SMLP 2022 - Intro Bayesian: Lecture notes

Daniela Palleschi

2022-09-14

Day 1

- se = σ/sqrt(n)
- Type M (for magnitude) error happens when power is low

	H0 True	H0 False
R eject	Type I α = .05	Power 1 - β
Accep t	All good	Type II β

Lecture 1.1 - Discrete random variables

In addition to r/d/pbinom, we can use r/d/pbern (for Bernoulli, but since Bernoulli = binomial these two families of functions are identical).

```
extraDistr::rbern(n = 10, prob = .5)
```

```
##  [1] 0 1 1 0 0 1 0 1 0 1
```

```
# probability for possible outcomes (0 or 1, 'cause binomial)
extraDistr::dbern(0, prob = .7)
```

```
## [1] 0.3
```

```
extraDistr::dbern(1, prob = .7)
```

```
## [1] 0.7
```

```
# probability of getting a success (=1) or failure (=0) given the probability of .7
↪  (e.g., if probability of tossing a coin and gatting a tails is .7 for some reason,
↪  then for a single coin toss what is the probability of getting a 1 or 0, given prob =
↪  .7?)
```

And cumulative probability distribution function with the bern family of functions:

```
extraDistr::pbern(1.,p=.5)
```

```
## [1] 1
```

Lecture 1.2: Discrete random variables (the binomial)

Lecture 1.3: Continuous random variables

p/d/rnorm family of functions for continuous:

```
# probability density function (PDF)
dnorm( # probability of observing
  400, # 400ms
  mean = 500, # when the true mean is 500
  sd = 100) # and the sd is 100
```

```
## [1] 0.002419707
```

```
# cumulative density function (CDF)
pnorm( # probability of observing
  400, # 400ms *or lower*
  mean = 500, # when the true mean is 500
  sd = 100) # and the sd is 100
```

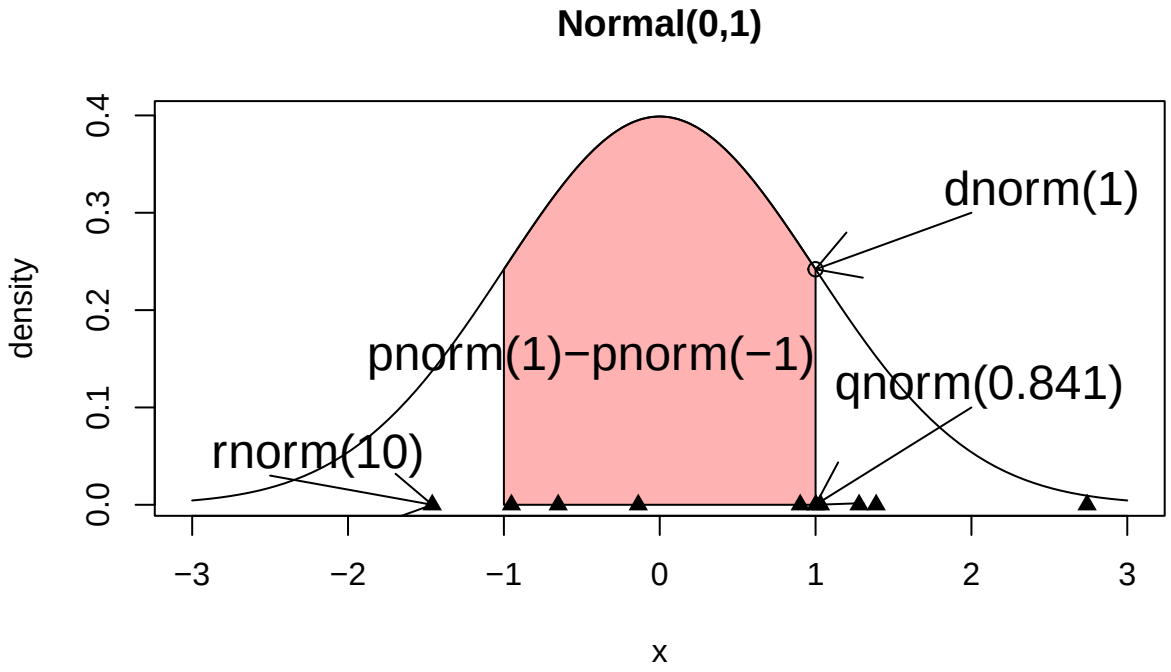
```
## [1] 0.1586553
```

```
# inverse cumulative density function (iCDF)
qnorm( # k (quantile) with a CDF of
  0.543, # 0.543
  mean = 500, # when the true mean is 500
  sd = 100) # and the sd is 100
```

```
## [1] 510.7995
```

(at this point my notes are no longer linked to a particular lecture)

Once you've generated a prior, these functions come in handy.



- common misunderstanding of the **maximum likelihood estimate (MLE)**: it doens't represent the true value of θ, because it's the MLE (best guess) for the *data you have*

– but the MLE will be closer to the true value of θ as sample size increases

Bivariate and multivariate distributions (discrete)

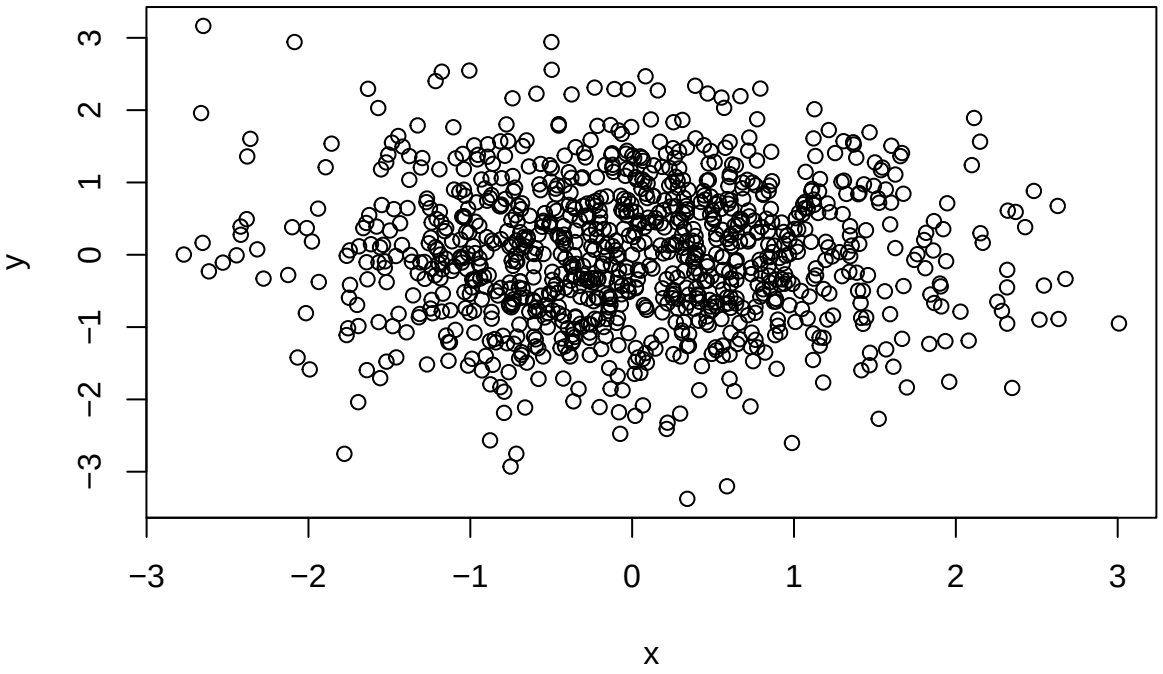
- joint probability mass function (joint PMF): for when we are considering *pairs* of observations

Bivariate and multivariate distributions (continuous)

- consider 2 random variables, X and Y, each with a normal distribution, e.g., each have the distribution *Normal*(0,1), each with 1000 observations. In this case, they're completely uncorrelated:

```
x <- rnorm(1000,0,1)
y <- rnorm(1000,0,1)

plot(x,y)
```




```
cor(x,y)
```

```
## [1] -0.01711059
```

If we want to describe a correlation between these two, we can find their **joint distribution** (bivariate distribution)

- **covariance** of two random variables = correation x sd of each variable: $\rho^*\sigma_x^*\sigma_y$

```
# calculate covariance of observed x and y:
cor(x,y)*sd(x)*sd(y)
```

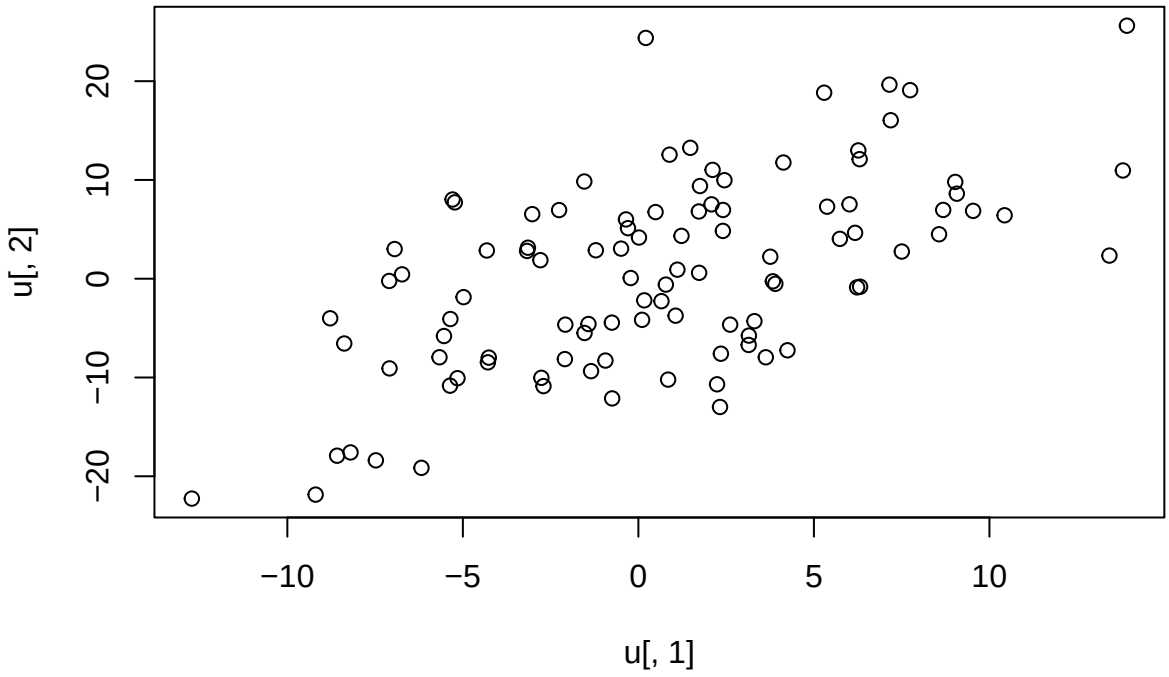
```
## [1] -0.01640798
```

```
# or simply use the cov() function
cov(x,y)
```

```
## [1] -0.01640798
```

Generate simulated bivariate data

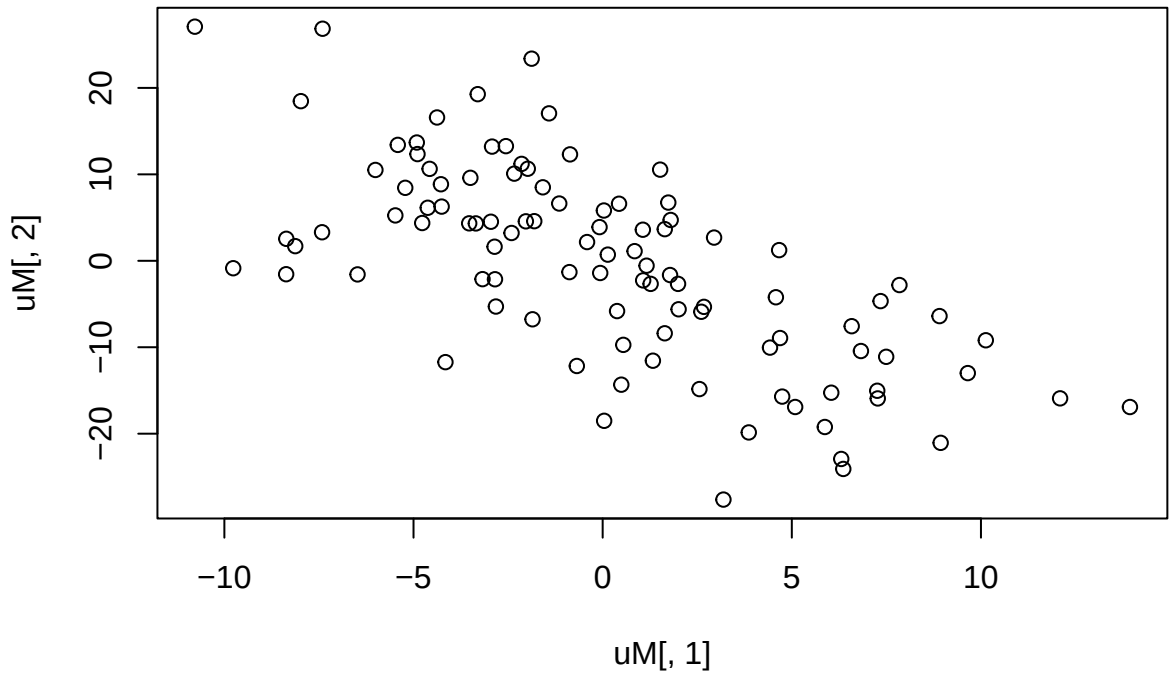
```
## define a variance-covariance matrix:
Sigma <- matrix(c(5^2, 5 * 10 * .6, 5 * 10 * .6, 10^2),
byrow = FALSE, ncol = 2
)
## generate data:
u <- MASS::mvrnorm(n = 100, # 100 data points
                   mu = c(0, 0), # means for 2 variables
                   Sigma = Sigma) # make sigma the matrix we just defined
# MASS::mvrnorm = "multivariate rnorm"
plot(u[,1],u[,2])
```




```
head(u, n = 3)
```

```
##           [,1]       [,2]
## [1,] -7.478261 -18.387624
## [2,] -1.540731   9.844360
## [3,] -4.260964  -7.985859
```

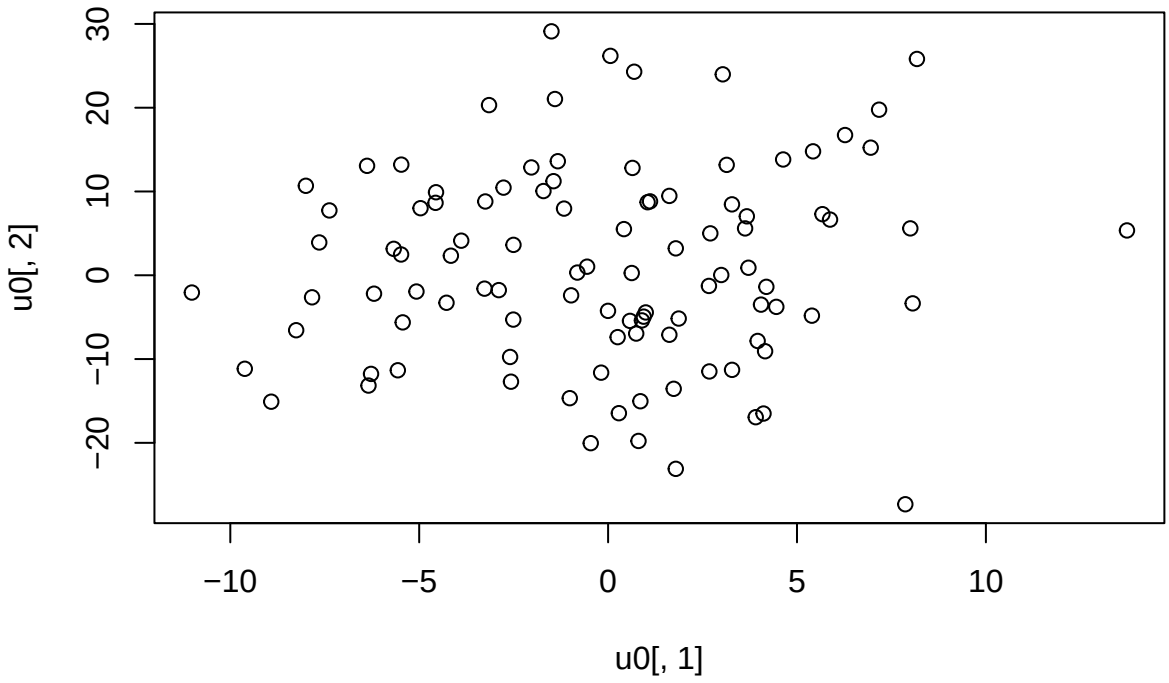
```
# and if we have a NEGATIVE correlation?
## define a variance-covariance matrix:
SigmaM <- matrix(c(5^2, 5 * 10 * -.6, 5 * 10 * -.6, 10^2),
byrow = FALSE, ncol = 2
)
## generate data:
uM <- MASS::mvrnorm(n = 100, # 100 data points
                   mu = c(0, 0), # means for 2 variables
                   Sigma = SigmaM) # make sigma the matrix we just defined
# MASS::mvrnorm = "multivariate rnorm"
plot(uM[,1],uM[,2])
```




```
head(uM, n = 3)
```

```
##           [,1]        [,2]
## [1,] 6.306397 -22.9248812
## [2,] 7.346035  -4.6604802
## [3,] 1.159650  -0.5636833
```

```
# and if we have correlation of 0?
## define a variance-covariance matrix:
Sigma0 <- matrix(c(5^2, 5 * 10 * 0, 5 * 10 * 0, 10^2),
byrow = FALSE, ncol = 2
)
## generate data:
u0 <- MASS::mvrnorm(n = 100, # 100 data points
                    mu = c(0, 0), # means for 2 variables
                    Sigma = Sigma0) # make sigma the matrix we just defined
# MASS::mvrnorm = "multivariate rnorm"
plot(u0[,1],u0[,2])
```




```
head(u0, n = 3)
```

```
##           [,1]     [,2]
## [1,] -3.250143 8.807469
## [2,] -5.674138 3.155389
## [3,]  3.625907 5.594771
```

```
library(bcogsci)
data("df_gibsonwu")
head(df_gibsonwu)
```

```
##     subj item     type   rt
## 94     1   13  obj-ext 1561
## 221    1    6 subj-ext  959
## 341    1    5  obj-ext  582
## 461    1    9  obj-ext  294
## 621    1   14 subj-ext  438
## 753    1    4 subj-ext  286
```

```
df_gibsonwu$cond <- ifelse(df_gibsonwu$type=="obj-ext",-.5,+.5)

m <- lme4::lmer(rt~cond + (1+cond|subj) + (1+cond|item), data = df_gibsonwu)
```

```
## boundary (singular) fit: see help('isSingular')
```

```
summary(m)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: rt ~ cond + (1 + cond | subj) + (1 + cond | item)
##    Data: df_gibsonwu
##
## REML criterion at convergence: 8480.1
##
## Scaled residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -1.8275 -0.4036 -0.1886  0.0575  8.4268
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  subj     (Intercept)  25725   160.4
##           cond         37966   194.8    1.00
##  item     (Intercept)  23834   154.4
##           cond         20139   141.9    1.00
##  Residual             295557   543.7
## Number of obs: 547, groups:  subj, 37; item, 15
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   547.33      53.21  10.287
## cond          119.70      67.48   1.774
##
## Correlation of Fixed Effects:
##      (Intr)
## cond 0.647
## optimizer (nloptwrap) convergence code: 0 (OK)
## boundary (singular) fit: see help('isSingular')
```

From the output:

```
# Random effects:
#  Groups   Name        Variance Std.Dev. Corr
#  subj     (Intercept)  25725   160.4
#           cond         37966   194.8    1.00
```

The Std. of the participants' means is less variable (16) than in the condition coded +0.5 (194). That's weird and is probably misestimated because the model is too complex (overparameterised for the given data). Let's try to fit a simpler model by removing the correlation parameter, which assumes the covariances are zero:

```
m <- lme4::lmer(rt~cond +
                  (1+cond||subj) +
                  (1+cond||item),
                data = df_gibsonwu)
summary(m)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: rt ~ cond + ((1 | subj) + (0 + cond | subj)) + ((1 | item) +
##     (0 + cond | item))
##    Data: df_gibsonwu
##
## REML criterion at convergence: 8498.8
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.3384 -0.3850 -0.2069  0.0194  8.8632
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subj     (Intercept)  21936   148.11
##  subj.1   cond          1196    34.58
```

```
##  item     (Intercept)  22587   150.29
##  item.1   cond         12390   111.31
##  Residual             310721   557.42
## Number of obs: 547, groups:  subj, 37; item, 15
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   548.45      51.67  10.614
## cond          120.57      56.03   2.152
##
## Correlation of Fixed Effects:
##      (Intr)
## cond 0.003
```

This singular fit error disapperas and the variance Std.Dev. makes more sense.

Ch. 1 Exercises

Exercise 1.1 Practice using the pnorm function - Part 1 Given a normal distribution with mean 500 and standard deviation 100, use the pnorm function to calculate the probability of obtaining values between 200 and 800 from this distribution.

```
pnorm(800, 500, 100) - pnorm(200, 500, 100)
```

```
## [1] 0.9973002
```

Exercise 1.2 Practice using the pnorm function - Part 2 Calculate the following probabilities. Given a normal distribution with mean 800 and standard deviation 150, what is the probability of obtaining:

a score of 700 or less a score of 900 or more a score of 800 or more

```
pnorm(700, 800, 150)
```

```
## [1] 0.2524925
```

```
pnorm(900, 800, 150, lower.tail=F)
```

```
## [1] 0.2524925
```

```
pnorm(800, 800, 150, lower.tail=F)
```

```
## [1] 0.5
```

Exercise 1.3 Practice using the pnorm function - Part 3 Given a normal distribution with mean 600 and standard deviation 200, what is the probability of obtaining:

a score of 550 or less. a score between 300 and 800. a score of 900 or more.

```
pnorm(550, 600, 200)
```

```
## [1] 0.4012937
```

```
pnorm(800, 600, 200) - pnorm(300, 600, 200)
```

```
## [1] 0.7745375
```

```
pnorm(900, 600, 200, lower.tail = F)
```

```
## [1] 0.0668072
```

Exercise 1.4 Practice using the qnorm function - Part 1 Consider a normal distribution with mean 1 and standard deviation 1. Compute the lower and upper boundaries such that:

the area (the probability) to the left of the lower boundary is 0.10. the area (the probability) to the left of the upper boundary is 0.90.

```
qnorm(c(.9,.1), 1,1)
```

```
## [1]  2.2815516 -0.2815516
```

Exercise 1.5 Practice using the qnorm function - Part 2 Given a normal distribution with mean 650 and standard deviation 125. There exist two quantiles, the lower quantile q1 and the upper quantile q2, that are equidistant from the mean 650, such that the area under the curve of the normal between q1 and q2 is 80%. Find q1 and q2.

```
qnorm(c(.1,.9),650,125)
```

```
## [1] 489.8061 810.1939
```

Exercise 1.6 Practice getting summaries from samples - Part 1 Given data that is generated as follows:

```
data_gen1 <- rnorm(1000, 300, 200)
```

Calculate the mean, variance, and the lower quantile q1 and the upper quantile q2, that are equidistant and such that the range of probability between them is 80%.

```
mean(data_gen1)
```

```
## [1] 310.1582
```

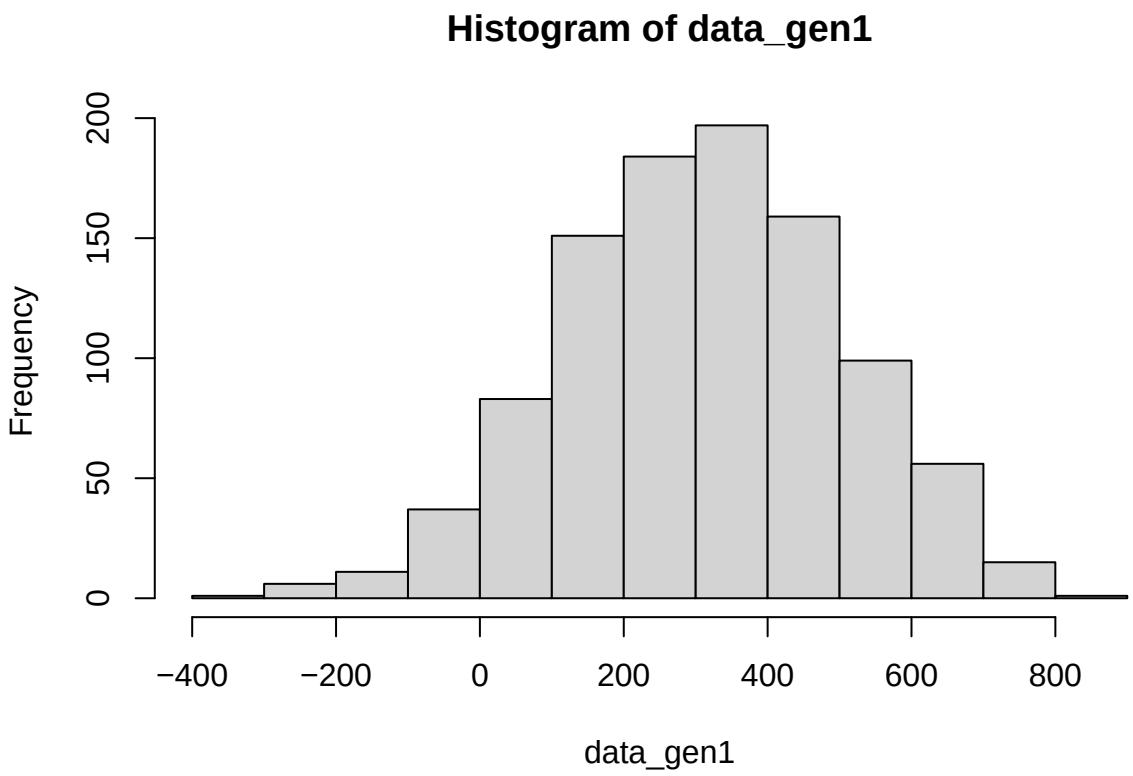
```
sd(data_gen1)
```

```
## [1] 194.0528
```

```
qnorm(c(.1,.9),mean(data_gen1), sd(data_gen1))
```

```
## [1]  61.46951 558.84695
```

```
hist(data_gen1)
```



Exercise 1.7 Practice getting summaries from samples - Part 2. This time we generate the data with a truncated normal distribution from the package extraDistr. The details of this distribution will be discussed later in section 4.1 and in the Box 4.1, but for now we can treat it as an unknown generative process:

```
data_gen2 <- extraDistr::rtnorm(1000, 300, 200, a = 0)
```

Calculate the mean, variance, and the lower quantile q1 and the upper quantile q2, that are equidistant and such that the range of probability between them is 80%.

```
mean(data_gen2)
```

```
## [1] 332.3941
```

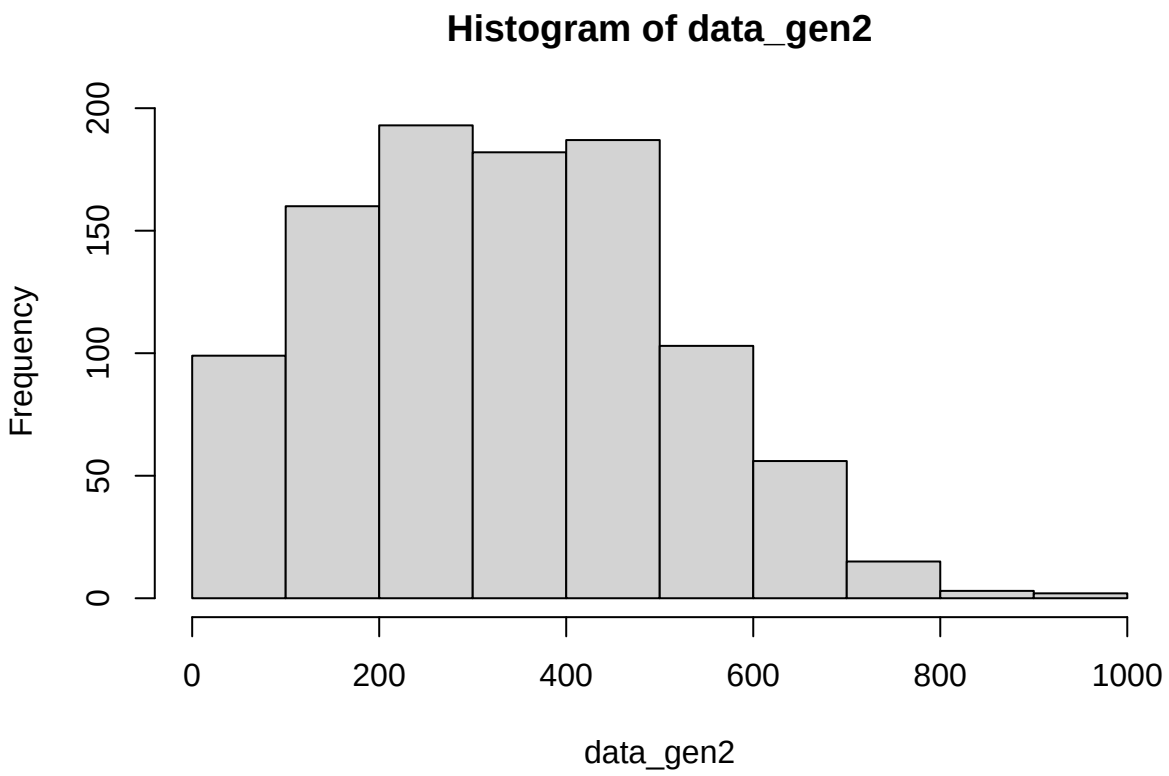
```
sd(data_gen2)
```

```
## [1] 176.5025
```

```
qnorm(c(.1,.9), mean(data_gen2), sd(data_gen2))
```

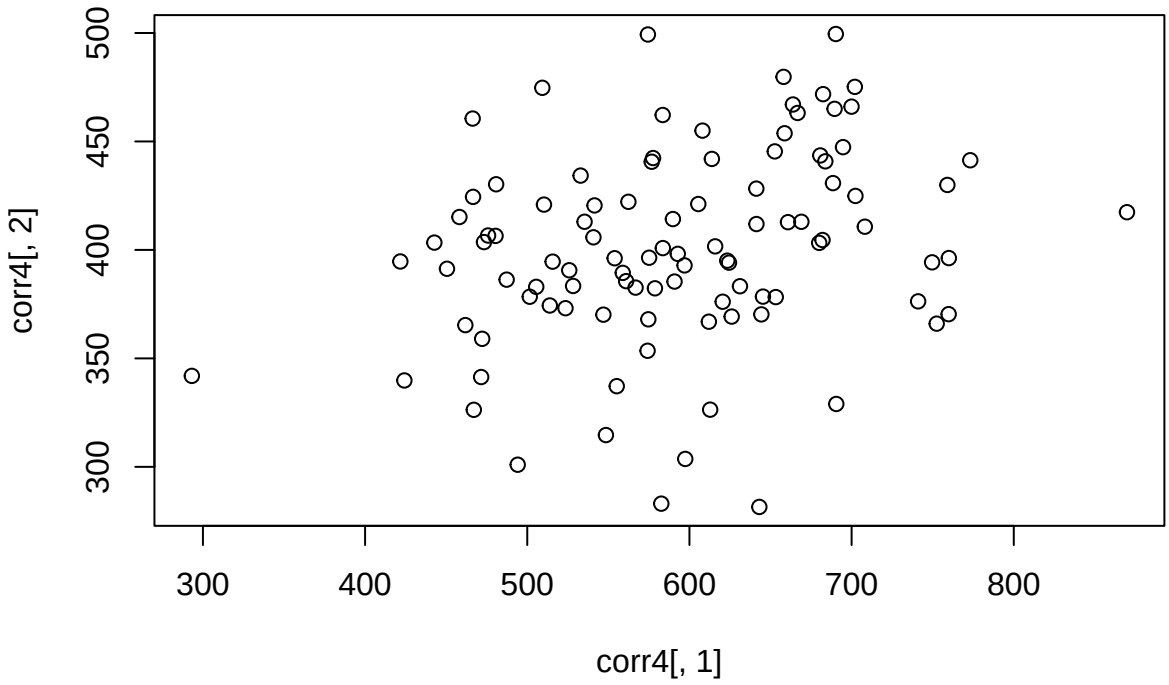
```
## [1] 106.1971 558.5911
```

```
hist(data_gen2)
```



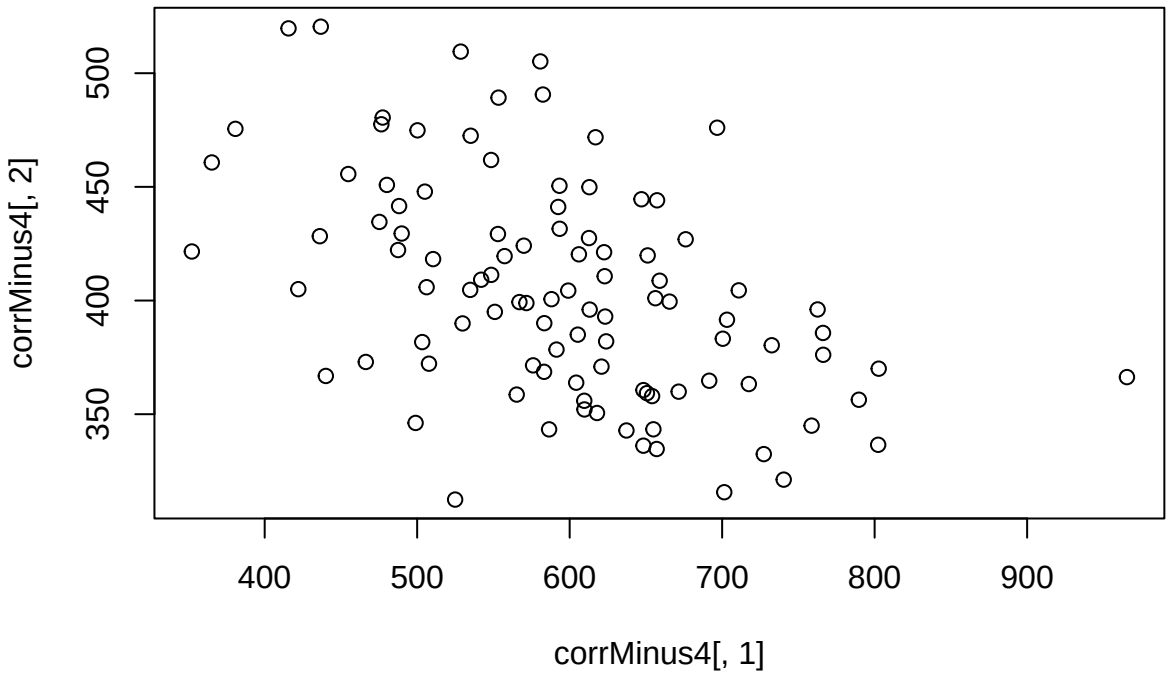
Exercise 1.8 Practice with a variance-covariance matrix for a bivariate distribution. Suppose that you have a bivariate distribution where one of the two random variables comes from a normal distribution with mean $\mu_X = 600$ and standard deviation $\sigma_X = 100$, and the other from a normal distribution with mean $\mu_Y = 400$ and standard deviation $\sigma_Y = 50$. The correlation ρ_{XY} between the two random variables is 0.4. Write down the variance-covariance matrix of this bivariate distribution as a matrix (with numerical values, not mathematical symbols), and then use it to generate 100 pairs of simulated data points. Plot the simulated data such that the relationship between the random variables X and Y is clear.

```
## define a variance-covariance matrix:
Sigma4 <- matrix(c(100^2, 100 * 50 * .4, # square sdX = 5^2, mean X * mean Y * their corr
                   100 * 50 * .4, 50^2), # mean X * mean Y * their corr, square sdY =
                   ↪  10^2,
byrow = FALSE, ncol = 2
)
## generate data:
corr4 <- MASS::mvrnorm(n = 100, # 100 data points
                    mu = c(600, 400), # means for 2 variables
                    Sigma = Sigma4) # make sigma the matrix we just defined
# MASS::mvrnorm = "multivariate rnorm"
plot(corr4[,1],corr4[,2]); corr4_plot <- recordPlot()
```

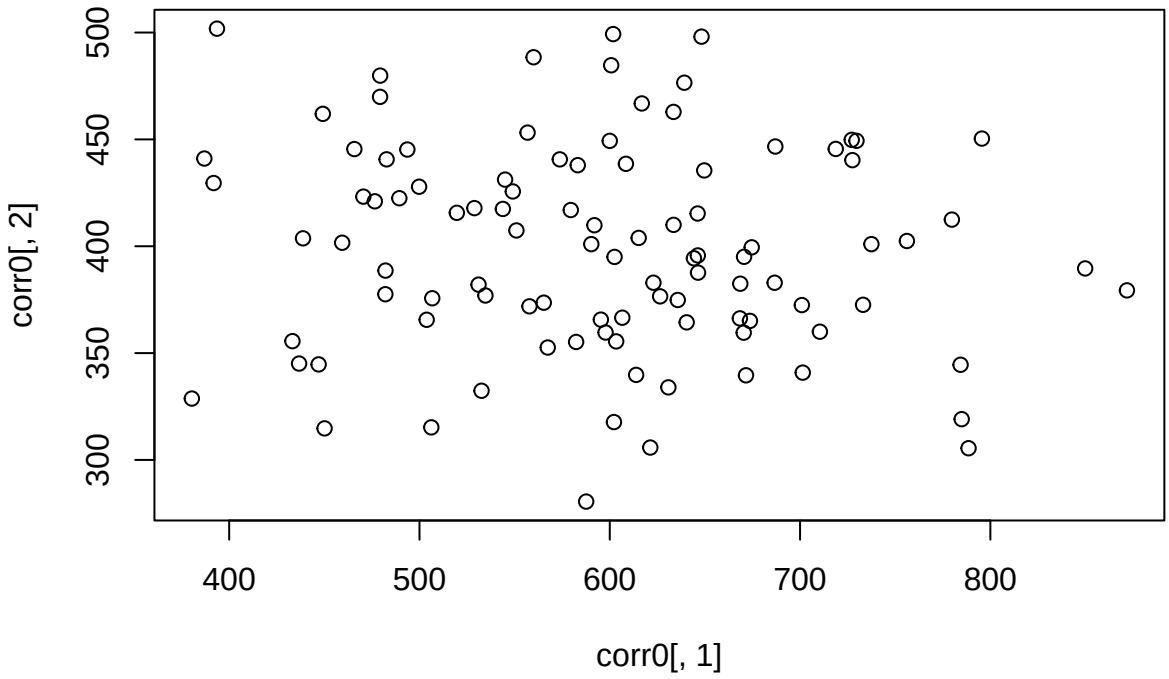


Generate two sets of new data (100 pairs of data points each) with correlation - 0.4 and 0, and plot these alongside the plot for the data with correlation 0.4.

```
## define a variance-covariance matrix:
SigmaMinus4 <- matrix(c(100^2, 100 * 50 * -.4, # square sdX = 5^2, mean X * mean Y *
↪  their corr
                  100 * 50 * -.4, 50^2), # mean X * mean Y * their corr, square sdY =
                  ↪  10^2,
byrow = FALSE, ncol = 2
)
## generate data:
corrMinus4 <- MASS::mvrnorm(n = 100, # 100 data points
                  mu = c(600, 400), # means for 2 variables
                  Sigma = SigmaMinus4) # make sigma the matrix we just defined
# MASS::mvrnorm = "multivariate rnorm"
plot(corrMinus4[,1],corrMinus4[,2]); corrMinus4_plot <- recordPlot()
```



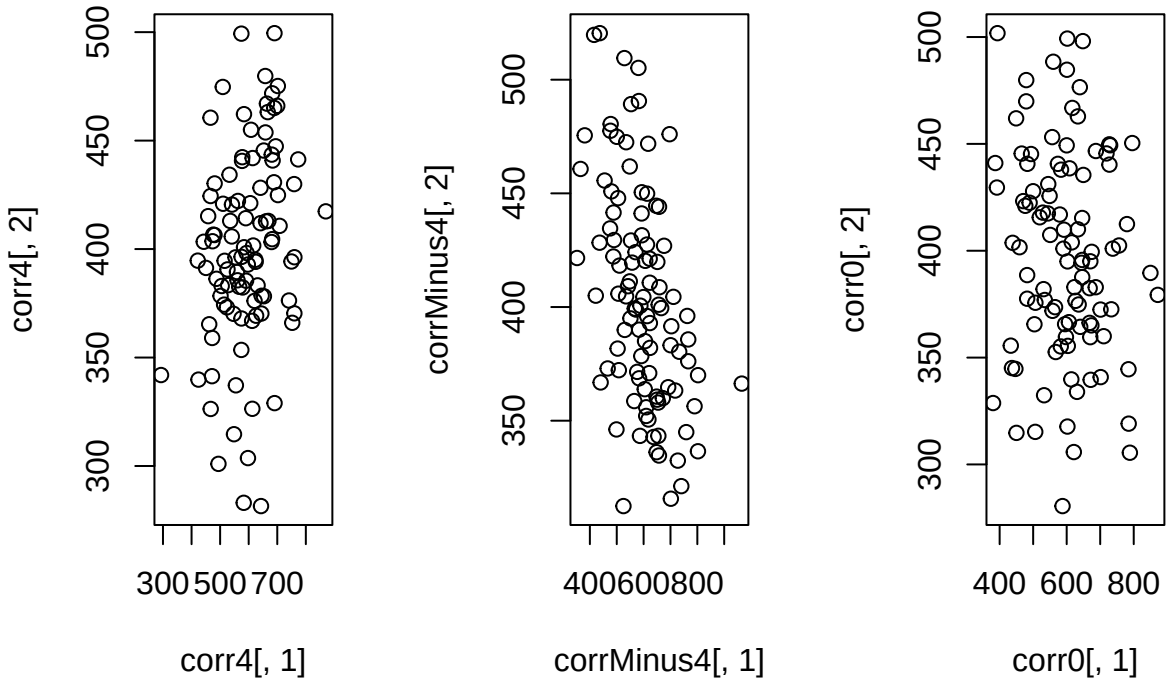

```
## define a variance-covariance matrix:
Sigma0 <- matrix(c(100^2, 100 * 50 * 0, # square sdX = 5^2, mean X * mean Y * their corr
                   100 * 50 * 0, 50^2), # mean X * mean Y * their corr, square sdY = 10^2,
                   ↪
byrow = FALSE, ncol = 2
)
## generate data:
corr0 <- MASS::mvrnorm(n = 100, # 100 data points
                       mu = c(600, 400), # means for 2 variables
                       Sigma = Sigma0) # make sigma the matrix we just defined
# MASS::mvrnorm = "multivariate rnorm"
plot(corr0[,1],corr0[,2]); corr0_plot <- recordPlot()
```




```
library(gridGraphics)
```

```
## Loading required package: grid
```

```
ggpubr::ggarrange(corr4_plot,corrMinus4_plot,corr0_plot,
                  nrow=1)
```



Day 2

Yesterday we look at the examples of *discrete* variables and *continuous* variables. In Bayesian, we're interested in how uncertain we are about the possible true values are: **uncertainty quantification** (uncertainty of μ and σ)

- probability density function (PMF) on the parameter is always the output of Bayesian analysis
- uniform distribution: rectangle; certainty is constant cause we are equally unsure

Lecture 1.2: Bayes' Rule

Event A: the streets are wet Event B: it is raining

Q: What's the probability of the streets being wet (B) *given that* it is raining? **P(A|B)**

P(A|B) = P(A,B)/P(B), given P(B)>0 P(A,B) = P(B,A)

- but in research we don't work with discrete events
- instead of *P(A/B)*, we look at *f(y/θ)*, which represents **probabiity density function** for a given *distribution*
- given the data y, we assume some density distribution that generated the data; basically y is our priors, θ represents posterior distribution of parameters from given data
- f(y) ('f of y') =

```
# density of a data point from a particular normal distribution
dnorm(0.1,0,1)
```

```
## [1] 0.3969525
```

```
# what is the density on the curve at data point 0.1, given a mean of 0 and sd of 1?
```

- μ has some prior, as does σ which has a uniform distribution

```
# produce random datapoint of sigma given a uniform distribution
runif(1, min=0,max=1)
```

```
## [1] 0.6842169
```

```
k = 46  # 46 succesess
n = 100 # 100 trials
theta <- seq(0,1,by=.01)
options(scipen = 0); dbinom(k,n,theta); options(scipen=999)
```

```
##   [1] 0.000000e+00 4.269888e-64 1.736616e-50 1.257093e-42 4.013489e-37
##   [6] 6.543511e-33 1.621726e-29 1.093214e-26 2.836599e-24 3.544105e-22
##  [11] 2.484342e-20 1.089532e-18 3.239795e-17 6.943042e-16 1.124403e-14
##  [16] 1.428683e-13 1.467968e-12 1.250211e-11 9.007395e-11 5.584350e-10
##  [21] 3.022422e-09 1.445661e-08 6.175439e-08 2.377363e-07 8.313177e-07
##  [26] 2.658671e-06 7.823531e-06 2.129529e-05 5.386916e-05 1.271671e-04
##  [31] 2.811822e-04 5.842581e-04 1.144185e-03 2.117369e-03 3.711237e-03
##  [36] 6.174009e-03 9.766739e-03 1.471584e-02 2.115011e-02 2.903347e-02
##  [41] 3.811036e-02 4.788307e-02 5.763615e-02 6.651309e-02 7.363639e-02
##  [46] 7.824864e-02 7.984344e-02 7.825541e-02 7.368743e-02 6.666918e-02
##  [51] 5.795840e-02 4.840970e-02 3.884155e-02 2.992887e-02 2.213868e-02
##  [56] 1.571359e-02 1.069571e-02 6.976794e-03 4.357780e-03 2.603975e-03
##  [61] 1.487007e-03 8.105450e-04 4.211607e-04 2.082928e-04 9.788807e-05
```

```
##  [66] 4.363196e-05 1.840766e-05 7.333472e-06 2.751850e-06 9.698523e-07
##  [71] 3.200174e-07 9.851262e-08 2.818023e-08 7.457801e-09 1.816920e-09
##  [76] 4.052234e-10 8.221447e-11 1.506581e-11 2.473389e-12 3.604160e-13
##  [81] 4.611849e-14 5.118247e-15 4.855885e-16 3.872130e-17 2.543561e-18
##  [86] 1.343738e-19 5.545729e-21 1.725610e-22 3.873523e-24 5.932821e-26
##  [91] 5.771270e-28 3.244259e-30 9.272894e-33 1.126226e-35 4.468531e-39
##  [96] 3.852849e-43 3.646130e-48 1.052358e-54 5.225588e-64 4.627377e-80
## [101] 0.000000e+00
```

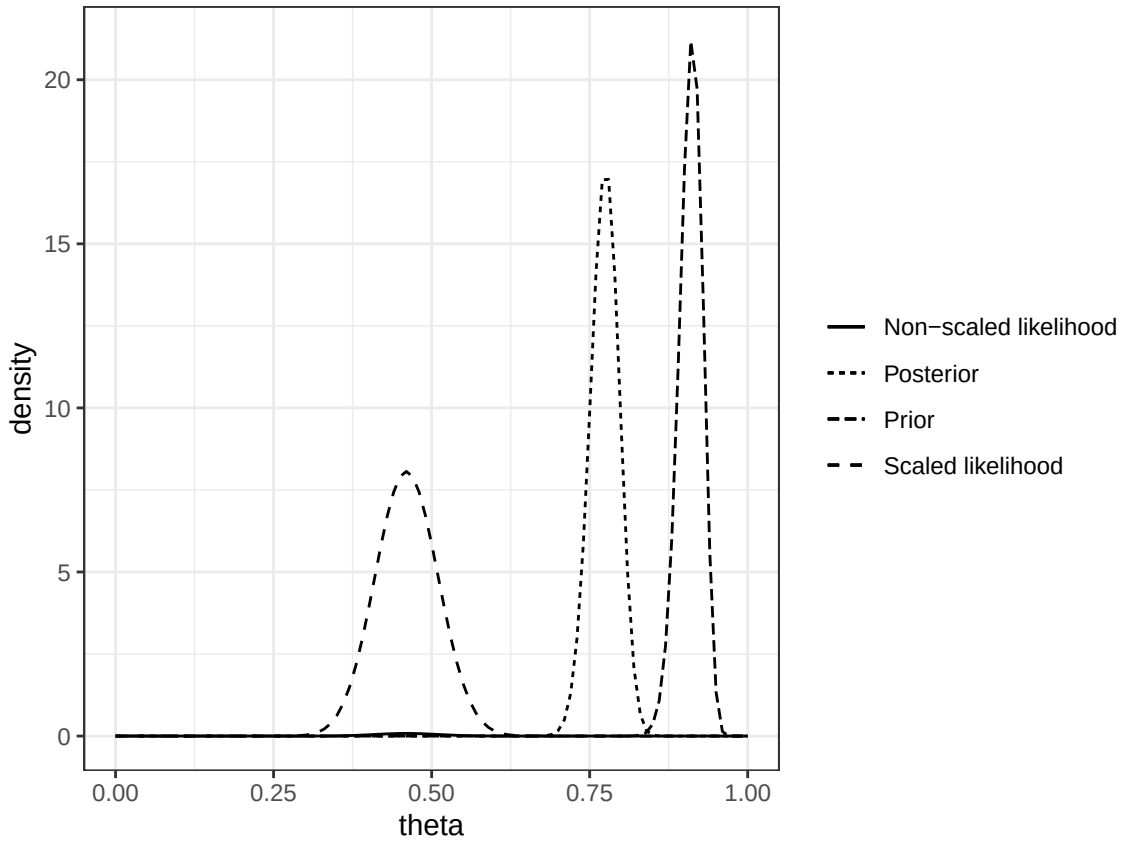
Beta distribution

- ranges 0-1
- can change the shape of the distribution by changing the parameters (a and b), in R called shape1 and shape2
- a = number of successes you expect a prior
- b = number of failures
- the size of the parameters also control trial uncertainty;the higher the numbers, the tighter (i.e., narrower) the curve
- if we don't have much prior knowledge, we can set a and b both to 1, which would give us a uniform distribution, and an **uniformative** prior
- if we have stronger prior knowledge/a strong belief that θ has a particular range of values, we can set a and b to have higher values

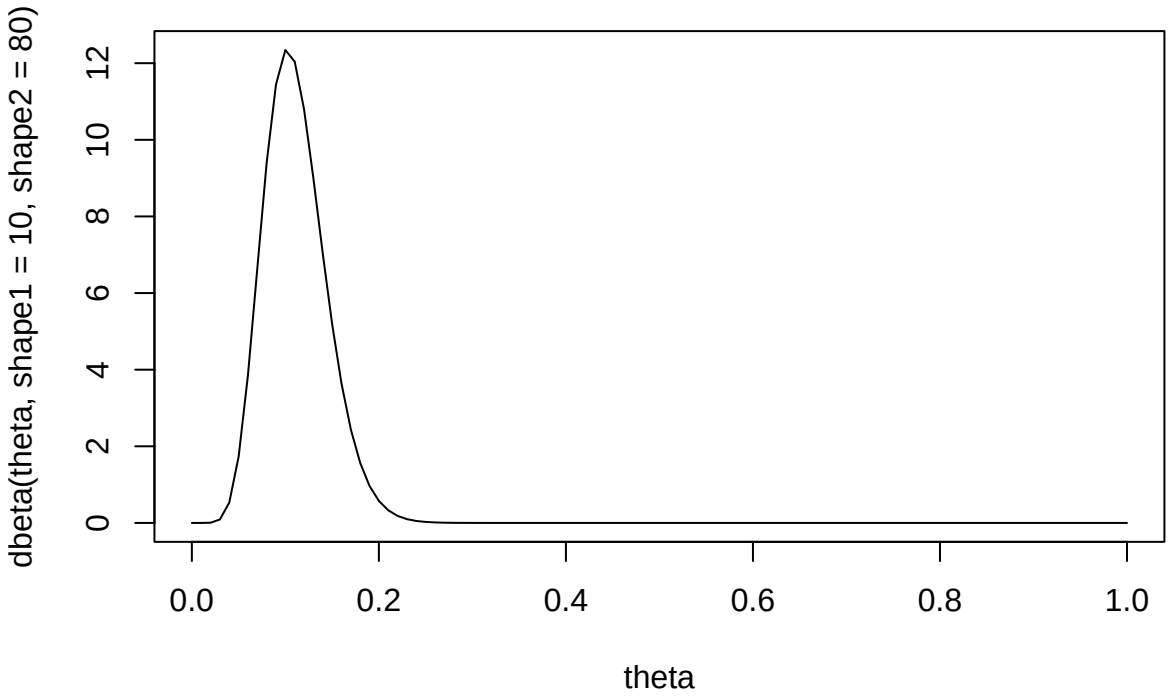
```r
x <- 46
n <- 100
## Prior specification:
a <- 210
b <- 21
binom_lh <- function(theta) {
dbinom(x=x, size =n, prob = theta)
}
## normalizing constant:
K <- 1/integrate(f = binom_lh, lower = 0, upper = 1)$value
binom_scaled_lh <- function(theta) K * binom_lh(theta) # compute likelihood, scaling it
↪  with normalising constant
```

```r
library(ggplot2)
## Likelihood
p_beta <- ggplot(data = tibble::tibble(theta = c(0, 1)), aes(theta)) +
  stat_function(
    fun = dbeta,
    args = list(shape1 = a, shape2 = b),
    aes(linetype = "Prior")
  ) +
  ylab("density") +
  stat_function(
    fun = dbeta,
    args = list(shape1 = x + a, shape2 = n - x + b), aes(linetype = "Posterior")
  ) +
  stat_function(
    fun = binom_lh,
    aes(linetype = "Non-scaled likelihood")
```

```
  ) +
  stat_function(
    fun = binom_scaled_lh,
    aes(linetype = "Scaled likelihood")
  ) +
  theme_bw() +
  theme(legend.title = element_blank())
p_beta
```




```
plot(theta,
     dbeta(theta,shape1=10,shape2=80), # distribution given priors of 10 succeses & 80
     ↪  failures
     type="l")
```



Poisson distribution

- from Lecture 2.3
- can be used to model e.g., number of fixations in a region

```
# simulate 10 random data points with Poisson distribution
# lambda = expected rate of 'successes', e.g., number of regressions into a region
(x<-rpois(n=10,lambda=3))
```

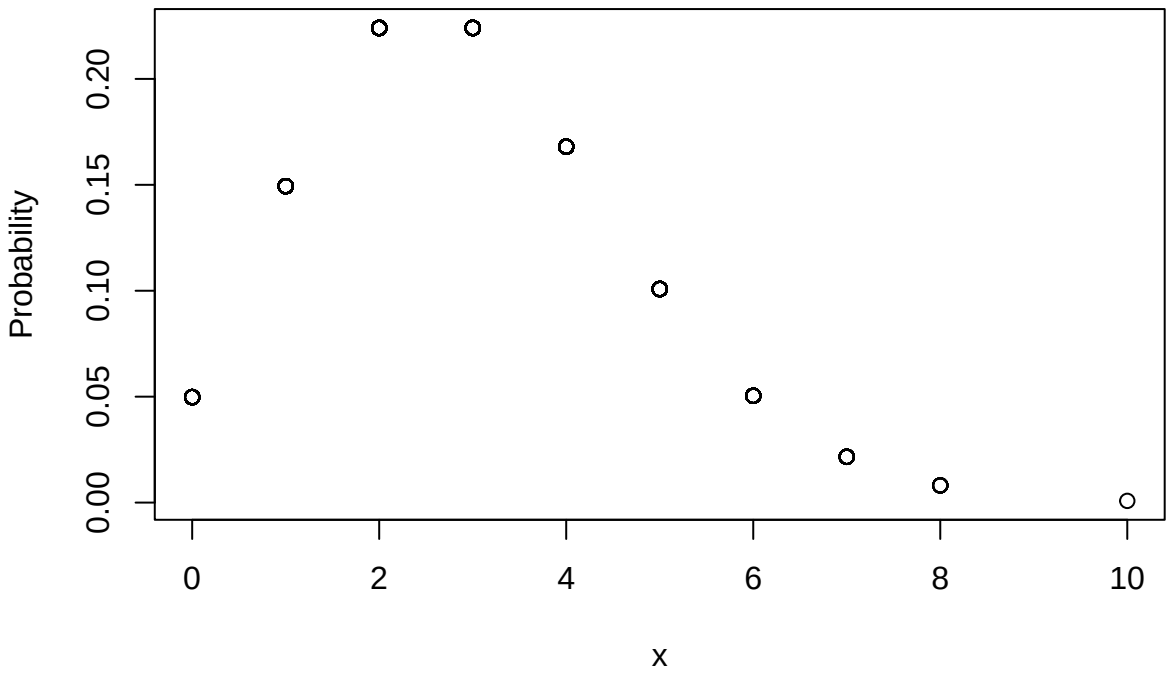
```
##  [1] 2 6 3 1 1 2 1 4 6 2
```

```
(x<-rpois(n=1000,lambda=3))
```

```
##    [1]  2  5  2  6  3  3  3  2  1  5  3  1  0  3  2  3  2  4  4  4  5  2  3  3
##   [25]  6  3  1  3  2  3  0  1  2  3  6  7  2  4  3  3  4  1  8  3  1  2  7  5
##   [49]  3  2  2  3  4  6  4  1  2  2  4  4  2  1  1  0  4  1  2  2  3  5  4  3
##   [73]  3  1  7  0  3  4  1  2  1  4  1  4  3  4  2  5  3  5  5  3  1  2  2  1
##   [97]  0  5  1  1  3  4  5  4  4  3  4  7  2  3  3  1  3  4  2  2  4  4  5  3
##  [121]  2  1  2  5  5  5  3  6  3  4  1  2  3  2  4  5  3  2  4  1  5  0  3  4
##  [145]  3  3  3  3  3  5  3  2  5  4  4  6  5  1  8  7  1  4  2  5  2  2  2  0
##  [169]  3  4  3  1  2  4  3  2  4  3  1  4  0  6  4  4  1  3  4  1  3  5  1  2
##  [193]  5  6  3  2  3  3  5  1  1  3  4  6  4  3  1  5  2  5  3  1  2  2  7  4
##  [217]  2  3  3  1  0  7  4  1  1  5  5  0  2  0  7  2  1  5  0  2  3  3  4  2
##  [241]  3  5  3  1  2  3  2  3  2  3  4  4  8  1  1  4  6  2  3  1  3  0  0  1
##  [265]  0  4  2  3  6  2  3  0  1  3  2  2  3  5  4  2  5  2  3  2  5  3  1  4
##  [289]  3  4  4  4  1  1  6  3  6  2  2  1  2  4  1  2  1  5  1  2  2  4  6  5
##  [313]  2  2  1  4  4  1  5  4  5  8  2  4  2  5  4  2  4  5  1  0  4  4  2  4
##  [337]  4  3  4  2  1  5  1  2  2  3  4  0  3  2  2  2  1  1  3  7  7  2  6  5
##  [361]  5  1  2  1  5  3  3  4  1  4  4  3  5  1  1  4  2  1  5  0  3  2  4  0
##  [385]  4  2  0  2  0  3  2  2  3  1  0  3  1  2  6  1  3  3  7  1  5  1  6  4
##  [409]  4  4  4  3  3  5  4  4  2  1  2  4  2  1  2  2  6  6  3  4  3  4  4  6
##  [433]  4  4  6  1  4  4  4  2  5  4  4  0  1  6  4  7  3  3  2  1  3  3  3  2
##  [457]  4  0  1  2  3  3  1  1  4  4  0  3  2  1  6  1  6  4  4  4  2  3  2  1
```

```
##  [481]  1  4  2  1  2  5  3  5  0  3  0  1  2  5  4  5  5  2  2  3  7  1  3  3
##  [505]  3  2  4  3  5  2  2  3  1  1  2  4  0  4  3  3  0  5  1  8  2  1  1  3
##  [529]  3  5  7  2  3  2  1  1  5  1  3  4  4  5  3  3  5  2  1  3  4  4  6  3
##  [553]  8  1  2  4  3  0  1  3  6  1  7  5  4  4  3  4  4  2  2  2  3  4  2  4
##  [577]  1  1  4  2  1  5  6  2  4  5  3  4  5  5  5  0  3  1  3  2  1  4  4  1
##  [601]  3  1  4  3  1  5  3  5  1  5  5  4  4  4  2  1  4  4  3  1  2  7  1  1
##  [625]  6  1  4  1  5  5  1  1  2  1  2  6  2  6  0  2  6  5  3  5  2  2  2  0
##  [649]  3  2  3  4  6  1  4  1  5  1  2  3  1  1  1  1  2  8  3  4  2  5  6  5
##  [673]  2  3  2  3  3  3  4  5  5  4  3  7  3  0  4  3  2  4  4  4  3  1  2  3
##  [697]  1  5  6  4  5  0  2  5  3  2  3  2  3  4  4  1  3  2  8  3  2  3  4  4
##  [721]  4  1  3  3  1  2  3  1  5  4  3  1  4  5  3  1  2  3  6  1  1  2  2  2
##  [745]  7  2  2  2  2  3  1  1  5  6  2  0  5  1  5  2  5  3  1  4  2  4  6  2
##  [769]  2  5  1  3  4  4  4  1  4  4  2  2  4  7  4  0  3  1  4  4  1  4  2  3
##  [793]  4  3  6  3  6  2  3  1  1  1  4  3  5  2  2  3  1  4  4  1  3  2  1  2
##  [817]  8  0  3  2  3  3  5  6  6  2  3  3  4  1  1  4  2  3  4  2  4  5  7  4
##  [841]  6  4  2  3  1  3  1  6  5  7  5  3  5  2  4  1  1  5  3  3  1  7  1  1
##  [865]  3  3  4  3  4  3  6  2  3  1  4  4  5  1  3  0  0  3  4  0  5  4  3  1
##  [889]  4  2  2  5  1  5  0  1  3  1  5  3  0  6  2  4  5  0  4  3  2  2  1  5
##  [913]  0  3  2  2  1  7  1  1  2  4  3  1  4  0  3  2  2  2  2  2  1  4  2  1
##  [937]  4  3  4  0  2  3  3  3  2  0  3  2  3  3  4  2  4  4  3  1  3  3  2  4
##  [961]  1  2  5  4  1  3  3  3  2  5  2  3  4  0 10  5  4  3  5  5  3  4  3  1
##  [985]  1  2  7  1  2  2  0  3  4  4  2  5  4  1  3  2
```

```
plot(x,dpois(x,lambda=3),
     ylab="Probability")
```



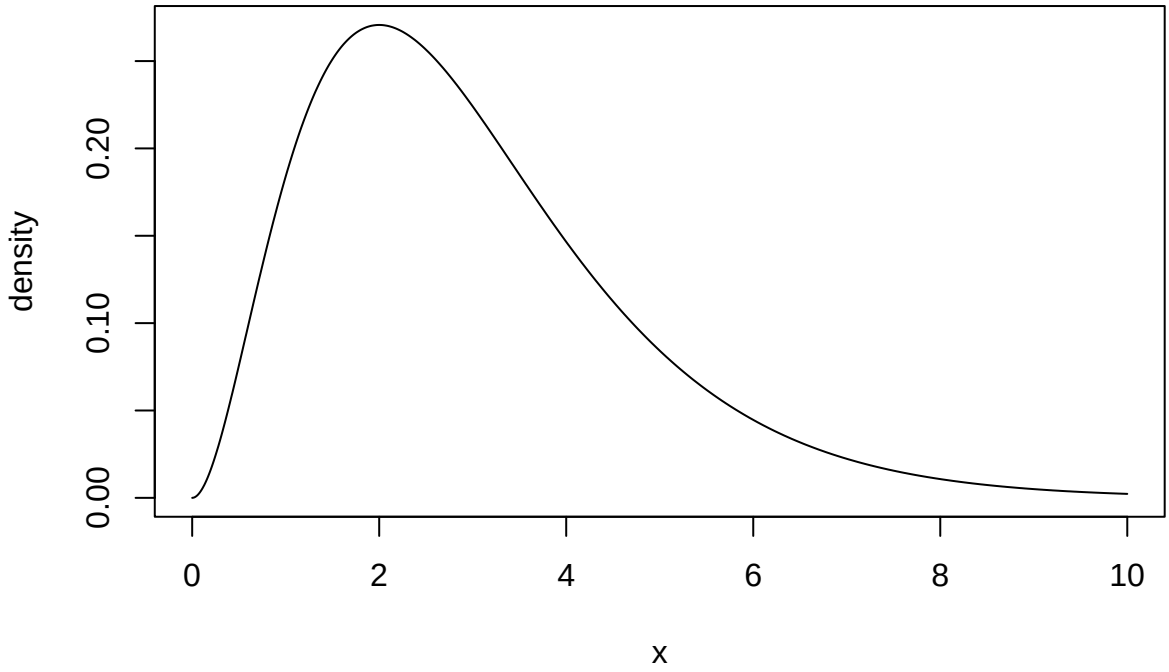
Gamma distribution

- in R, the a and b parameters are called *shape* and *scale*
- mean of gamma distr. = a/b
- variance of gramma distr. = a/b^2

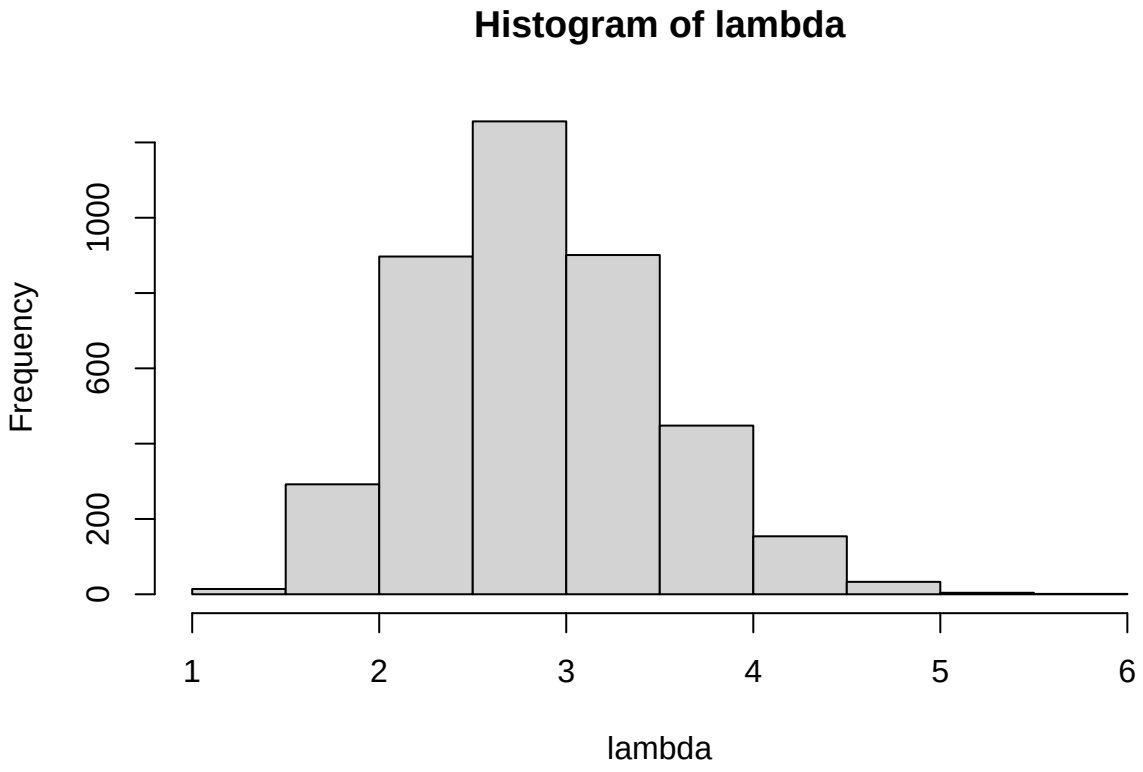
```r
round(rgamma(n=10, shape=3,scale=1),2)
```

```
##  [1] 1.13 2.99 2.71 2.40 1.72 4.30 5.34 2.81 1.37 3.08
```

```r
# plot gamma distribution given your parameter values
x<-seq(0,10,by=0.01)
plot(x,dgamma(x,shape=3,scale=1),type="l",
     ylab="density")
```




```r
lambda <- rgamma(4000,shape=20,rate=7) # sample from the posterior distribution
hist(lambda)
```



Computational Bayesian

```
library(brms)
```

```
## Loading required package: Rcpp
```

```
## Loading 'brms' package (version 2.17.0). Useful instructions
## can be found by typing help('brms'). A more detailed introduction
## to the package is available through vignette('brms_overview').
```

```
##
## Attaching package: 'brms'
```

```
## The following object is masked from 'package:stats':
##
##     ar
```

```
fit_press <- brm(rt ~ 1,
data = df_spacebar,
family = gaussian(),
prior = c(
prior(uniform(0, 60000), class = Intercept, lb = 0,
ub = 60000),
prior(uniform(0, 2000), class = sigma, lb = 0,
ub = 2000)
),
chains = 4,
iter = 2000,
warmup = 1000
)
```

```
## Compiling Stan program...
## Start sampling
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000161 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.61 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.068 seconds (Warm-up)
## Chain 1:                0.058 seconds (Sampling)
## Chain 1:                0.126 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.9e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.19 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.063 seconds (Warm-up)
## Chain 2:                0.052 seconds (Sampling)
## Chain 2:                0.115 seconds (Total)
## Chain 2:
##
```

```
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.7e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.079 seconds (Warm-up)
## Chain 3:                0.065 seconds (Sampling)
## Chain 3:                0.144 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.5e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.15 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.06 seconds (Warm-up)
## Chain 4:                0.059 seconds (Sampling)
## Chain 4:                0.119 seconds (Total)
## Chain 4:
```

```
fit_press
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
```

```
## Formula: rt ~ 1
##    Data: df_spacebar (Number of observations: 361)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   168.63      1.36   166.00   171.33 1.00     3747     2772
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    25.00      0.93    23.23    26.82 1.00     3295     2632
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
# fit the model
# The model specification:
brm(rt ~ 1, data = df_spacebar,
# The likelihood assumed:
family = gaussian(),
# The prior specification:
prior = c(
prior(uniform(0, 60000), class = Intercept),
prior(uniform(0, 2000), class = sigma)
),
# Sampling specifications:
chains = 4,
iter = 2000,
warmup = 1000
)
```

```
## Warning: It appears as if you have specified a lower bounded prior on a parameter that has no natural
## If this is really what you want, please specify argument 'lb' of 'set_prior' appropriately.
## Warning occurred for prior
## Intercept ~ uniform(0, 60000)
```

```
## Warning: It appears as if you have specified an upper bounded prior on a parameter that has no natura
## If this is really what you want, please specify argument 'ub' of 'set_prior' appropriately.
## Warning occurred for prior
## Intercept ~ uniform(0, 60000)
## <lower=0> sigma ~ uniform(0, 2000)
```

```
## Compiling Stan program...
```

```
## Start sampling
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1: Rejecting initial value:
## Chain 1:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 1:   Stan can't start sampling from this initial value.
## Chain 1: Rejecting initial value:
## Chain 1:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 1:   Stan can't start sampling from this initial value.
```

```
## Chain 1: Rejecting initial value:
## Chain 1:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 1:   Stan can't start sampling from this initial value.
## Chain 1:
## Chain 1: Gradient evaluation took 3.7e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.37 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.085 seconds (Warm-up)
## Chain 1:                0.055 seconds (Sampling)
## Chain 1:                0.14 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2: Rejecting initial value:
## Chain 2:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 2:   Stan can't start sampling from this initial value.
## Chain 2: Rejecting initial value:
## Chain 2:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 2:   Stan can't start sampling from this initial value.
## Chain 2: Rejecting initial value:
## Chain 2:   Log probability evaluates to log(0), i.e. negative infinity.
## Chain 2:   Stan can't start sampling from this initial value.
## Chain 2:
## Chain 2: Gradient evaluation took 1.3e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.13 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
```

```
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.073 seconds (Warm-up)
## Chain 2:                0.075 seconds (Sampling)
## Chain 2:                0.148 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.6e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.138 seconds (Warm-up)
## Chain 3:                0.058 seconds (Sampling)
## Chain 3:                0.196 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.6e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.16 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.073 seconds (Warm-up)
## Chain 4:                0.057 seconds (Sampling)
```

```
## Chain 4:                0.13 seconds (Total)
## Chain 4:
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: rt ~ 1
##    Data: df_spacebar (Number of observations: 361)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   168.65      1.33   166.13   171.30 1.00     3710     2644
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma    24.99      0.94    23.24    26.84 1.00     3927     3086
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Day 3

Can we say that a 95% CI (confidence) contains the true value of μ with probability of 95%? i.e., P(lowerCI $< \mu <$ upperCI) = .95

No: from a frequentist perspective, μ is a **point value**. It would have to be a **random variable** to talk about the probability of it lying within a range.

Fisher and Pearson were working infields with very tightly controlled designs/high power.

Sampling agorithms

Day 4

Day 5