Simulation: A Reverse-Engineering Approach to Understand OLS, MLE

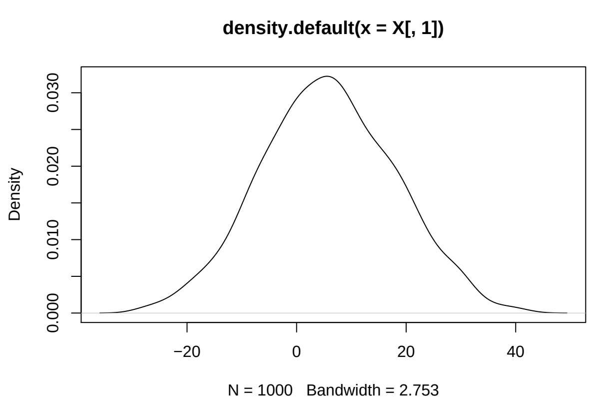
Haohan Chen

February 16, 2018

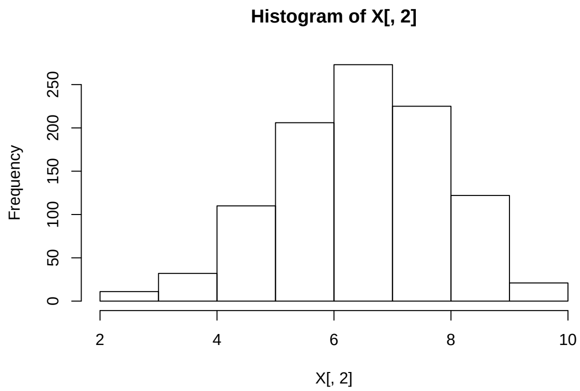
The Big Picture: How Do We Make Predictions?

The setup of a linear model (various notation systems)

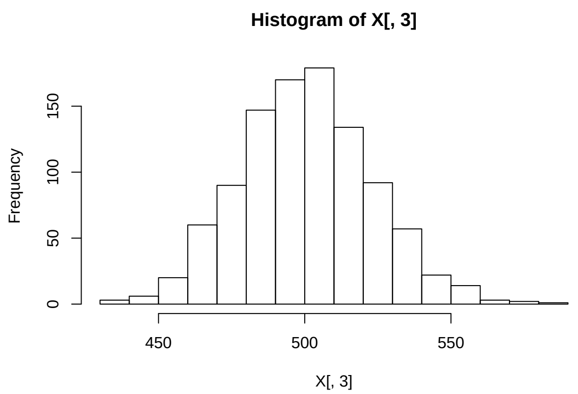
```
# ------------------------------------
# Simulation for reverse-engineering
# ------------------------------------

# Parameters about the simulation
#----------------------------------

# Set N = sample size, k = number of independent variables
  N <- 1000
  k <- 5
# Set random seed so you get the same result every time.
  set.seed(2.16)

# We first look at the Systematic component
#-------------------------------------------

# Generate our independent variables X (in a matrix)
  X <- matrix(NA, nrow = N, ncol = k)
# Note: no requirement about their distribution
  # x1 is a continuous variable drawn from normal dist
  X[, 1] <- rnorm(N, 5, 12)
  plot(density(X[, 1]))
```



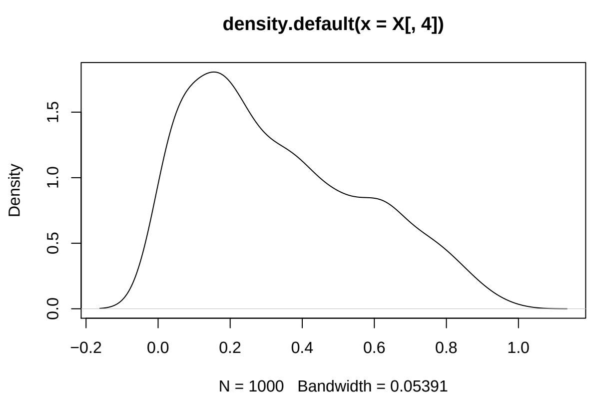

```
  # x2 is a count variable drawn from a binomial dist, N = 10, p = 0.7
  X[, 2] <- rbinom(N, 10, 0.7)
  hist(X[, 2])
```



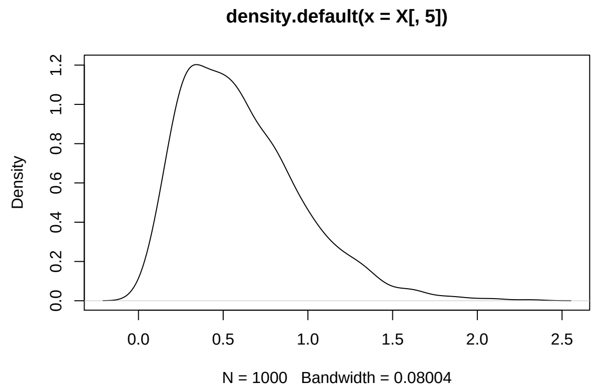

```r
# x3 is another count variable drawn form a poisson dist, mean = 500
X[, 3] <- rpois(N, 500)
hist(X[, 3])
```




```r
# x4 is a proportion variable drawn form a beta distribution
X[, 4] <- rbeta(N, 1, 2)
plot(density(X[, 4]))
```




```r
# x5 is a continuous variable drawn form a gamma distribution
X[, 5] <- rgamma(N, 3, 5)
plot(density(X[, 5]))
```




```r
# These are our independent variables, but we are not done yet. add the intercept
  X <- cbind(rep(1, N), X)

# What does X look like?
  colnames(X) <- c("constant", paste0("x", 1:k))
  head(X)
```

```
##      constant         x1 x2  x3         x4        x5
## [1,]        1 -5.762975  6 481 0.02002443 0.2423048
## [2,]        1  7.218190  6 473 0.42308101 1.0173314
## [3,]        1 24.054144  6 500 0.24040910 0.6037380
## [4,]        1 -8.564508  7 497 0.97297753 0.2308831
## [5,]        1  4.036979  8 461 0.29668598 0.1388283
## [6,]        1  6.589043  8 510 0.41919450 0.2561764
```

```r
# Now, determine the "ground-truth" parameters
  beta <- c(40, 1, 4, 0.2, 9, 20) # Your choice, make sure length of beta = k
  names(beta) <- c("(Intercept)", paste0("x", 1:k))
# Then determine the systematic component
  sys_component <- X %*% beta
  head(sys_component)
```

```
##          [,1]
## [1,] 159.4633
## [2,] 189.9725
## [3,] 202.2926
## [4,] 172.2100
## [5,] 173.6837
## [6,] 189.4853
```

```r
# We are done with the systematic component. No assumptions.
# Now we turn to the stochastic component. I am jumping ahead to make the
# Gauss-Markov + normality assumption (not required
# in the first half of demo about OLS, check lecture slides)

# The Stochastic Component
# --------------------
  epsilon <- rnorm(N, 0, 2) # mean = 0, sd = 3
  stochastic_component <- epsilon
```

Getting OLS estimators for linear models

(Whiteboard demo)

```
# --------------------------------------------------
# Put together a simulated dataset for linear models
# -------------------------------------------------

# Y for linear relation
# -------------------------
  Y <- sys_component + stochastic_component

# Voila: we are done simulating for a linear model. Think about this
# We observe only Y and X, while not observing beta, epsilon. like this
  data_linear <- data.frame(Y = Y, X)
  head(data_linear)
```

```
##          Y constant        x1 x2  x3         x4        x5
## 1 156.2328        1 -5.762975  6 481 0.02002443 0.2423048
## 2 188.9919        1  7.218190  6 473 0.42308101 1.0173314
## 3 201.6978        1 24.054144  6 500 0.24040910 0.6037380
## 4 172.2819        1 -8.564508  7 497 0.97297753 0.2308831
## 5 173.5673        1  4.036979  8 461 0.29668598 0.1388283
## 6 189.1417        1  6.589043  8 510 0.41919450 0.2561764
```

```
# I want to separate out a few samples to demo prediction so I just
# Get a slice out of the complete dataset
# You heard about "training" and "test" set. But let's not discuss it now
  data_new <- data_linear[1:10, ]
  data_fit <- data_linear[-(1:10), ]
  head(data_new)
```

```
##          Y constant        x1 x2  x3         x4        x5
## 1 156.2328        1 -5.762975  6 481 0.02002443 0.2423048
## 2 188.9919        1  7.218190  6 473 0.42308101 1.0173314
## 3 201.6978        1 24.054144  6 500 0.24040910 0.6037380
## 4 172.2819        1 -8.564508  7 497 0.97297753 0.2308831
## 5 173.5673        1  4.036979  8 461 0.29668598 0.1388283
## 6 189.1417        1  6.589043  8 510 0.41919450 0.2561764
```

```
  head(data_fit) # we use this to fit the model
```

```
##           Y constant          x1 x2  x3          x4        x5
## 11 191.2679        1  10.0118090  6 519 0.455841528 0.5060738
## 12 197.2827        1  16.7810333  9 487 0.047185283 0.4537083
## 13 176.7305        1   0.2876557  7 486 0.108345348 0.5818760
## 14 164.7003        1  -7.4760277  4 532 0.007592964 0.5590694
## 15 204.2397        1  26.3867475  8 497 0.412413849 0.1984410
## 16 157.4544        1 -22.7328290  7 502 0.308822018 0.5325092
```

```
# --------------------------------------------
# Get OLS estimators for the simulated data
# --------------------------------------------

# Fit the model
# -----------------
# You've done it a million times with R funciton. I suppose
```

```r
m_ols <- lm(Y ~ x1 + x2 + x3 + x4 + x5, data = data_fit)
coef(m_ols) # The estimated coefficient
```

```
## (Intercept)          x1          x2          x3          x4          x5 
##  41.9894037   1.0094391   4.0234648   0.1953583   8.8127729  20.0088841
```

```r
beta # The "ground truth"
```

```
## (Intercept)          x1          x2          x3          x4          x5 
##        40.0         1.0         4.0         0.2         9.0        20.0
```

```r
# You find from above the estimated beta is almost the "ground truth"
# So you know it's working

# Inference (uncertainty of beta's)
-----------------------------------
# Variance in estimation?
vcov(m_ols)
```

```
##               (Intercept)            x1            x2            x3
## (Intercept) -2.1840170426 -3.216162e-04  1.304090e-02  4.144213e-03
## x1          -0.0003216162 -2.857943e-05 -9.748134e-06  1.124962e-06
## x2           0.0130408975 -9.748134e-06 -2.077069e-03  2.268167e-06
## x3           0.0041442129  1.124962e-06  2.268167e-06 -8.329995e-06
## x4           0.0225956737 -1.208696e-05  6.535147e-04 -6.172582e-06
## x5           0.0106067288 -1.070876e-05  2.058067e-04  1.631596e-05
##                        x4            x5
## (Intercept)  2.259567e-02  1.060673e-02
## x1          -1.208696e-05 -1.070876e-05
## x2           6.535147e-04  2.058067e-04
## x3          -6.172582e-06  1.631596e-05
## x4          -7.455160e-02  1.484871e-03
## x5           1.484871e-03 -3.370285e-02
```

```r
# Simulate beta to get conficence interval of your coefficients
N_sim <- 10000 # note: a different thing from N defined above
library(MASS) # library for the mvrnorm function
beta_sim <- mvrnorm(N_sim, mu = coef(m_ols), Sigma = vcov(m_ols))
head(beta_sim)
```

```
##      (Intercept)       x1       x2        x3       x4       x5
## [1,]    41.45633 1.002515 4.043962 0.1962681 9.275595 19.67583
## [2,]    45.40551 1.011187 3.997409 0.1887883 8.841115 19.95738
## [3,]    43.83916 1.011979 4.060883 0.1909360 8.621648 20.13476
## [4,]    41.46801 1.016323 4.059884 0.1965087 8.832222 19.46800
## [5,]    42.59391 1.011847 4.031356 0.1938568 8.795301 20.06492
## [6,]    40.00121 1.015074 4.047794 0.1987134 8.784852 20.18626
```

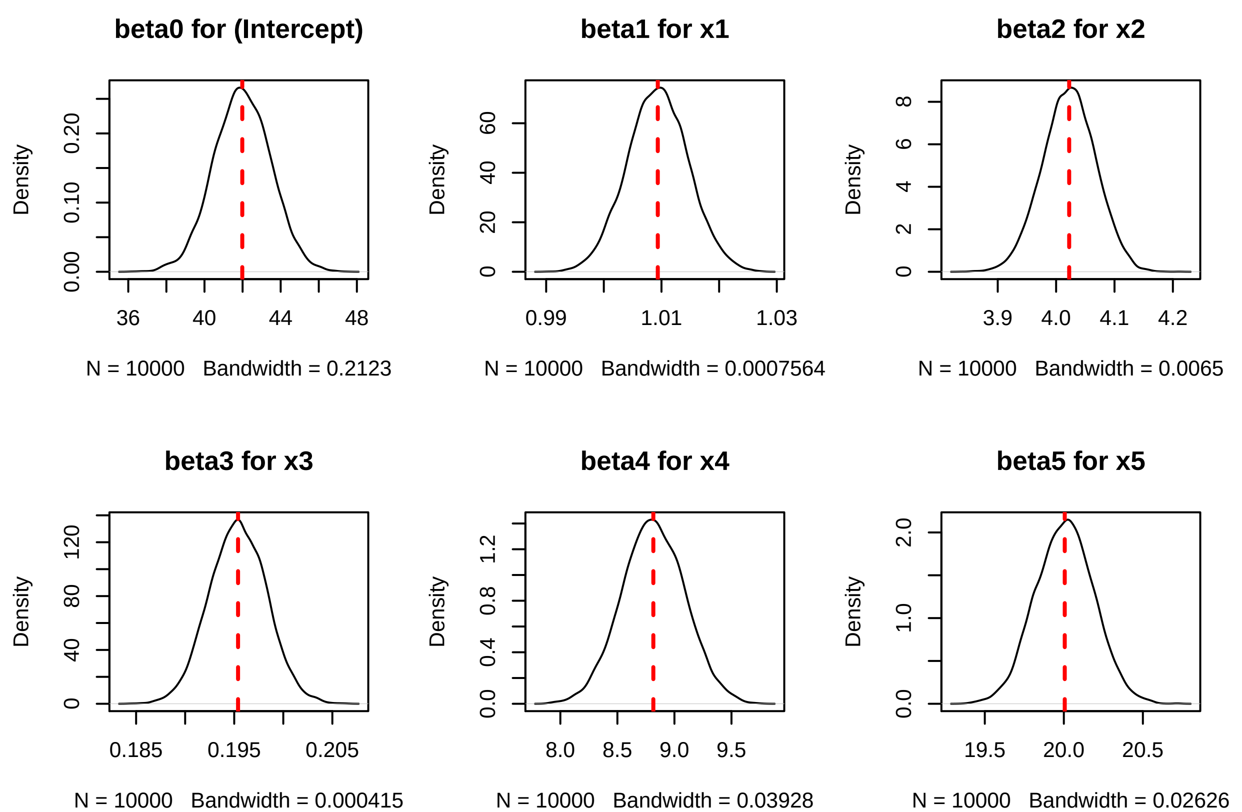
```r
# Get summary statistics of this sample
beta_confint95 <- apply(beta_sim, 2, function(x) quantile(x, c(0.025, 0.5, 0.975)))
beta_confint95
```

```
##       (Intercept)        x1       x2        x3       x4       x5
## 2.5%     39.08532 0.9990038 3.932624 0.1897456 8.278858 19.64569
## 50%      41.97990 1.0093562 4.022835 0.1953956 8.813528 20.00757
## 97.5%    44.89896 1.0199328 4.110426 0.2011247 9.362238 20.36326
```

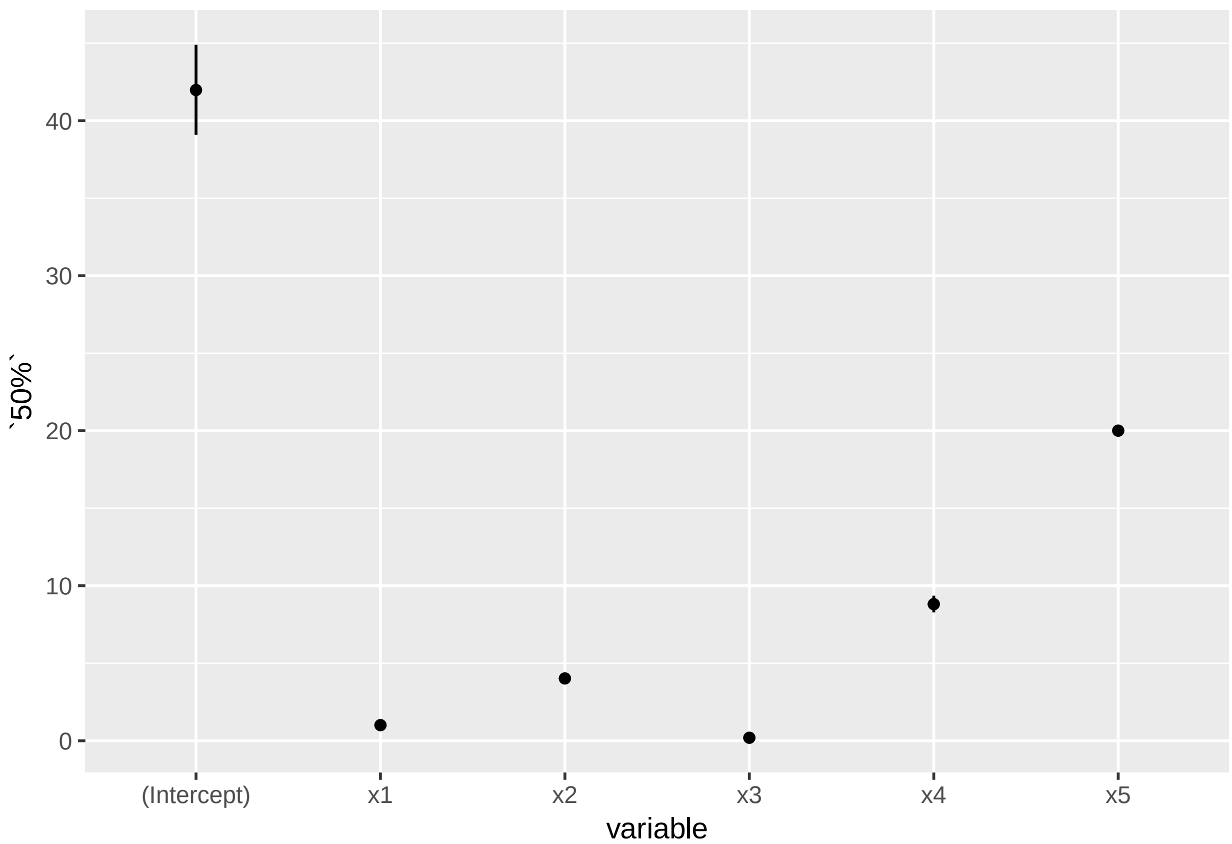
```r
# Plot the distribution of beta's
  par(mfrow = c(2, ceiling((k+1)/2)))
  for (i in 1:ncol(beta_sim)){
    plot(density(beta_sim[, i]),
         main = paste(paste0("beta", i-1), "for", colnames(beta_sim)[i]))
    abline(v = mean(beta_sim[, i]), col = "red", lty = 2, lwd = 2)
  }
# Plot coefficients (my way... feel free to do it in another way)
  beta_confint95_2 <- as.data.frame(t(beta_confint95))
  beta_confint95_2$variable <- row.names(beta_confint95_2)
  names(beta_confint95_2)
```

```
## [1] "2.5%"      "50%"       "97.5%"     "variable"
```

```r
  library(ggplot2)
```




```r
  ggplot(beta_confint95_2, aes(y = `50%`, x = variable)) + geom_point() +
    geom_linerange(aes(ymin = `2.5%`, ymax = `97.5%`))
```




```r
  # Some are very small, because not at the same scale.

# Prediction (uncertainty about a new y)
# ------------------------------------------
# Task: you have a new x (vector), you want to know what the corresponding y is
  head(data_new)
```

```
##          Y constant        x1 x2  x3         x4        x5
## 1 156.2328        1 -5.762975  6 481 0.02002443 0.2423048
## 2 188.9919        1  7.218190  6 473 0.42308101 1.0173314
## 3 201.6978        1 24.054144  6 500 0.24040910 0.6037380
## 4 172.2819        1 -8.564508  7 497 0.97297753 0.2308831
## 5 173.5673        1  4.036979  8 461 0.29668598 0.1388283
## 6 189.1417        1  6.589043  8 510 0.41919450 0.2561764
```

```r
# Let's take one of them
  x_new <- data_new[1, c("constant", paste0("x", 1:k))]
  x_new
```

```
##   constant        x1 x2  x3         x4        x5
## 1        1 -5.762975  6 481 0.02002443 0.2423048
```

```r
# Challenge: What one of the x to be certain value?

# Want: y_new ~ N(mu_y_new, sigma_y).
# When we predict, we get a distribution, not one number

  # First we get mu_y_new: harder part
```

```r
mu_y_new <- as.matrix(x_new, byrow = T) %*% t(beta_sim)
dim(mu_y_new) # It's a vector of 10000 sample of predicted y
```

```
## [1]     1 10000
```

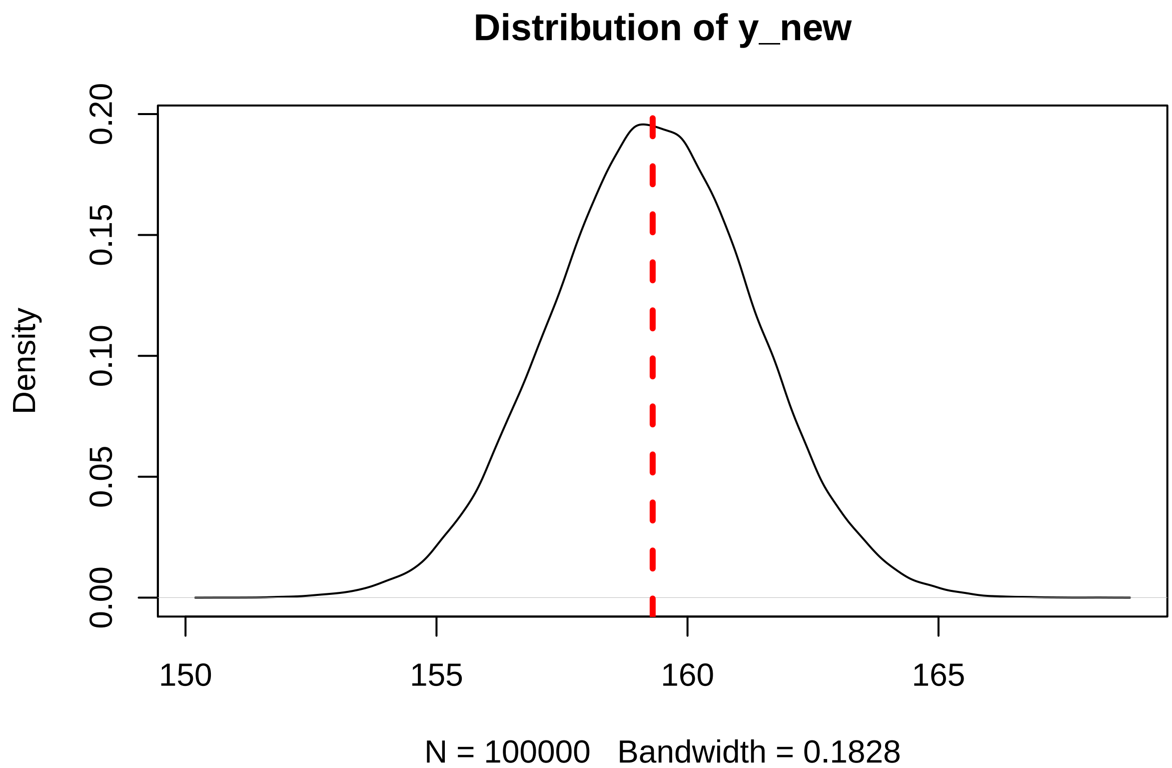
```r
mu_y_new <- as.vector(mu_y_new) # just to clean up. transform it into a vector
mu_y_new[1:30] # See how it looks like
```

```
##  [1] 159.3009 159.3825 159.2640 159.3850 159.2339 159.0864 159.6307
##  [8] 159.5388 159.2754 159.6779 159.3888 159.0747 159.2673 159.2374
## [15] 159.4132 159.0057 159.1771 159.1440 159.3516 159.4732 159.1585
## [22] 159.3798 159.2244 159.5618 159.1205 159.1474 159.0909 159.5011
## [29] 159.2602 158.9382
```

```r
  # Then we get sigma_y. easy
  sigma_y <- sigma(m_ols)
  # Want it by hand (using week 2 page 11 formula)
  sigma_y <- sqrt((t(residuals(m_ols)) %*% residuals(m_ols)) /
    (N - (k + 1)))

# Then, simulate y_new useing this list of mu_y_new
  y_new <- rnorm(100000, mu_y_new, sigma_y)

# We are done with prediction! You have predicted y_new. Plot it, summarise it
  quantile(y_new, c(0.025, 0.5, 0.975)) # 95% confidence interval
```

```
##     2.5%      50%    97.5%
## 155.3163 159.3115 163.2921
```

```r
par(mfrow = c(1, 1))
plot(density(y_new), main = "Distribution of y_new")
abline(v = mean(y_new), col = "red", lty = 2, lwd = 3)
```




```r
# Expected new y? Simply take the mean!
  mean(y_new) # this is your expected value
```

```
## [1] 159.3062
```

```r
# Now, challenge: do it without using lm().
  est_beta_manual <- solve(t(X) %*% X) %*% t(X) %*% Y
  est_beta_manual
```

```
##                [,1]
## constant 41.8937720
## x1        1.0093564
## x2        4.0231604
## x3        0.1955287
## x4        8.8384388
## x5       20.0106527
```

```r
  e <- Y - X %*% est_beta_manual
  # a bit different because we remove 10 cases for the prediction task
  est_beta_vcov <- as.numeric((t(e) %*% e) / (N - (k + 1))) * solve(t(X) %*% X)
  est_beta_vcov
```

```
##               constant            x1            x2            x3
## constant  2.1561203113  3.240649e-04 -1.286443e-02 -4.092776e-03
## x1        0.0003240649  2.815450e-05  9.242122e-06 -1.122038e-06
## x2       -0.0128644328  9.242122e-06  2.052472e-03 -2.295528e-06
## x3       -0.0040927759 -1.122038e-06 -2.295528e-06  8.229755e-06
## x4       -0.0217919051  1.907055e-05 -6.444249e-04  5.063731e-06
```

```
## x5       -0.0102673796  9.309850e-06 -1.908965e-04 -1.644516e-05
##                     x4            x5
## constant -2.179191e-02 -1.026738e-02
## x1        1.907055e-05  9.309850e-06
## x2       -6.444249e-04 -1.908965e-04
## x3        5.063731e-06 -1.644516e-05
## x4        7.320729e-02 -1.365543e-03
## x5       -1.365543e-03  3.316061e-02
```

MLE estimator for linear model

(Whiteboard demo)

```r
# Fit the model
  m_mle <- glm(Y ~ x1 + x2 + x3 + x4 + x5, data = data_fit, family = "gaussian")
# Get estimated beta
  coef(m_mle)
```

```
## (Intercept)          x1          x2          x3          x4          x5
##  41.9894037   1.0094391   4.0234648   0.1953583   8.8127729  20.0088841
```

```r
  beta
```

```
## (Intercept)          x1          x2          x3          x4          x5
##        40.0         1.0         4.0         0.2         9.0        20.0
```

```r
# Get variance of the estimation
  vcov(m_mle)
```

```
##               (Intercept)            x1            x2            x3
## (Intercept)  2.1840170426  3.216162e-04 -1.304090e-02 -4.144213e-03
## x1           0.0003216162  2.857943e-05  9.748134e-06 -1.124962e-06
## x2          -0.0130408975  9.748134e-06  2.077069e-03 -2.268167e-06
## x3          -0.0041442129 -1.124962e-06 -2.268167e-06  8.329995e-06
## x4          -0.0225956737  1.208696e-05 -6.535147e-04  6.172582e-06
## x5          -0.0106067288  1.070876e-05 -2.058067e-04 -1.631596e-05
##                        x4            x5
## (Intercept) -2.259567e-02 -1.060673e-02
## x1           1.208696e-05  1.070876e-05
## x2          -6.535147e-04 -2.058067e-04
## x3           6.172582e-06 -1.631596e-05
## x4           7.455160e-02 -1.484871e-03
## x5          -1.484871e-03  3.370285e-02
```

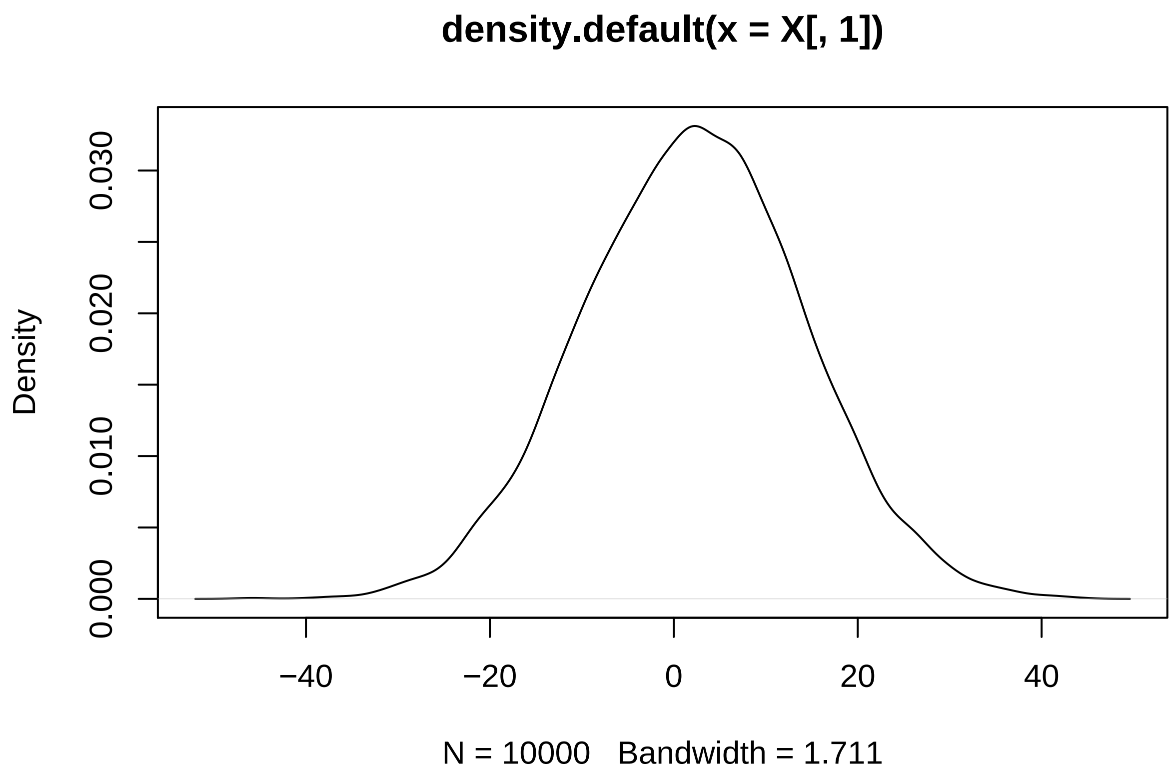
```r
# Inference
# ------------
  # (exercise)

# Predictions
# ------------
  # (exercise)
```

Modeling Dicotomous Outcome Using Generalized Linear Model: Setup

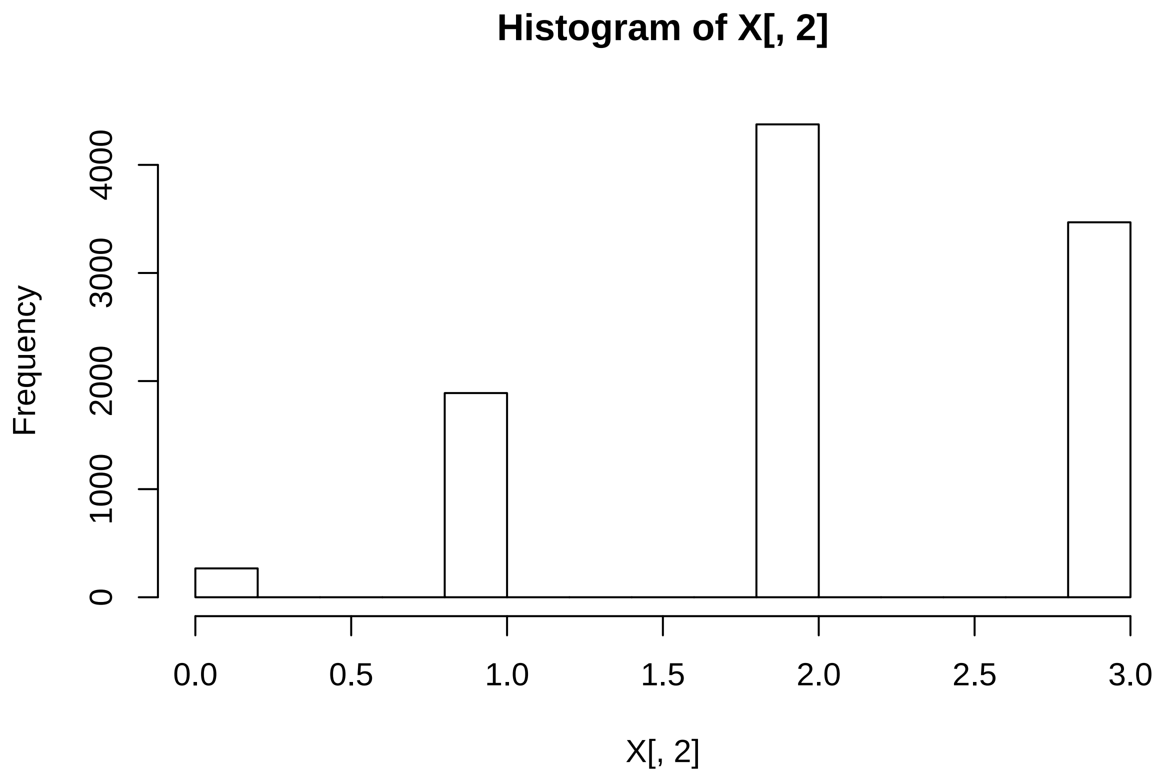
(Whiteboard demonstration)

```r
rm(list=ls())
```

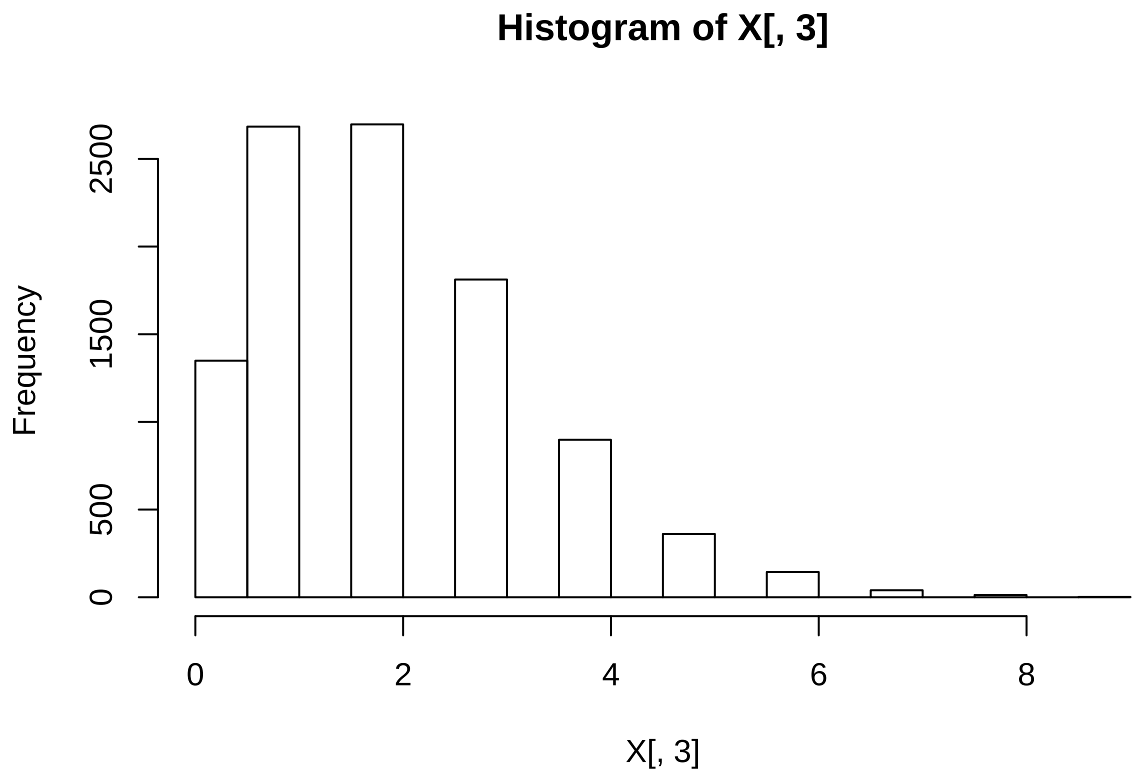
```r
# Just copy pasting. should've made it a function. pressed for time.

# Parameters about the simulation (nothing to do with the model)
#-----------------------------------------------------------------

# Set N = sample size, k = number of independent variables
  N <- 10000
  k <- 5
# Set random seed so you get the same result every time.
  set.seed(2.16)

# We first look at the Systematic component
#-------------------------------------------

# Generate our independent variables X (in a matrix)
  X <- matrix(NA, nrow = N, ncol = k)
# Note: no requirement about their distribution
  # x1 is a continuous variable drawn from normal dist
  X[, 1] <- rnorm(N, 2, 12)
  plot(density(X[, 1]))
```



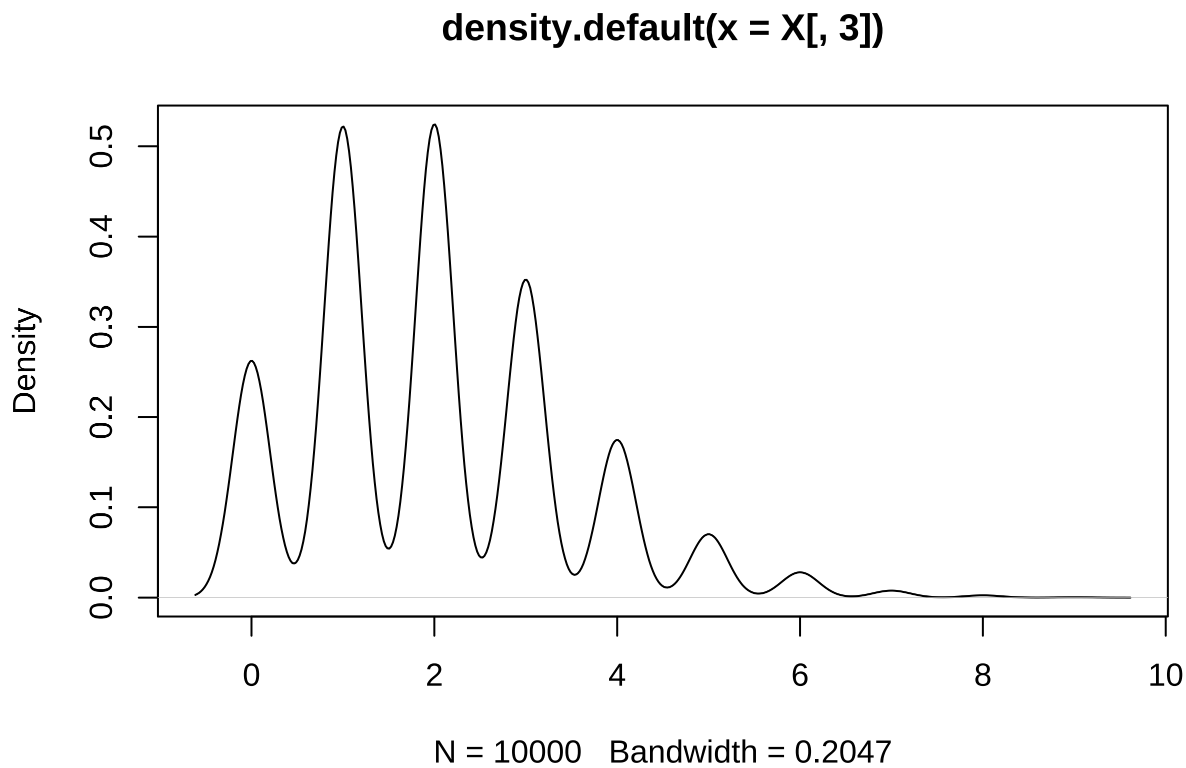

```r
  # x2 is a count variable drawn from a binomial dist, N = 10, p = 0.7
  X[, 2] <- rbinom(N, 3, 0.7)
  hist(X[, 2])
```



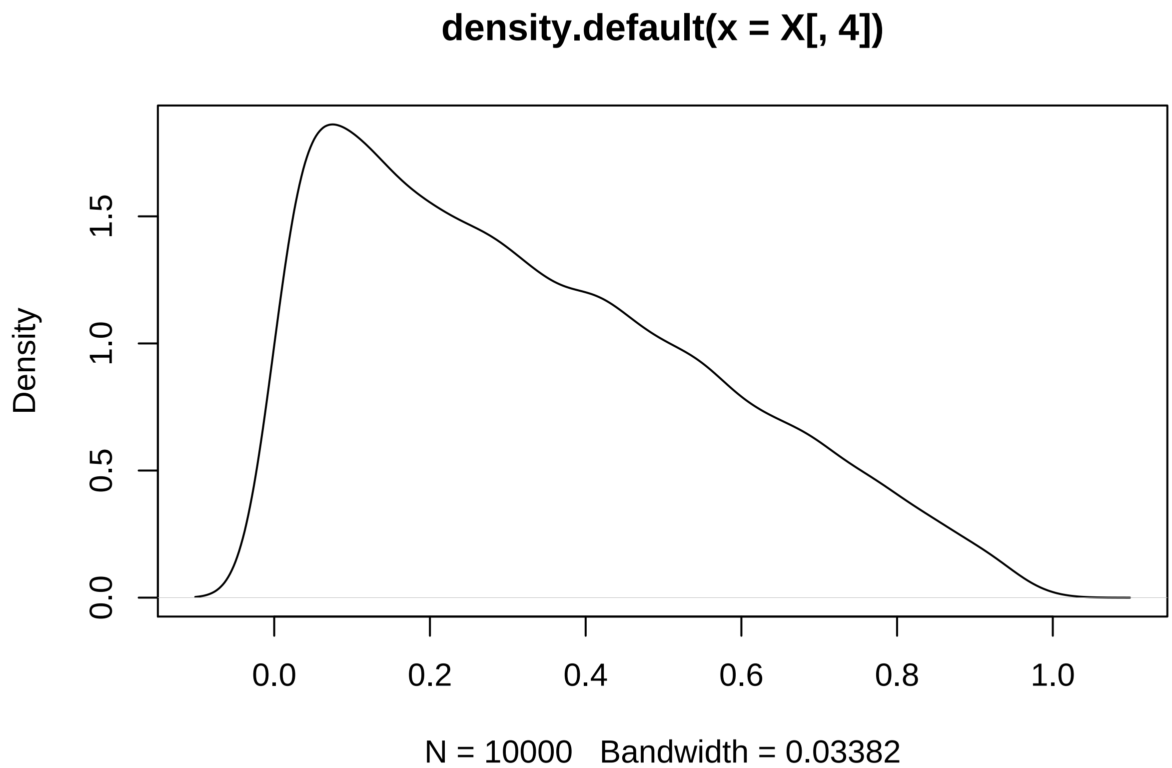

```r
# x3 is another count variable drawn form a poisson dist, mean = 500
X[, 3] <- rpois(N, 2)
hist(X[, 3])
```




```r
# x4 is a proportion variable drawn form a beta distribution
X[, 4] <- rbeta(N, 1, 2)
plot(density(X[, 3]))
```




```r
# x5 is a continuous variable drawn form a gamma distribution
X[, 5] <- rgamma(N, 3, 5)
plot(density(X[, 4]))
```




```r
# These are our independent variables, but we are not done yet. add the intercept
  X <- cbind(rep(1, N), X)

# What does X look like?
  colnames(X) <- c("constant", paste0("x", 1:k))
  head(X)
```

```
##      constant         x1 x2 x3         x4        x5
## [1,]        1  -8.762975  1  1 0.14158054 0.2043488
## [2,]        1   4.218190  1  2 0.35102508 0.8078260
## [3,]        1  21.054144  3  2 0.02282631 0.6614940
## [4,]        1 -11.564508  2  3 0.25226094 1.5345986
## [5,]        1   1.036979  1  2 0.18886517 0.4645816
## [6,]        1   3.589043  1  0 0.35007593 1.1798757
```

```r
# Now, determine the "ground-truth" parameters
  beta <- c(0.02, 0.01, 0.04, 0.02, 0.09, 0.05) # Your choice, make sure length of beta = k
  names(beta) <- c("(Intercept)", paste0("x", 1:k))
# Then determine the systematic component
  sys_component <- X %*% beta
  head(sys_component)
```

```
##            [,1]
## [1,] 0.01532995
## [2,] 0.21416546
## [3,] 0.42567051
## [4,] 0.14378834
```

```
## [5,] 0.15059674
## [6,] 0.18639105
```

```
# We are done with the systematic component. No assumptions.
# Now we turn to the stochastic component. I am jumping a head to make the
# Gauss-Markov + normality assumption (not required
# in the first half of demo about OLS, check lecture slides)

# The Stochastic Component
# --------------------
  epsilon <- rnorm(N, 0, 0.001) # mean = 0, sd = 3
  stochastic_component <- epsilon
```

```
# -----------------------------------
# Set up the simulated dataset
# -----------------------------------

# The probabilities with a logistic link
  pi_logit <- 1 / (1 + exp(-sys_component))
# The probabilities with a probit link
  pi_probit <- pnorm(sys_component)

set.seed(1)
# The outcome y of a logit link
  y_logit <- sapply(pi_logit, function(x) rbinom(1, 1, x))
  table(y_logit)
```

```
## y_logit
##    0    1
## 4377 5623
```

```
# The outcome y of a probit link
  y_probit <- sapply(pi_probit, function(x) rbinom(1, 1, x))
  table(y_probit)
```

```
## y_probit
##    0    1
## 4149 5851
```

```
# Dataset
  data_logit <- data.frame(Y = y_logit, X)
  data_probit <- data.frame(Y = y_probit, X)
```

MLE Estimator (Logit)

```
# -----------------------------------
# Fit the model
# -----------------------------------
m_logit <- glm(Y ~ x1 + x2 + x3 + x4 + x5, data = data_logit, family = binomial(link = "logit"))
coef(m_logit)
```

```
## (Intercept)          x1          x2          x3          x4          x5
## -0.02225989  0.01058423  0.07264404  0.01315730  0.10814500  0.05971384
```

```
beta
```

```
## (Intercept)          x1          x2          x3          x4          x5 
##        0.02        0.01        0.04        0.02        0.09        0.05 
```

```r
# -----------------------------------
# Inference (exercise)
# -----------------------------------

# -----------------------------------
# Prediction (exercise)
# -----------------------------------
```

MLE Estimator (Probit)

```r
# -----------------------------------
# Fit the model
# -----------------------------------
m_probit <- glm(Y ~ x1 + x2 + x3 + x4 + x5, data = data_logit, family = binomial(link = "probit"))
coef(m_probit)
```

```
##  (Intercept)           x1           x2           x3           x4 
## -0.013643820  0.006600958  0.045366949  0.008194967  0.067244283 
##           x5 
##  0.037436864 
```

```r
beta
```

```
## (Intercept)          x1          x2          x3          x4          x5 
##        0.02        0.01        0.04        0.02        0.09        0.05 
```

```r
vcov(m_probit)
```

```
##               (Intercept)            x1            x2            x3
## (Intercept)  2.358841e-03 -1.287707e-06 -5.231372e-04 -1.498670e-04
## x1          -1.287707e-06  1.110506e-06 -1.275007e-07  4.897661e-09
## x2          -5.231372e-04 -1.275007e-07  2.514135e-04 -1.534684e-06
## x3          -1.498670e-04  4.897661e-09 -1.534684e-06  7.730016e-05
## x4          -9.311772e-04 -5.034428e-07 -8.383379e-06 -9.198598e-06
## x5          -8.067830e-04 -8.863650e-07  1.577707e-06  5.804393e-07
##                        x4            x5
## (Intercept) -9.311772e-04 -8.067830e-04
## x1          -5.034428e-07 -8.863650e-07
## x2          -8.383379e-06  1.577707e-06
## x3          -9.198598e-06  5.804393e-07
## x4           2.834946e-03  3.863221e-05
## x5           3.863221e-05  1.315015e-03
```

```r
# -----------------------------------
# Inference (exercise)
# -----------------------------------

# -----------------------------------
# Prediction (exercise)
# -----------------------------------
```