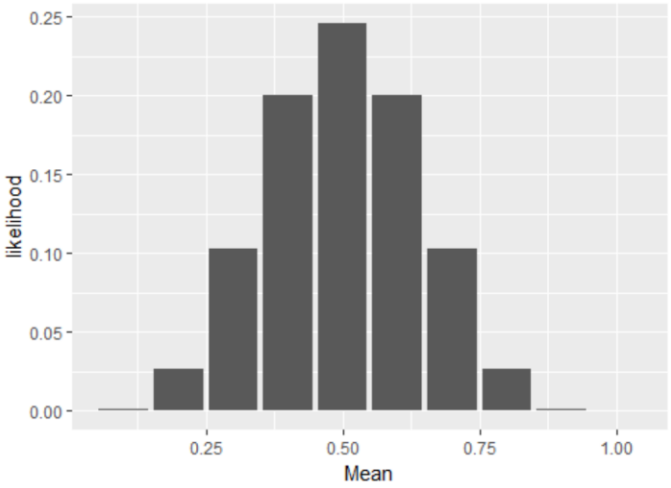

```
> mu <- n*p
> sigma <- n*p*(1-p)
> dfData <- data.frame(x = seq(0, 10, by = 1))
> p1 <- ggplot(dfData) +
+   geom_line(aes(x,y= dnorm(x, mean = mu, sd = sqrt(sigma))))
> p1
```

Note: reparameterization will become an important tool in Bayesian inference.

Binomial Likelihood

So let's change things a bit; *instead of varying h – let's vary p* and plot a density value for different values of p *using the same pdf*

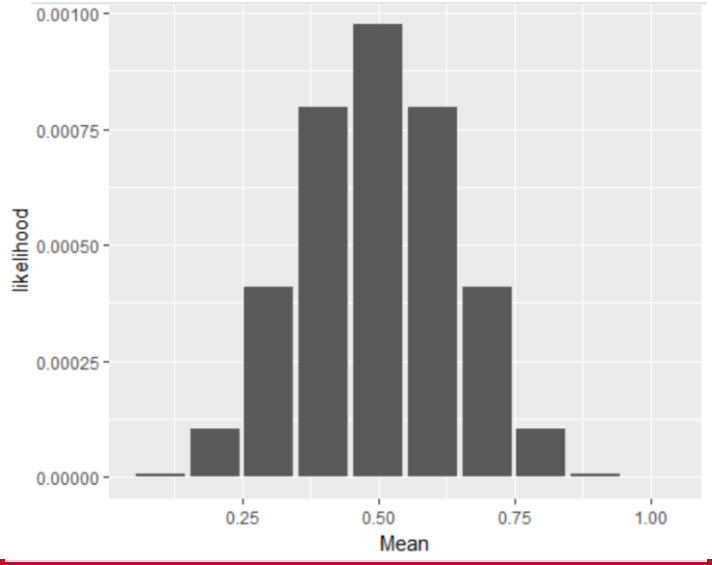
```
> h <- 5
> p <- seq(.1, 1, .1)
> dfManBinom <- data.frame(x=p, y = manBinom(n, p, h))
> ggplot(data = dfManBinom)+ geom_bar(aes(x, y), stat = 'identity')
+   xlab("Mean") + ylab("likelihood")
```



Now our pmf is a likelihood function. And the expected mean is now on the x axis.

But also notice that the first part of our pmf $\frac{n!}{h!(n-h)!}p^h(1-p)^{n-h}$ now becomes a constant because we're varying p, while holding the data constant.. So, we can simplify the likelihood function by dropping the constant

```
> likeBinom <- function(n, p, h)
+ {
+   (p)^h*(1-p)^(n-h)
+ }
> # so let's try this out
> dfManBinom <- data.frame(x=p, y = likeBinom(n, p, h))
> ggplot(data = dfManBinom)+
+   geom_bar(aes(x, y), stat = 'identity') +
+   xlab("Mean") + ylab("likelihood")
```

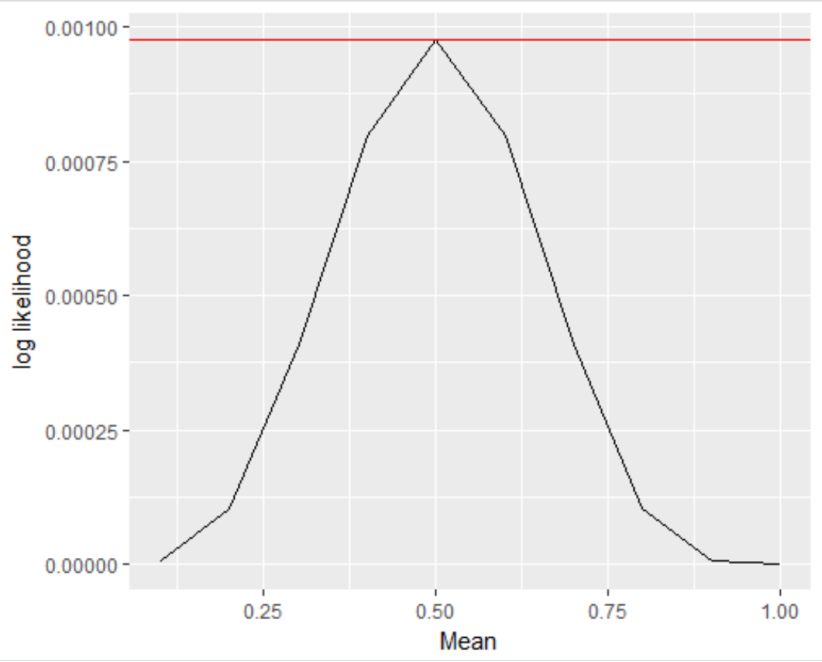


Also notice that the likelihood values decrease considerably *(without the constant multiplier)*, but the maximum point is still the same

And also, since we've switched to a grid of mean values, we've also switched to a continuous function, which means we can plot this out using a geom_line.

```
> p1 <- p1 + geom_hline(yintercept = y, color = 'red')
> p1
> p1 <- ggplot(data = dfManBinom)+ geom_line(aes(x, y), stat = 'identity') +
+   xlab("Mean") + ylab("log likelihood")
> p1
> x <- .5
> y <- (x)^h*(1-x)^(n-h)
> y
[1] 0.0009765625
> p1 <- p1 + geom_hline(yintercept = y, color = 'red')
> p1
```

And just for fun, instead of finding the maximum likelihood, we could use a derivative based solution. We know the function is $p^h(1-p)^{n-h}$, and the derivative (using the chain rule) is $5p^4(1-p)^4$ (-2p+1) $= 0$. The solution is .5, and the likelihood value at .5 $= .5^5 * (1-.5)^5 = .0009765625$ (see above). The line for the derivative is drawn to the right. You can solve simple problems using a derivative, but these become intractable quickly



(the derivative illustration was just for orientation and appreciation of the limits of derivative based solutions, and you won't be asked to optimize manually).

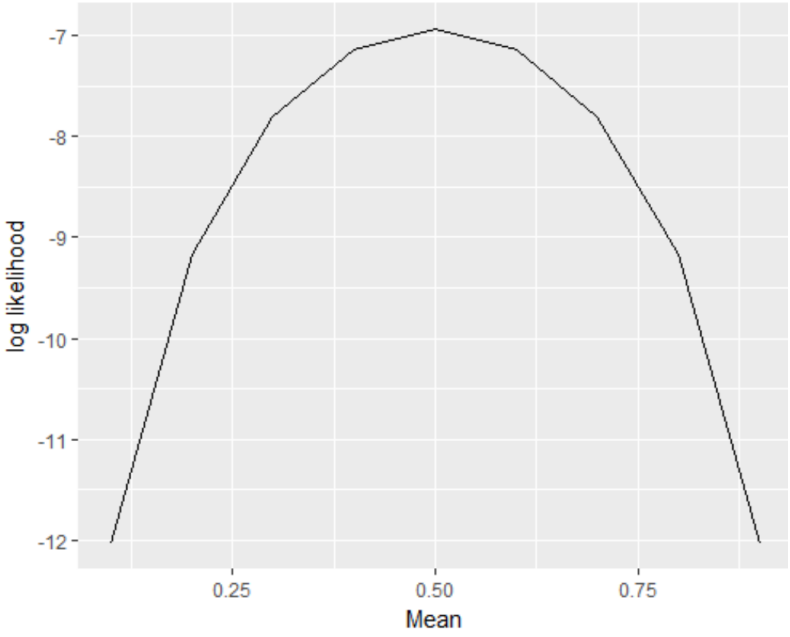
The best way to solve these problems is by using non-derivative based optimization. R provides a wide range of optimization packages, of which the based function optimize is the simplest

```
> negLikeBinom <- function(n, p, h)
+ {
+   -(p)^h*(1-p)^(n-h)
+ }
> mLO <- optimize(negLikeBinom, interval = c(0,1), n=n, h = h)
> mLO$minimum
[1] 0.5
```

And, as with the normal likelihood, the binomial likelihood can be simplified by taking the log of the function:

$$p^h(1-p)^{n-h} \propto h*log(p) + (n-h)*log(1-p)$$

```
> logLikeBinom <- function (n, p, h)
+ {
+  h*log(p) + (n-h)*log(1-p)
+ }
>
> dfManBinom <- data.frame(x=p, y = logLikeBinom(n, p, h))
> dfManBinom <- filter(dfManBinom, y > -100)
> # just to compenstate simple function
> ggplot(data = dfManBinom)+ geom_line(aes(x, y),
+   stat = 'identity') + xlab("Mean") + ylab("log likelihood")
```



So, to review, the likelihood and the log likelihood have different shapes *(the functions are different)*, but reach optimum at the same point with regard to θ values.

```
> negLogLikeBinom <- function (n, p, h)
+ {
+   -(h*log(p) + (n-h)*log(1-p))
+ }
> mLO <- optimize(negLogLikeBinom, interval = c(0,1,1), n=n, h = h)
> mLO$minimum
[1] 0.5
```

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

General Likelihood

Which brings us to the key concept of maximum likelihood: Likelihood is built on the PDF and is proportionate to probability $\mathcal{L} \propto \mathbb{P}$:

$\mathcal{L} \neq \mathbb{P}$	likelihood is not probability and it doesn't sum to 1
$\mathcal{L} \propto \mathbb{P}$	likelihood is proportional to probability *(we use the probability function (or log) to determine the likelihood)*.
$\mathcal{L}(\theta \mid D) = f * \mathbb{P}(D \mid \theta)$	The likelihood of a hypothesis *(or parameter θ)* given some data is proportional to the probability of obtaining data D given the hypothesis θ is true *(multiplied by a constant or proportional function)*

The critical difference between probability and likelihood is the interpretation of what is fixed and what can vary. In the case of traditional probability $\mathbb{P}(D \mid H)$, the hypothesis is fixed and the data are free to vary. In the case of $\mathcal{L}(H \mid D)$, the data are fixed and the hypothesis is free to vary. When we build models, we want to find the parameters θ of the models, so the likelihood function is $\mathcal{L}(\theta|D)$, where θ is a set *(or vector),* of parameters *(only 1 in this case – p).*

So, $p^h(1-p)^{n-h}$ becomes our likelihood function, and we can state $\mathcal{L}(\theta|D) = \theta^h(1-\theta)^{n-h}$ One last point: notice how we summed dfManBinom$y = 1. That was a probability as it summed (in the case of a pMf, or integrated, in the case of a pDf, to 1). And also notice how, when we dropped the constant from our pmf $\frac{n!}{h!(n-h)!}p^h(1-p)^{n-h}$ it became $p^h(1-p)^{n-h}$, and without the multipliers, the likelihood values were much lower (and will be even lower we tatke the log *h*log(p) + (n-h)*log(1-p)* in the next section. We can reallocate all of these by dividing by the sum (or integral) and then use those values comparatively.

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Beta Distribution

Recall how the likelihood values were smaller than the probability values because we dropped the first term of the pdf $\frac{n!}{h!(n-h)!}$, and even smaller still with the log likelihood *(sometimes, likelihood values get really tiny, or really large, and your desktop floating point limits will bomb the function – be aware!)*. Nevertheless, the function still produced the optimum parameter value we were seeking.

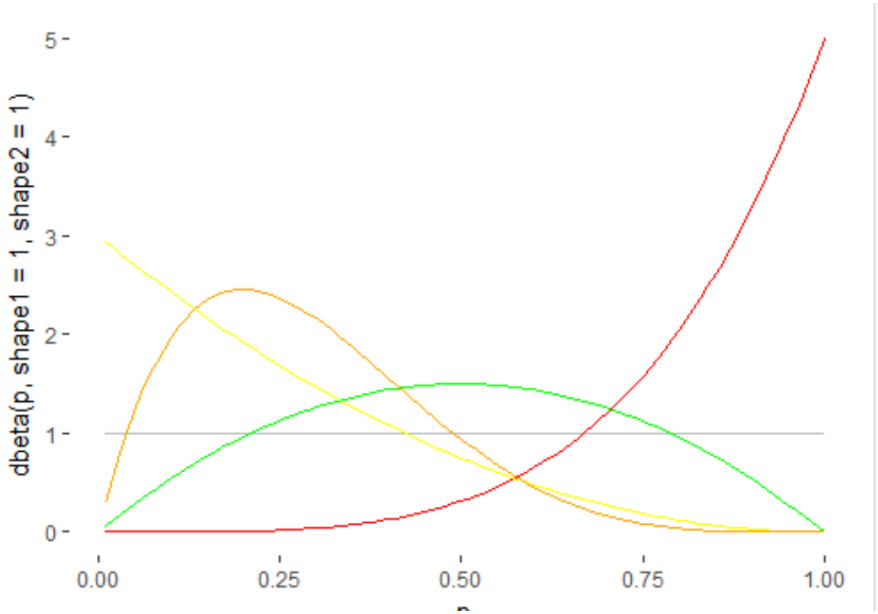
The **beta distribution** can be thought of as an extension of the binomial likelihood function $\theta^{h}(1-\theta)^{n-h}$, where h becomes alpha α, and n-h becomes beta β *(it's a little different, but that's a good starting estimate).* This is a reparameterization or sorts, and we're now getting into, what we call shape parameters. Shape parameters are a functional abstraction of parameters. They are sometimes called hyper-parameters *(which is incorrect - hyperparameters control the behavior of algorithms that find the parameters of other algorithms)*. But as you will see, varying shape parameters in distributions *(e.g., Beta, Gamma, Skew-Normal),* allows you to directly control… ah… the shape.

So the Binomial likelihood function $\theta^{h}(1-\theta)^{n-h}$, can be reparametrized as the beta pdf $\frac{\theta^{a-1}(1-\theta)^{b-1}}{\int_0^1 \theta^{a-1}(1-\theta)^{b-1}}$ The denominator, $\int_0^1 \theta^{a-1}(1-\theta)^{b-1}$, is a standardizing value *(like when we summed the binomial pdf to adjust to a probability scale)*. In the case of one θ this is not a big deal, but when we start adding theta vectors ($\theta_1\ \theta_2\ \theta_3$ …) and we start multiplying likelihood functions, then we'll be looking real hard to get rid of this integral.

For our purposes here, we can just work with $\theta^{a-1}(1-\theta)^{b-1}$ and divide by the sum (like we did before). Later on, we'll call this ***marginal probability***.

Beta Distribution

$Beta(\theta, \alpha, \beta)$




```
> p <- seq(from = .01, to = 1, by = .01)
> dfData <- data.frame(p)
>
> # dbinom gives us the probablity of a # of successes
> # based on the number of trials and probablity
>
> p1 <- ggplot( data = dfData) +
+   geom_line(mapping = aes(x = p, y = dbeta(p, shape1 = 1, shape2 = 1)),
+             stat = 'identity',  color = "gray") +
+   geom_line(mapping = aes(x = p, y = dbeta(p, shape1 = 5, shape2 = 1)),
+             stat = 'identity',  color = "red") +
+   geom_line(mapping = aes(x = p, y = dbeta(p, shape1 = 1, shape2 = 3)),
+             stat = 'identity', color = "yellow") +
+   geom_line(mapping = aes(x = p, y = dbeta(p, shape1 = 2, shape2 = 2)),
+             stat = 'identity',  color = "green") +
+   geom_line(mapping = aes(x = p, y = dbeta(p, shape1 = 2, shape2 = 5)),
+             stat = 'identity', color = "orange") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```

Density Function *(PDF)*

dbeta

$$\frac{\theta^{a-1}(1-\theta)^{b-1}}{\int_0^1 \theta^{a-1}(1-\theta)^{b-1}} \propto \theta^{a-1}(1-\theta)^{b-1}$$

```
manBeta <- function(theta, a, b)
{
  (theta)^(a-1)*(1-theta)^(b-1)
}
```

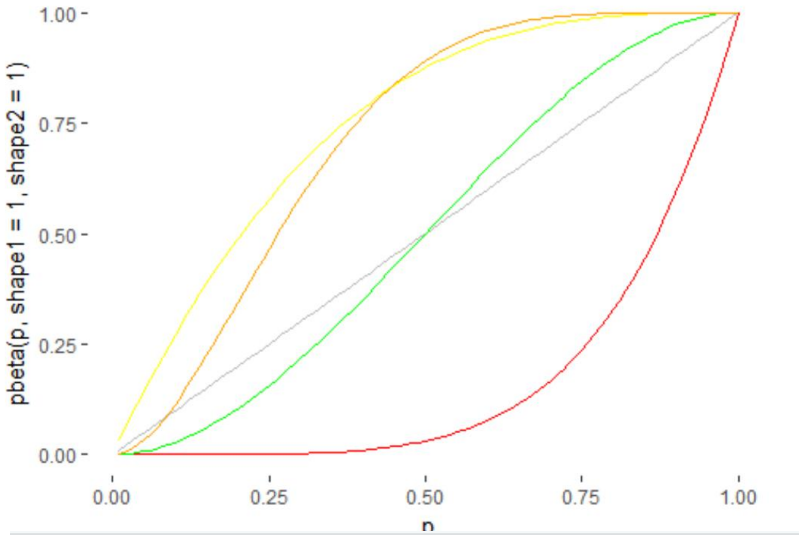
Notice how the probabilities are flat when the a and b are both = 1 (this is an important concept in Bayesian inference)

Also notice how when both = 2, the distribution starts to take more of a normal distribution shape, but when the parameters are proportionately far apart, the shape skews – sometimes drastically.

Cumulative Density

pbeta

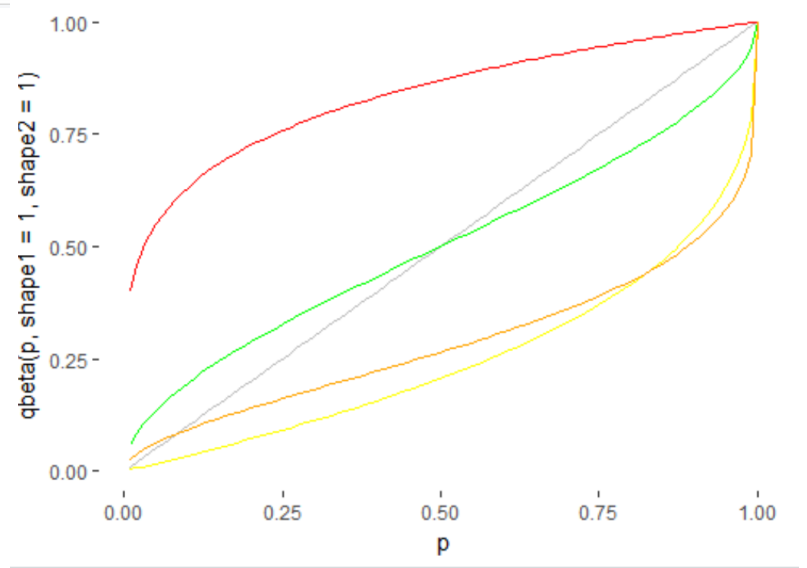
```
> p1 <- ggplot( data = dfData) +
+   geom_line(mapping = aes(x = p, y = pbeta(p, shape1 = 1, shape2 = 1
+             stat = 'identity',  color = "gray") +
+   geom_line(mapping = aes(x = p, y = pbeta(p, shape1 = 5, shape2 = 1
+             stat = 'identity',  color = "red") +
+   geom_line(mapping = aes(x = p, y = pbeta(p, shape1 = 1, shape2 = 3
+             stat = 'identity', color = "yellow") +
+   geom_line(mapping = aes(x = p, y = pbeta(p, shape1 = 2, shape2 = 2
+             stat = 'identity',  color = "green") +
+   geom_line(mapping = aes(x = p, y = pbeta(p, shape1 = 2, shape2 = 5
+             stat = 'identity', color = "orange") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```



Quantile

pbeta

```
> p1 <- ggplot( data = dfData) +
+   geom_line(mapping = aes(x = p, y = qbeta(p, shape1 = 1, shape2 = 1)),
+             stat = 'identity',  color = "gray") +
+   geom_line(mapping = aes(x = p, y = qbeta(p, shape1 = 5, shape2 = 1)),
+             stat = 'identity',  color = "red") +
+   geom_line(mapping = aes(x = p, y = qbeta(p, shape1 = 1, shape2 = 3)),
+             stat = 'identity', color = "yellow") +
+   geom_line(mapping = aes(x = p, y = qbeta(p, shape1 = 2, shape2 = 2)),
+             stat = 'identity',  color = "green") +
+   geom_line(mapping = aes(x = p, y = qbeta(p, shape1 = 2, shape2 = 5)),
+             stat = 'identity', color = "orange") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```



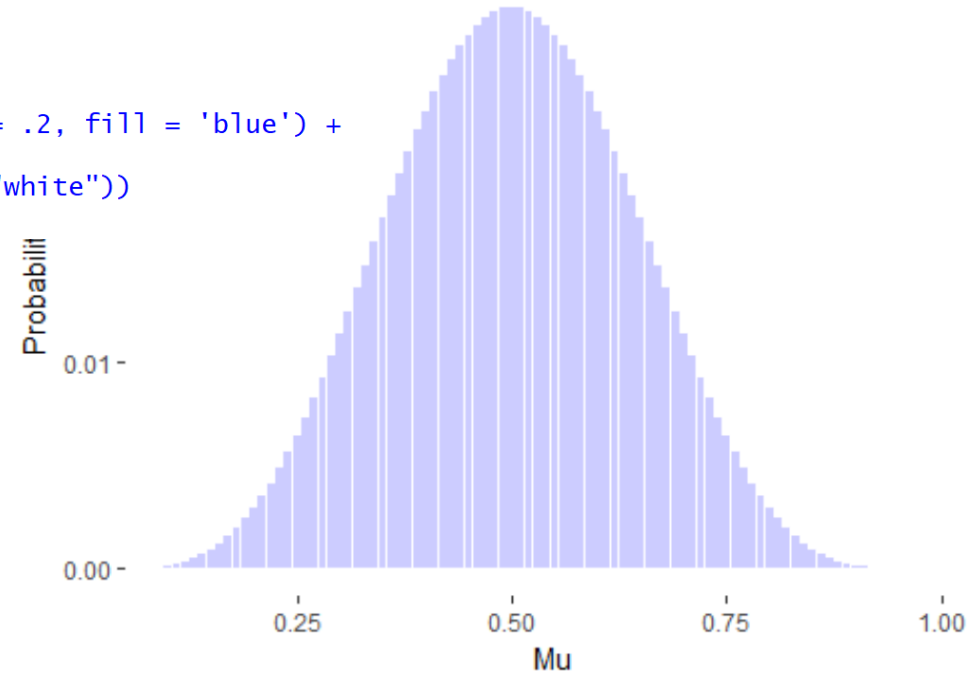
Beta:

dbeta(x, shape1, shape2, ncp = 0, log = FALSE) returns density function
pbeta(q, shape1, shape2, ncp = 0, lower.tail = TRUE, log.p = FALSE) returns probability
qbeta(p, shape1, shape2, ncp = 0, lower.tail = TRUE, log.p = FALSE) returns quantile
rbeta(n, shape1, shape2, ncp = 0) returns n random values based on distribution parameters

These functions all in the gamma multiplier, which we dropped for our manBeta function. So, like the binomial likelihood, we need to standardize. But also like the binomial likelihood, we still get to the same place for purposes of parameter analysis.

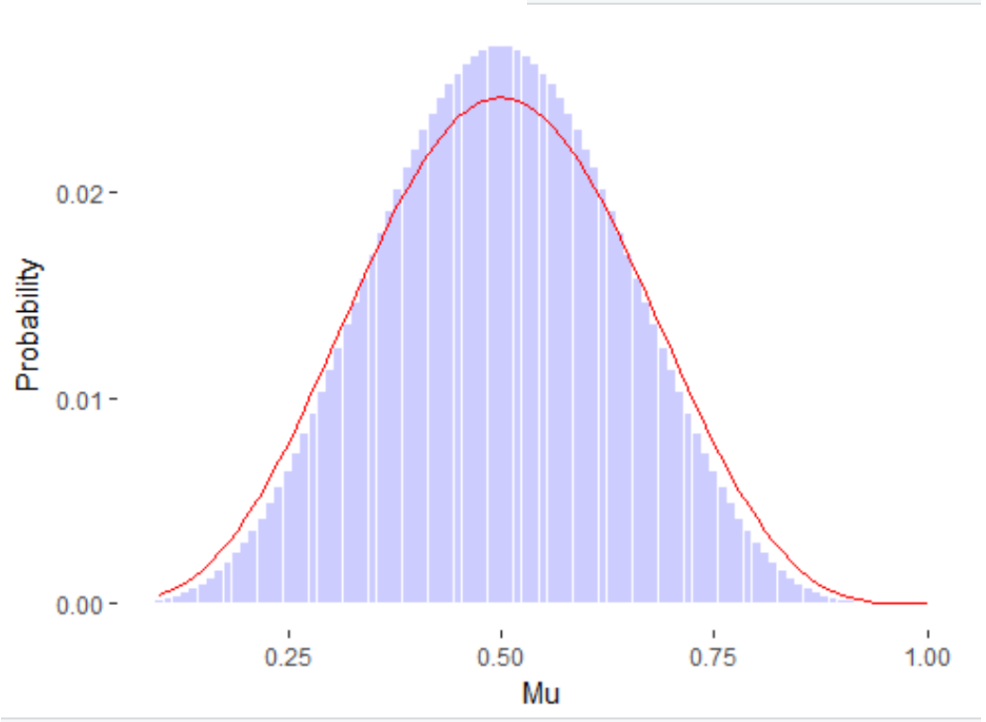
Using a binomial distribution to parameterize a beta.

```
> # first lets review the binomial likelihood
>
> n <- 10
> h <- 5
> p <- seq(.1, 1, .01) # changing p
> dfManBinom <- data.frame(x=p, y = manBinom(n, p, h))
> sum(dfManBinom$y)
[1] 9.08891
> # the likelihodd doesn't sum to 1, so we standardize
> dfManBinom$y <- dfManBinom$y/sum(dfManBinom$y)
> sum(dfManBinom$y) # standardize
[1] 1
>
> p10 <- ggplot(data = dfManBinom)+
+   geom_bar(aes(x, y), stat = 'identity', alpha = .2, fill = 'blue') +
+   xlab("Mu") + ylab("Probability") +
+   theme(panel.background = element_rect(fill = "white"))
> p10
```



Here, we use # of trials (n) and # of successes (h) to set the a and b parameters of the beta distribution. As you can see, it's a nice fit.

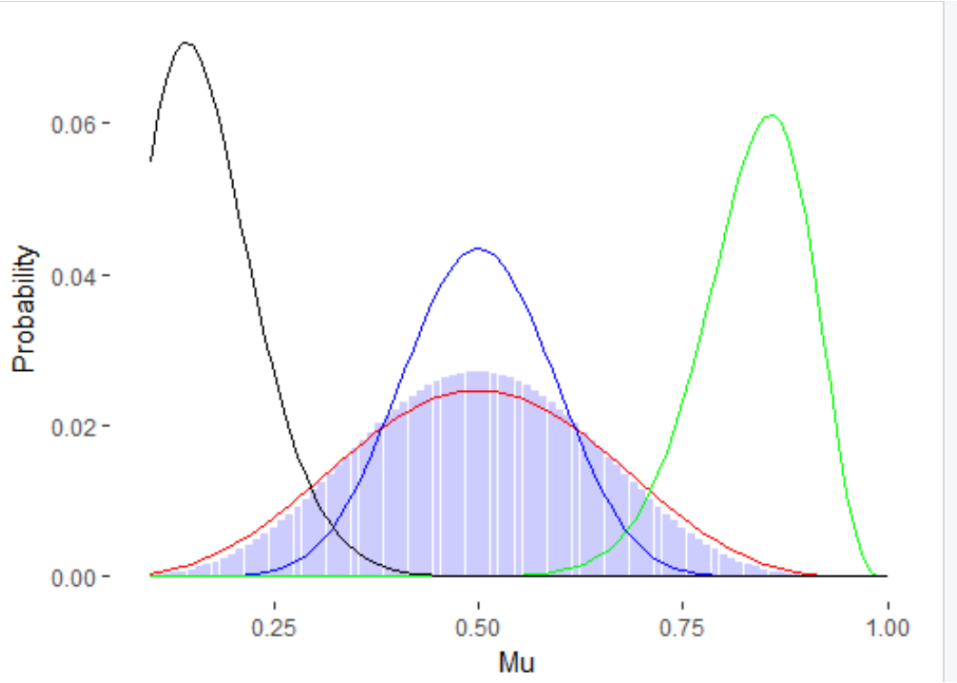
```
> dfBeta <- data.frame(x <- p, ManBeta = manBeta(p, h, (n-h)))
> dfBeta$ManBeta <- dfBeta$ManBeta/sum(dfBeta$ManBeta) # standardize the beta
> sum(dfBeta$ManBeta)
[1] 1
> p10 <- p10 + geom_line(data = dfBeta, aes(x, ManBeta), color = "red")
> p10
```



What's really special about the beta distribution is that we can quickly change the location and shape by adding and subtractiing from total trials, successes and failures (n-h).

You should try this yourself. Change the a and b and see what happens.

```
> # let's increase the number of successes and trails (proportionately)
> dfBeta$z <- manBeta(p, h+10, (n-h+10))
> dfBeta$z <- dfBeta$z/sum(dfBeta$z)
> p10 <- p10 + geom_line(data = dfBeta, aes(x, z), color = "blue")
> p10
> # notice how the variance decreases as we increase sample size
>
> # or what if we only increase successes
> dfBeta$z2 <- manBeta(p, h+20, ((n+20)-(h+20)))
> dfBeta$z2 <- dfBeta$z2/sum(dfBeta$z2)
> sum(dfBeta$z2)
[1] 1
> p10 <- p10 + geom_line(data = dfBeta, aes(x, z2), color = "green")
> p10
>
> # or what if we increase failures
> dfBeta$z3 <- manBeta(p, h, ((n+20)-(h)))
> dfBeta$z3 <- dfBeta$z3/sum(dfBeta$z3)
> sum(dfBeta$z3)
[1] 1
> p10 <- p10 + geom_line(data = dfBeta, aes(x, z3))
> p10
```



BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Exercise

```
> dfSO <- dbGetQuery(con2,"
+ Select [Quote_ID]
+ ,[Customer_ID]
+ ,[Product_ID]
+ ,[Quote]
+ ,[Competitor_ID]
+ ,[Competitor_Quote]
+ ,[Date_Received]
+ ,[Date_Due]
+ ,[Date_Submitted]
+ ,[Date_Required]
+ ,[ATP]
+ ,[Result]
+ FROM [dbo].[Quote]
+ ")
>
> SalesStats <- dfSO %>% filter(Result %in% c("L", "W")) %>% dplyr::count(Result)
```

Select all the data from the table Quote in the Accouting database.

Put that in a dataframe called SalesStats, but first filter to include only “L” and “W” values in Result *(there are some data errors)*, and then count each result. The SalesStats table should only have two rows.

Then convert the total results to numeric and put that in a variable n. Summarize the “W”s and put that in a variable h. and then set up a grid of p values sea(from .1 to 1, by .01) .01

```
> n <- as.numeric(sum(SalesStats$n))
> h <- as.numeric(SalesStats %>% filter(Result == "W") %>% summarise(Tot=sum(n)))
> p <- seq(.1, 1, .01) # changing p
```

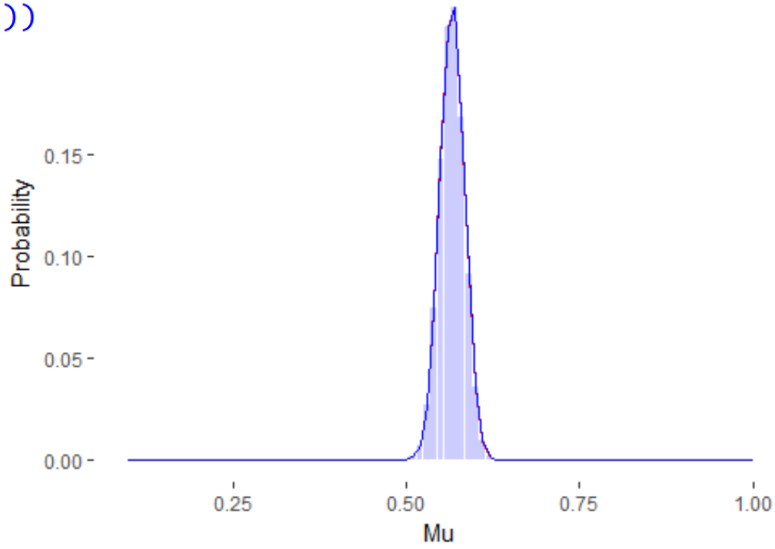
```
> dfSalesSuccess <- data.frame(x = p, y = dbinom(h, size = n, prob = p),
+   z = likeBinom(n, p, h),
+   z2 = dbeta(p, h, (n-h))) # manBeta blows up the factorals
> dfSalesSuccess$y <- dfSalesSuccess$y/sum(dfSalesSuccess$y)
> dfSalesSuccess$z <- dfSalesSuccess$z/sum(dfSalesSuccess$z)
> dfSalesSuccess$z2 <- dfSalesSuccess$z2/sum(dfSalesSuccess$z2)
> sum(dfSalesSuccess$y)
[1] 1
> sum(dfSalesSuccess$z)
[1] 1
> sum(dfSalesSuccess$z2)
[1] 1
```

Now create a dataframe called dfSalesSuccess with a column x = your probability grid, and y = dbinom(h, size, prob). Also, create a column z with binomial likelihood values, and z2 with dbeta values). Then standardize all the computed columns.

Note: You can't use manBeta here – the values will blow up on a laptop (dBinom calls a c program that scales the values – later we will be using our likelihood functions in a couple of programs, JAGS and STAN, that also interface with R)

Now plot the data. The y column is a dbinom, so it's discrete – plot that using geom_bar. Then use geom_line to plot out likebinom and the dbeta functions. Compare all.

```
> p12 <- ggplot(data = dfSalesSuccess)+
+   geom_bar(aes(x, y), stat = 'identity', alpha = .2, fill = 'blue') +
+   xlab("Mu") + ylab("Probability") +
+   geom_line(data = dfSalesSuccess, aes(x, z), color = "red") +
+   geom_line(data = dfSalesSuccess, aes(x, z2), color = "blue") +
+   theme(panel.background = element_rect(fill = "white"))
> p12
```



BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Skew Normal

$SN(\lambda, \delta, \alpha)$

Recall normal PDF $\frac{1}{\sqrt{2\pi} * \sigma} - \frac{(x-\mu)^2}{e\ 2\sigma^2}$, and standardized z score $Z = \frac{X - \mu}{\sigma}$

So, we can restate the standardized normal $N(0,1)$ pdf as: $\boldsymbol{\phi}(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$, or $exp\left[\frac{\frac{-x^2}{2}}{\sqrt{2\pi}}\right]$

Note: phi $\boldsymbol{\phi}$ is used to denote a DF

We'll say that the cumulative distribution function *(CDF)* can be denoted as:

$\boldsymbol{\Phi(z) = \int_{-\infty}^{\alpha x} \boldsymbol{\phi}(x)\ dx}$,

and the random variable x is said to have a skew normal distribution if its pdf is given by:

$f_\lambda(x) = 2\ \boldsymbol{\phi}(x)\ \boldsymbol{\Phi}(\alpha x)$

where α is a shape parameter. As α parameter increases in value from 0 *(normal distribution)*, skewness increases (with a $+\alpha$ corresponding to a positive skew, and $-\alpha$ corresponding to a negative skew.

This is the basic concept of a skew normal, but we add location λ and scale δ parameters to define values. (we will typically call λ xi)

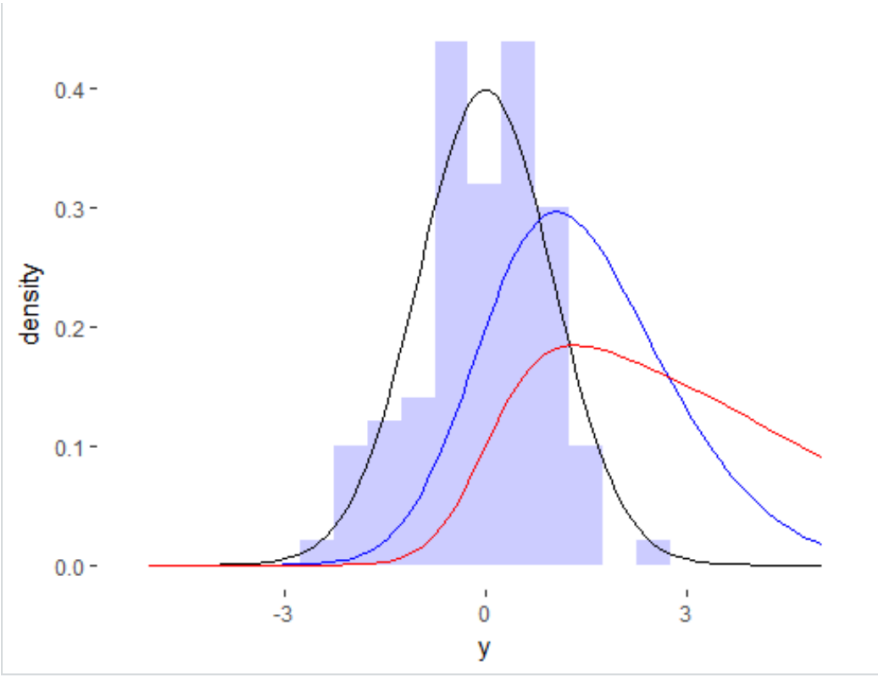
BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Skew Normal

So, a very simple version of the skew normal pdf works out to be be $\mathrm{SN}(\lambda, \delta, \alpha)$

Notice how we vary the values xi *(location)*, w *(scale)* and a *(shape)*.

```
> skew <- function(x,e,w,a){
+   t <- (x-e)/w
+   2/w * dnorm(t) * pnorm(a*t)
+ }
> # e location
> # w scale
> # a shape
>
> x = seq(from = -5, to = 5, length.out = 100)
>
> dfSN <- data.frame(x = x, y = rnorm(x, 0, 1))
>
> p1 <- ggplot(data = dfSN) +
+   geom_histogram(aes(y,..density..), alpha = .2,
+                  fill = 'blue', binwidth = .5) +
+   geom_line(aes(x, skew(x, 0, 1, 0))) +
+   geom_line(aes(x, skew(x, 0, 2, 2)), color = "blue") +
+   geom_line(aes(x, skew(x, 0, 4, 6)), color = "red") +
+   theme(panel.background = element_rect(fill = "white"))
> p1
```

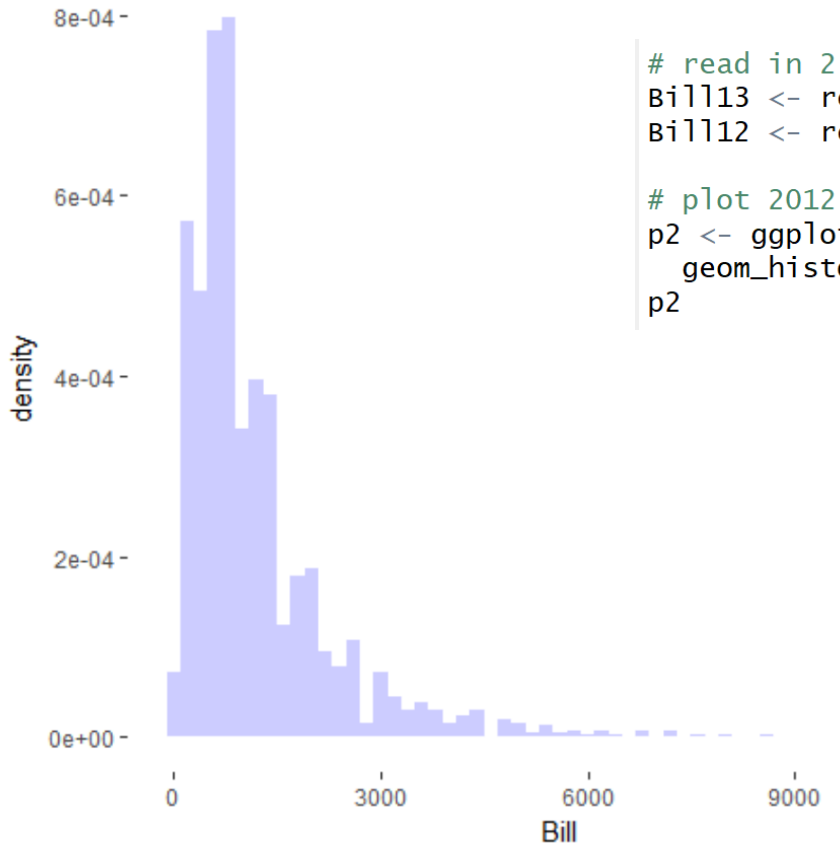


Also notice that we used the $f_\lambda(x) = 2\,\boldsymbol{\phi}(x)\,\boldsymbol{\Phi}(\alpha x)$
With location and scale coefficients added

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Skew Normal

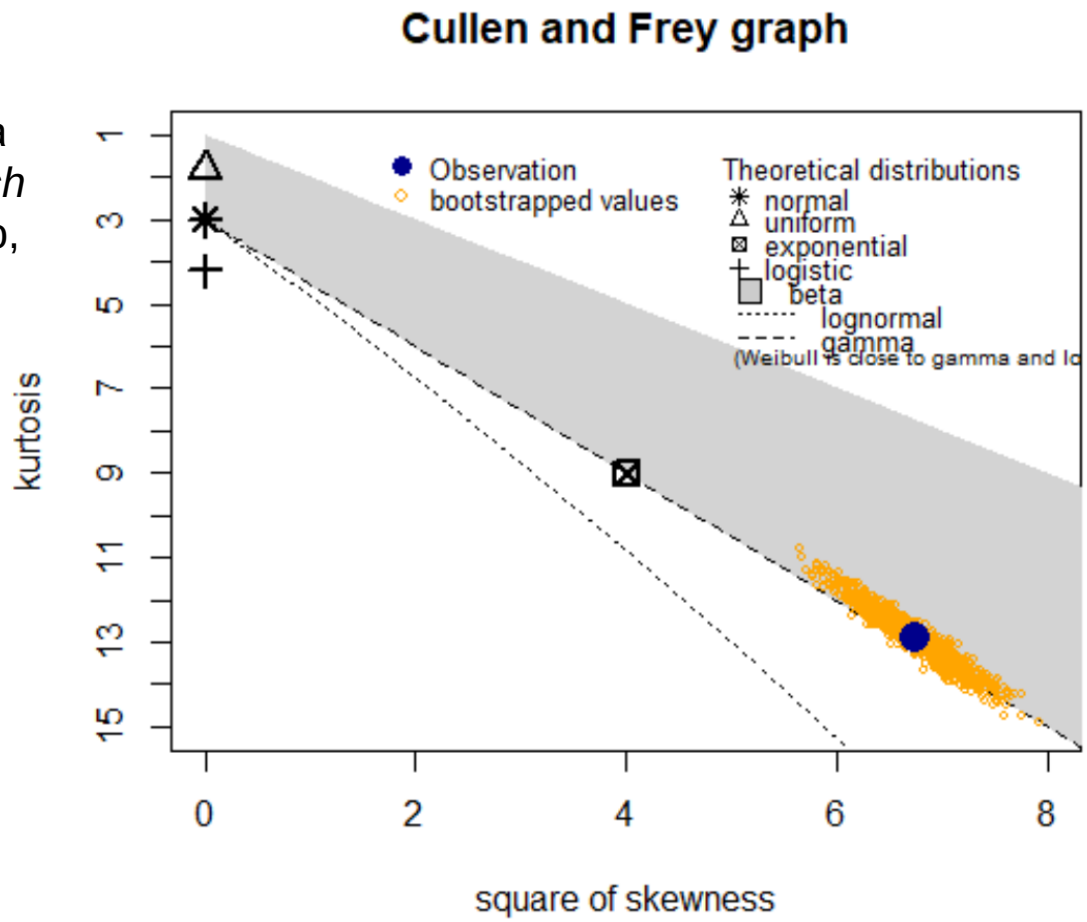
Let's walk through an example. Read in the billing data for 2012 and 2013, and plot a histogram of 2012




```
# read in 2 years of billings
Bill13 <- read_csv("sumBill13.csv")
Bill12 <- read_csv("sumBill12.csv")

# plot 2012 transactions
p2 <- ggplot(data =Bill12, aes(Bill, ..density..))  +
  geom_histogram(binwidth = 200, alpha = .2, fill = 'blue')
p2
```

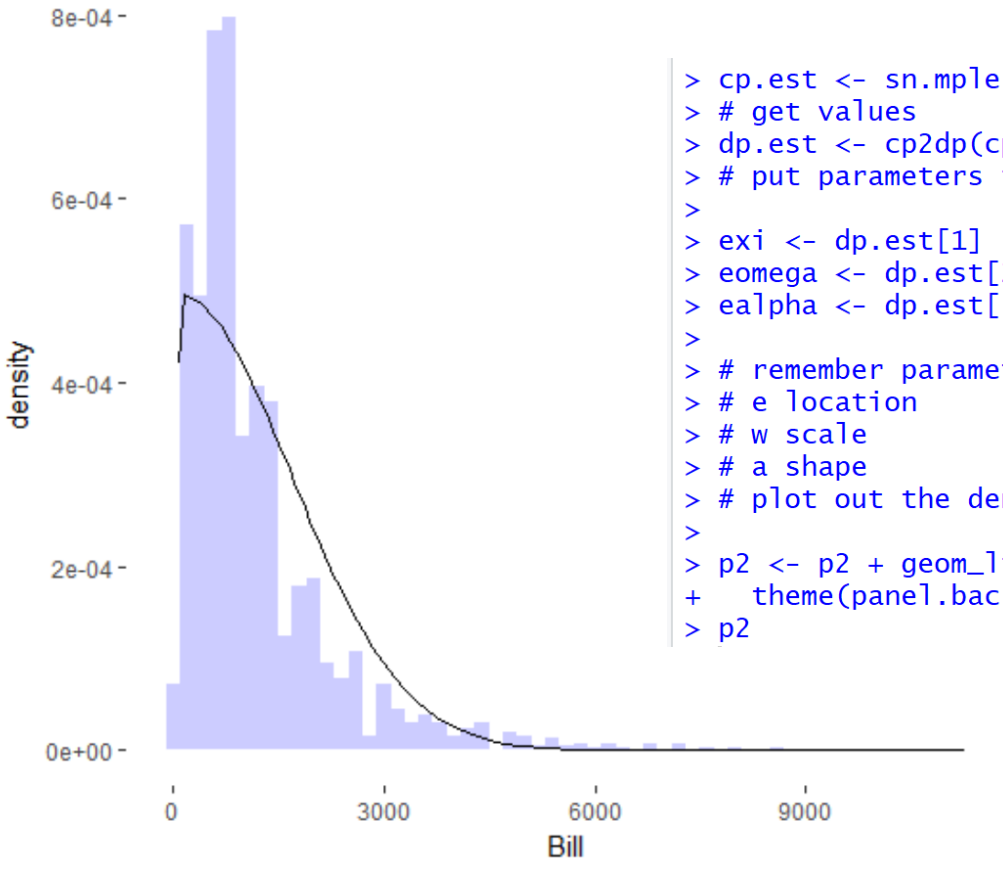
We'll run a quick CF analysis. Obviously the distribution is skewed.

A gamma or Weibull don't fit this data *(nor most transaction type data, which always runs into negative values)*. So, we'll go with a skew normal.



Exercise

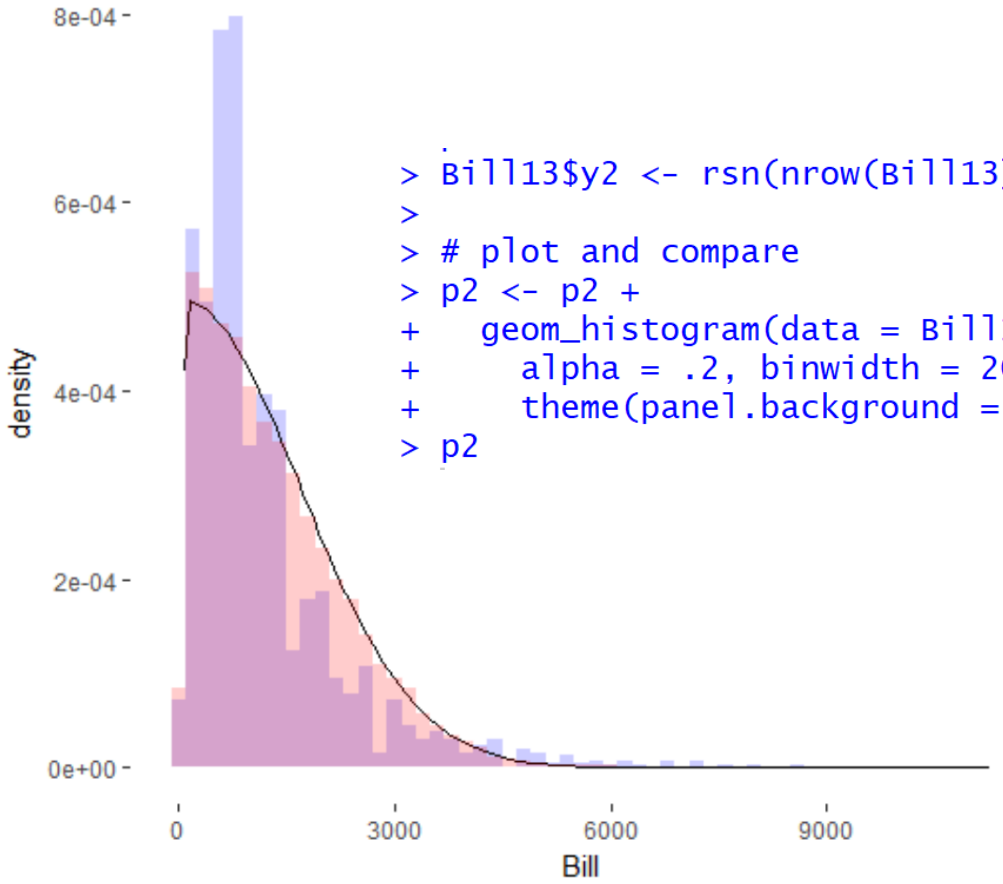
(good conceptual frame for baysian analysis which will be introduced shortly).
The sn package has a maximum likelihood parameter estimate. Here, we use sn.mple to estimate the distribution, and then use cp2dp to get parameters. We then pass the parameters to our simple skew function and then we plot of the pdf values…




```
> cp.est <- sn.mple(y = Bill12$Bill, opt.method = "nlminb")$cp
> # get values
> dp.est <- cp2dp(cp.est, family = "SN")
> # put parameters in variables
>
> exi <- dp.est[1]
> eomega <- dp.est[2]
> ealpha <- dp.est[3]
>
> # remember parameter meanings!
> # e location
> # w scale
> # a shape
> # plot out the density using the skew fuction (just an estimate)
>
> p2 <- p2 + geom_line(aes(Bill, skew(Bill, exi, eomega, ealpha))) +
+   theme(panel.background = element_rect(fill = "white"))
> p2
```

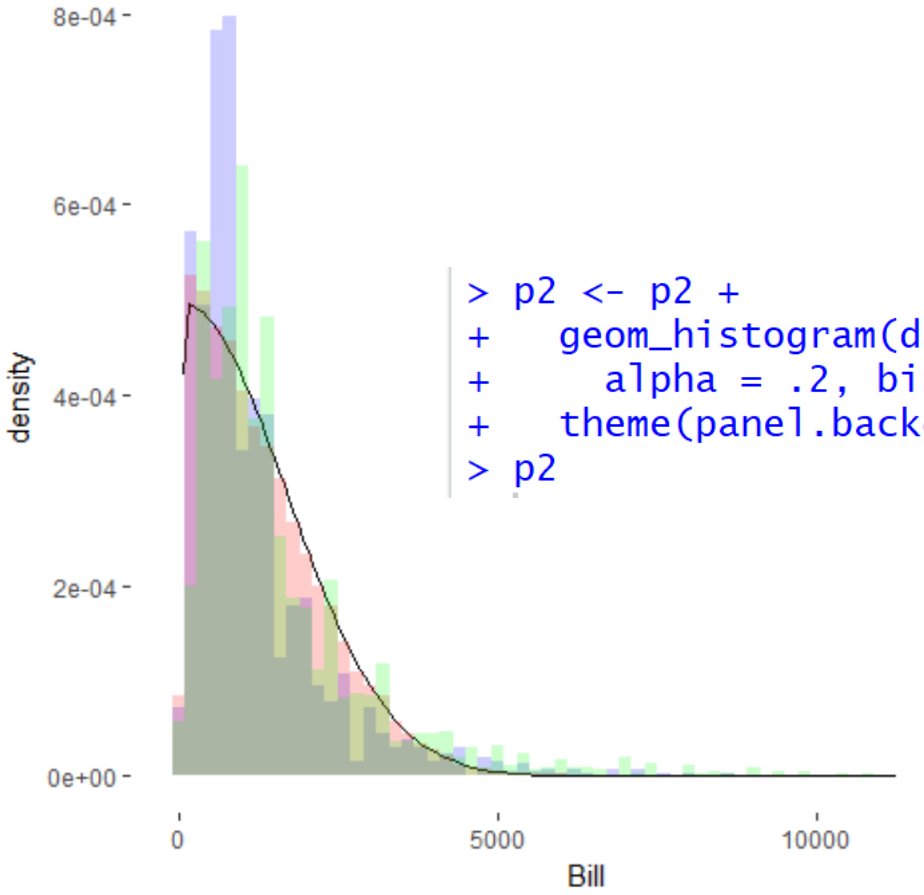
TA2 Distributions Review 3 Skew Normal.R

Now we use the same parameters we estimated for 2012, and we simulate 2013 using the same number of 2013 transactions with the 2012 parameters. We plot that out in red *(notice how well it fits the density outline – we would expect that in simple simulation)*



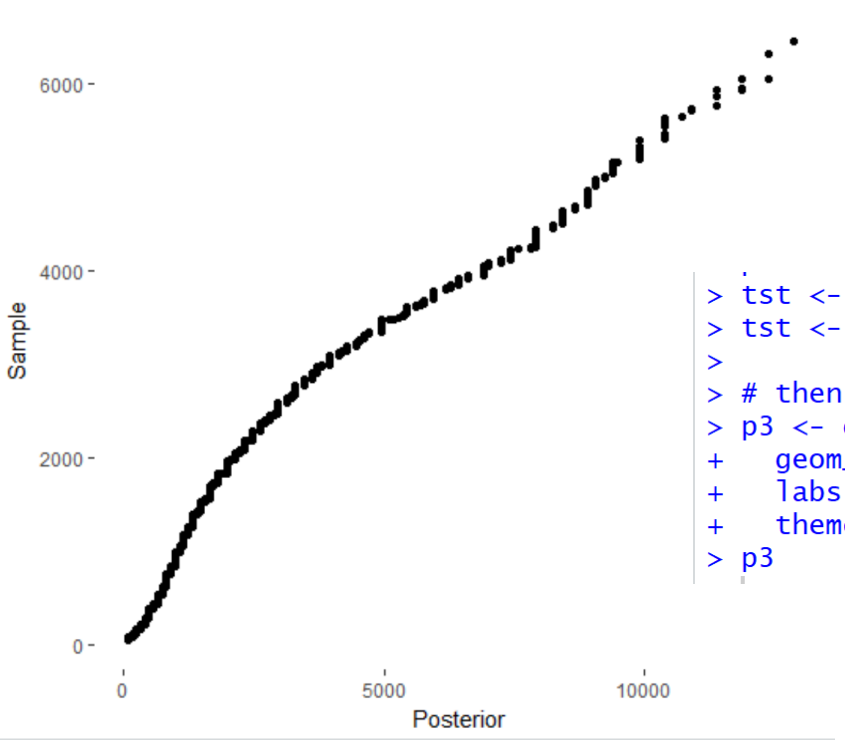

```
> Bill13$y2 <- rsn(nrow(Bill13), xi=exi , omega=eomega , alpha=ealpha )
>
> # plot and compare
> p2 <- p2 +
+   geom_histogram(data = Bill13, aes(y2, ..density..),
+     alpha = .2, binwidth = 200, fill = 'red') +
+     theme(panel.background = element_rect(fill = "white"))
> p2
```

Now we plot out the actual 2013 data and compare that with the simulated. Still a pretty good fit. Let's check for outliers.



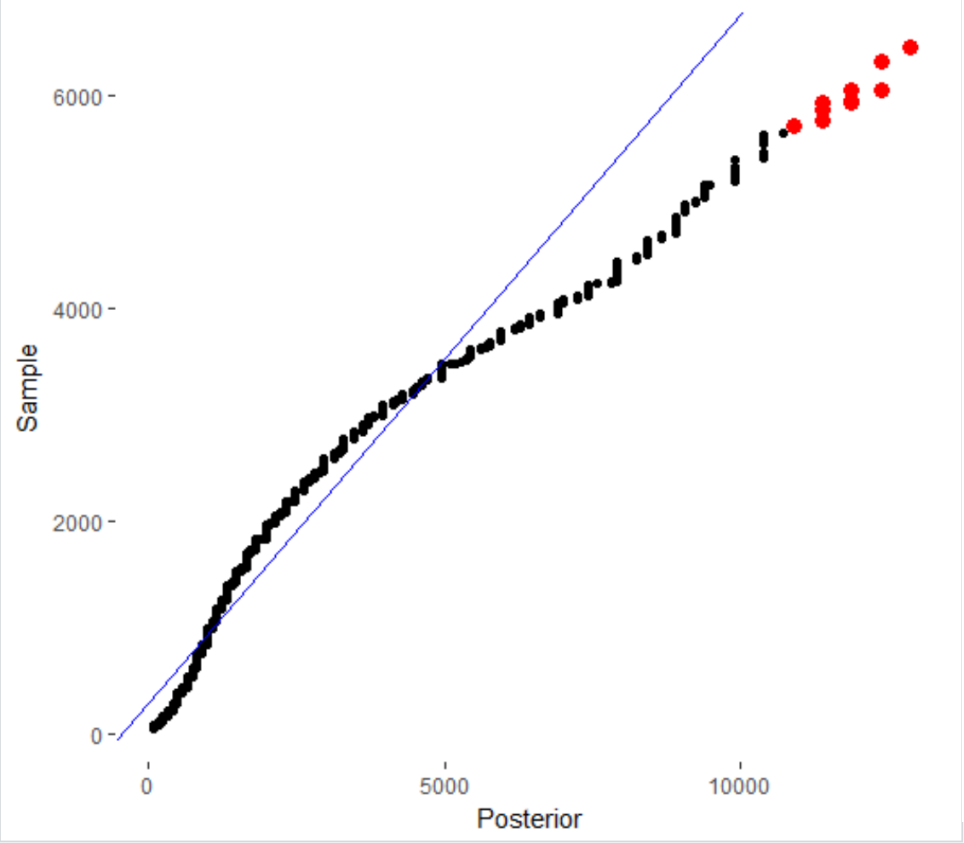

```
> p2 <- p2 +
+   geom_histogram(data = Bill13, aes(Bill, ..density..),
+     alpha = .2, binwidth = 200, fill = 'green') +
+   theme(panel.background = element_rect(fill = "white"))
> p2
```

Remember the QQ? It's not just for normal distributions. We can compare any two distributions that are conjugate. In this case, we're getting a fairly good tracking between the two distributions' quantiles. Maybe a few outliners…



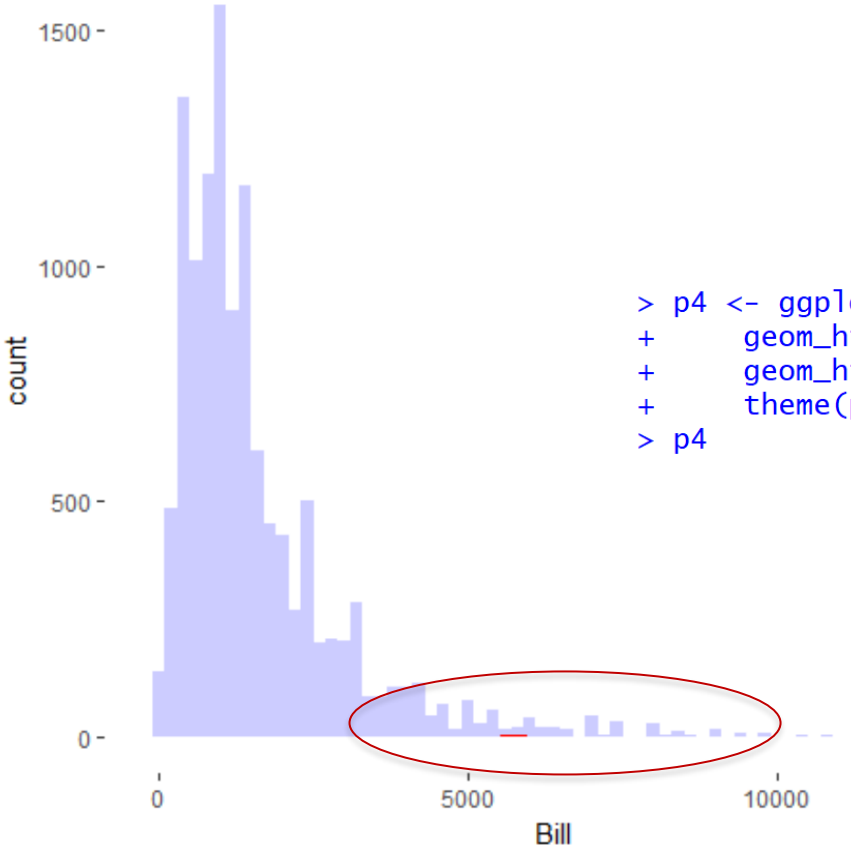

```
> tst <- data.frame(q5 = sort(Bill13$Bill), q6 = sort(Bill13$y2))
> tst <- rownames_to_column(tst, "SampleID")
>
> # then plot using a qq
> p3 <- ggplot(tst) +
+   geom_point(aes(x=q5, y=q6)) +
+   labs(x="Posterior",y="Sample") +
+   theme(panel.background = element_rect(fill = "white"))
> p3
```

Here, we create a liner model of the qq values, we plot it out (see the blue line). And then we apply a Bonferonni p-value test on the residuals to identify outliers. We then replot the outliers in red.




```
> # let's create a linear model to help us identify any outliers
> tstMod <- lm(q6 ~ q5, data = tst)
> summary(tstMod)

Call:
lm(formula = q6 ~ q5, data = tst)

Residuals:
     Min       1Q   Median       3Q      Max
-2258.18  -234.23    25.66   252.55   395.20

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.001e+02  3.809e+00   78.77   <2e-16 ***
q5          6.461e-01  1.754e-03  368.33   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 281.5 on 12114 degrees of freedom
Multiple R-squared:  0.918,     Adjusted R-squared:  0.918
F-statistic: 1.357e+05 on 1 and 12114 DF,  p-value: < 2.2e-16

>
> p3 <- p3 + geom_abline(intercept = tstMod$coefficients[1],
+   slope = tstMod$coefficients[2], color = 'blue') +
+   theme(panel.background = element_rect(fill = "white"))
> p3
>
> # Bonferonni p-value for most extreme values
> outliers <- data.frame(outlierTest(tstMod)[1])
> outliers <- rownames_to_column(outliers, "SampleID")
> outliers <- tst %>% inner_join(outliers, by = "SampleID")
>
> p3 <- p3 + geom_point(data = outliers, aes(x = q5, y = q6),
+     color = 'red', size = 3) +
+     theme(panel.background = element_rect(fill = "white"))
> p3
```

We also plot the outliers against the original histogram. Hard to see because there's not many (good fit). If you were performing a audit of the billing account, these might be worth looking at… note: you typically don't select audit transactions based on one dimension – this is for learning.




```
> p4 <- ggplot(data = Bill13, aes(Bill)) +
+     geom_histogram(alpha = .2, binwidth = 200, fill = 'blue')+
+     geom_histogram(data = outliers, aes(x = q6), fill = 'red') +
+     theme(panel.background = element_rect(fill = "white"))
> p4
```

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Terminology

Terminology *(just to review so we don't get confused)*:

Continuous Y:

One y: univariate analysis
One y and one X: bivariate analysis
One y and many x: multiple analysis
Many y and one x: multivariate analysis
Many y and many x: multivariate multiple analysis

Nominal Y:

One y with 2 values: binomial
Many y values: multinomial

BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Bivariate & Multivariate

Bivariate Normal

Recall normal PDF $\frac{1}{\sqrt{2\pi} * \sigma} \; e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Extending to bivariate: $\frac{1}{2\pi\sigma_x\sigma_x\sqrt{1-p^2}} \; e^{-\frac{z}{2(1-p)^2}}$ Where z = $\frac{(x_1-\mu_1)^2}{\sigma^2{}_1} - \frac{2p(x_1-\mu_1)}{\sigma^2{}_1\sigma^2{}_2} + \frac{(x_2-\mu_2)^2}{\sigma^2{}_2}$

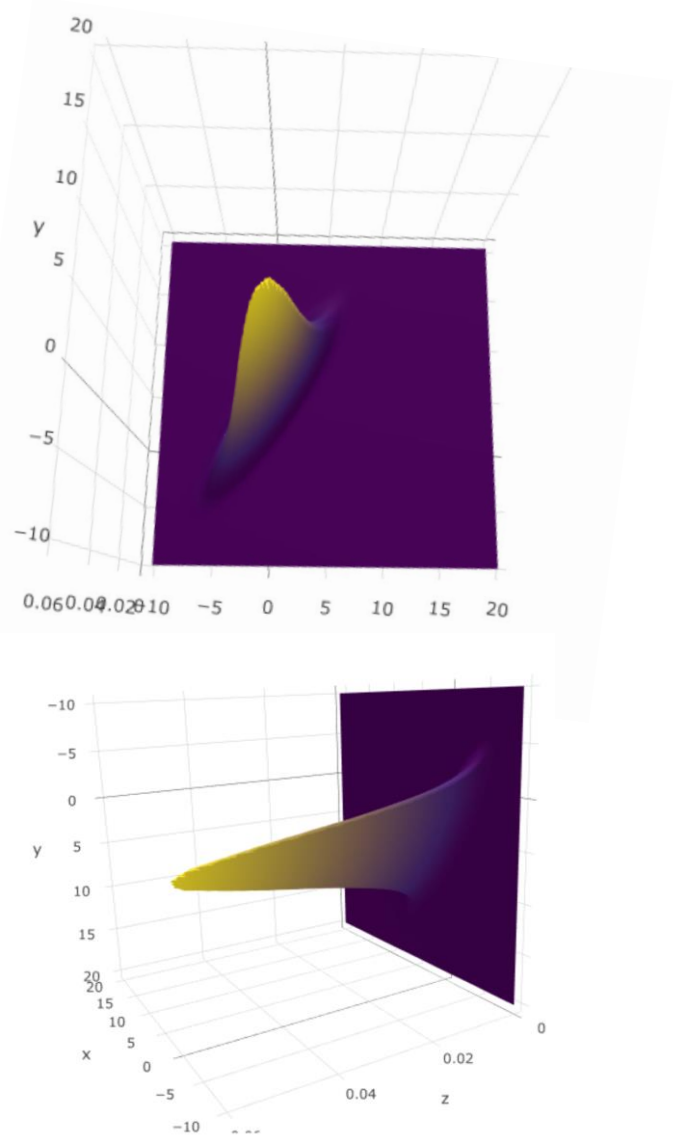
And p = cor(x_1,x_2) = $\frac{v_{12}}{\sigma^2{}_1\sigma^2{}_2}$

```
binorm<-function(x,y,muX=0,muY=0,sigmaX=1,sigmaY=1,rho=0){

  c    <- sigmaX*sigmaY*sqrt(1-rho^2)*2*pi
  d    <- 1/(1-rho^2)
  z_x  <- (x-muX)/sigmaX
  z_y  <- (y-muY)/sigmaY

  pdf <- (1/c)*exp(-0.5*d*(z_x^2+z_y^2-2*rho*z_x*z_y))

  return(pdf)}
```

(thanks to Brian Reich professor Biostatistics @ NC State)

BAUER
COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

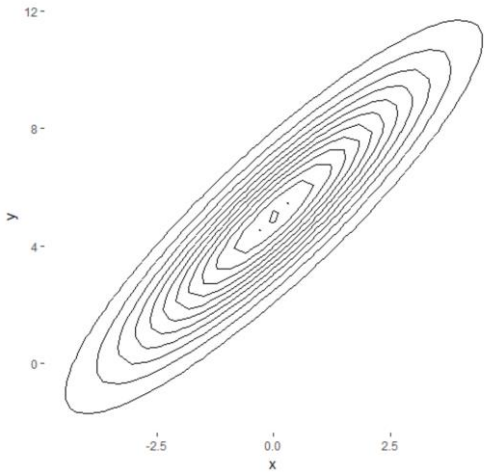
```
> binorm<-function(x,y,muX=0,muY=0,sigmaX=1,sigmaY=1,rho=0){
+
+   c    <- sigmaX*sigmaY*sqrt(1-rho^2)*2*pi
+   d    <- 1/(1-rho^2)
+   z_x  <- (x-muX)/sigmaX
+   z_y  <- (y-muY)/sigmaY
+
+   pdf <- (1/c)*exp(-0.5*d*(z_x^2+z_y^2-2*rho*z_x*z_y))
+
+   return(pdf)}
> # thanks to Brian Reich professor Biostatistics @ NC State for this pdf
>
>
> # --------------------  illustration
>
> muX    <- 0
> muY    <- 5
> sigmaX <- 2
> sigmaY <- 3
> rho    <- 0.9
>
> m    <- 100
> pts  <- seq(-10,20,length=m)
> grid <- expand.grid(pts,pts)
>
> pdf    <- binorm(grid[,1],grid[,2],muX,muY,sigmaX,sigmaY,rho)
> pdf    <- matrix(pdf,m,m)
> pdf  <- t(pdf)
>
> p <- plot_ly(x = pts, y = pts, z = pdf) %>% add_surface()
> p
```

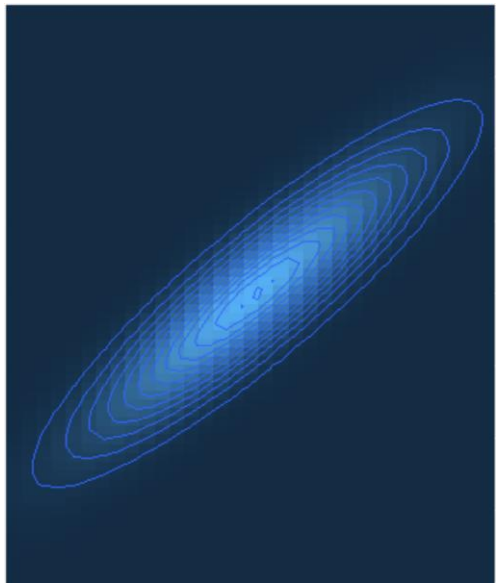


BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Binomial (revisited) and Multinomial

```
> pdf2    <- binorm(grid[,1],grid[,2],muX,muY,sigmaX,sigmaY,rho)
> pdf3 <- data.frame(cbind(x= grid[,1], y= grid[,2], pdf = pdf2))
>
> p2 <- ggplot(pdf3, aes(x, y, z=pdf)) +
+   geom_contour(color = 'black') +
+   theme(legend.position="none", panel.background = element_rect(fill = "white"))
> p2
> pdf2    <- binorm(grid[,1],grid[,2],muX,muY,sigmaX,sigmaY,rho)
> pdf3 <- data.frame(cbind(x= grid[,1], y= grid[,2], pdf = pdf2))
>
> p2 <- ggplot(pdf3, aes(x, y, z=pdf)) +
+   geom_contour(color = 'black') +
+   theme(legend.position="none", panel.background = element_rect(fill = "white"))
> p2
>
> p3 <- ggplot(pdf3, aes(x, y, z=pdf)) +
+   geom_raster(aes(fill= pdf)) +
+   geom_contour() +
+   xlim(-5, 5) + ylim(-5, 15)
> p3
```



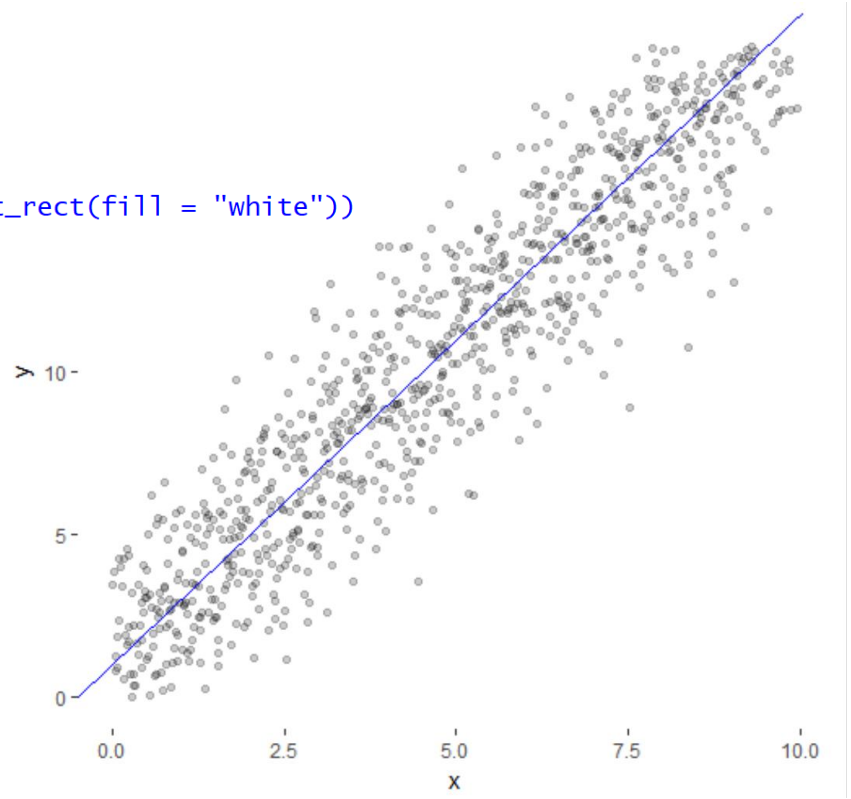


```
> MaxLike <- pdf3[which(pdf3$pdf==max(pdf3$pdf)),]
> round(MaxLike,0)
     x y pdf
4934 0 5   0
```

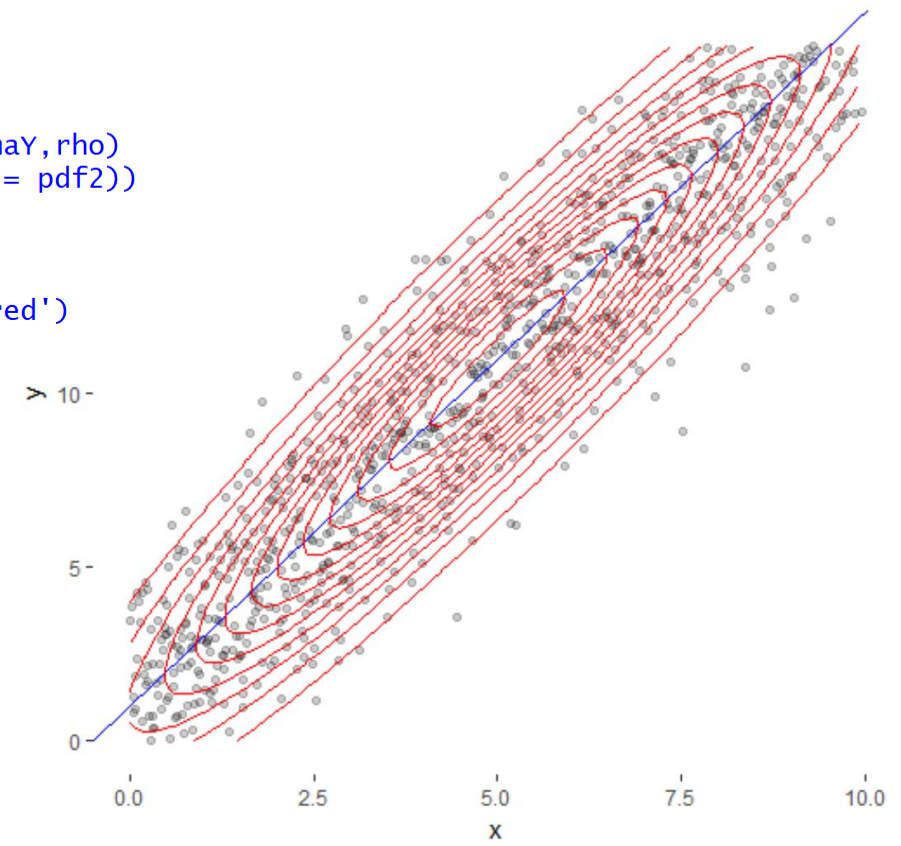
BAUER COLLEGE OF BUSINESS UNIVERSITY of HOUSTON

Exercise

```
> # beginning assumption
> x <- seq(from=0, to=10, by=.01)
> # now create a linear equation wtih intercept and slope
> b_0 <- 1
> b_1 <- 2
> # add some random noise
> noise <- rnorm(length(x), mean=0, sd=2)
>
> y <- b_0 + b_1*x + noise
>
> # put it in a dataframe
> kData <- data.frame(x, y)
> # take a look
> p <- ggplot(kData, aes(x, y))+geom_point(alpha = .2) +
+   xlim(0,10) + ylim(0,20) +
+   theme(legend.position="none", panel.background = element_rect(fill = "white"))
> p
```

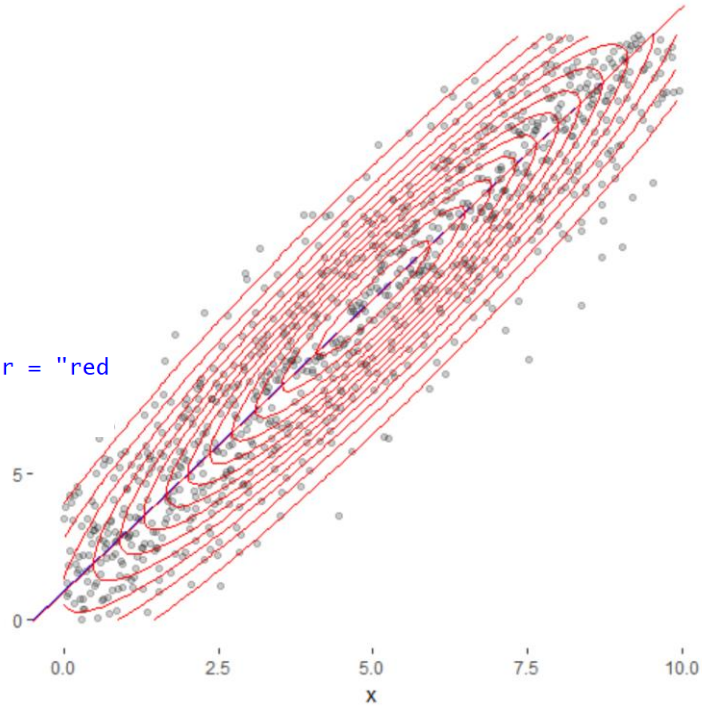


```
> # get the parameters of the distribution
>
> muX    <- mean(kData$x)
> muY    <- mean(kData$y)
> sigmaX <- sd(kData$x)
> sigmaY <- sd(kData$y)
> rho    <- cor(kData$x, kData$y, method = 'spearman')
>
> m    <- 100
> pts  <- seq(0,20,length=m)
> grid <- expand.grid(pts,pts)
>
> # create the pdf
> pdf2    <- binorm(grid[,1],grid[,2],muX,muY,sigmaX,sigmaY,rho)
> pdf3 <- data.frame(cbind(x= grid[,1], y= grid[,2], pdf = pdf2))
>
> # add pdf to plot
> p <- p +
+   geom_contour(data = pdf3, aes(x, y, z=pdf),color = 'red')
> p
```

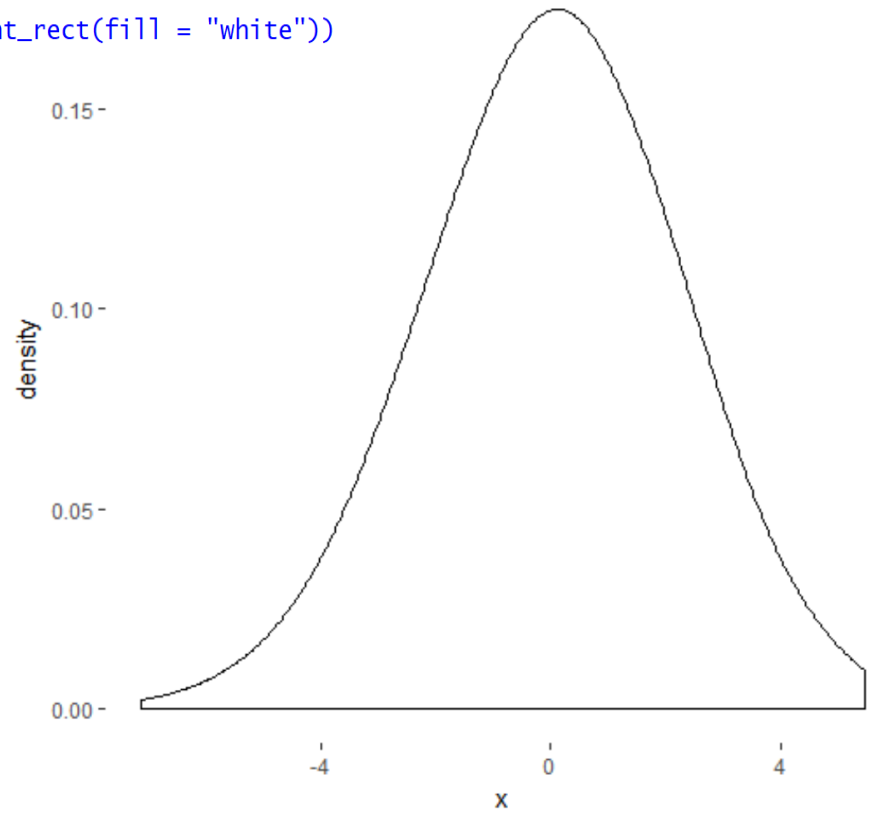



```
> MaxLike <- pdf3[which(pdf3$pdf==max(pdf3$pdf)),]
> round(MaxLike,0)
     x  y pdf
5526 5 11   0
```

```
> # now let's create a linear model
>
> mod <- lm(y~x, data = kData)
> summary(mod)

Call:
lm(formula = y ~ x, data = kData)

Residuals:
    Min      1Q  Median      3Q     Max
-7.1233 -1.4061  0.0337  1.4889  5.4893

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.97547    0.13140   7.424 2.43e-13 ***
x            1.99945    0.02275  87.877  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.08 on 999 degrees of freedom
Multiple R-squared:  0.8855,    Adjusted R-squared:  0.8853
F-statistic:  7722 on 1 and 999 DF,  p-value: < 2.2e-16

>
> p <- p +
+   geom_abline(intercept = mod$coefficients[1], slope = mod$coefficients[2], color = "red
")
> p
```



```
> # get the model residuals
> res <- data.frame(x = mod$residuals)
> ggplot(data = res, aes(x)) + geom_density(bw = 1) +
+   theme(legend.position="none", panel.background = element_rect(fill = "white"))
```



```
> # taking a look at the relationship between a range of x and y
>
> dfRegSamp <- kData %>% filter(x > 4.5 & x < 5.5)
> ggplot(data = dfRegSamp, aes(y)) + geom_density(bw = 1) +
+   theme(legend.position="none", panel.background = element_rect(fill = "white"))
```