Econ613: Homework 2

Peng Peng

2/1/2019

Exercise 1 Data Creation

```r
set.seed(1234)

# Create Xs and eps-----------------------------------------------

X1 = runif(10000, min = 1, max = 3)
X2 = rgamma(10000, shape = 3, scale = 2)
X3 = rbinom(10000, size = 1, prob = 0.3)
eps = rnorm(10000, mean = 2, sd = 1)
intercept = c(rep(1, 10000))
d1 = data.frame(cbind(intercept, X1, X2, X3, eps))


# Create Y and ydum----------------------------------------------

d1 %<>%
  mutate(Y = intercept + 1.2*X1 - 0.9*X2 + 0.1*X3 + eps) %<>%
  mutate(ydum = ifelse(Y > mean(Y), 1, 0))
```

Exercise 2 OLS

```r
# Calculate corr(Y, X1)--------------------------------------

x1 = d1$X1
y = d1$Y
n = nrow(d1)

# Calculate sample standard deviation of X1 and Y
sd_x = sqrt((sum((x1 - mean(x1))^2))/(n - 1))
sd_y = sqrt((sum((y - mean(y))^2))/(n - 1))

# Calculate correlation
corr = cov(x1, y)/(sd_x*sd_y)

# How different is it from 1.2
diff = 1.2 - corr

# Calculate the coefficients on the regression--------------------

X = cbind(intercept, X1, X2, X3, eps)
Y = as.matrix(d1$Y)
A = solve(t(X) %*% X) %>% as.matrix()
```

```r
b_hat = A %*% t(X) %*% Y
b_hat
```

```
##            [,1]
## intercept  1.0
## X1         1.2
## X2        -0.9
## X3         0.1
## eps        1.0
```

```r
# Calculate the standard errors-----------------------------------
n = nrow(d1)
k = ncol(X) - 1
u_hat = Y - X %*% b_hat
sigsq_hat = as.numeric(t(u_hat) %*% u_hat / (n - k - 1))
b_cov = sigsq_hat * A
se_ols = sqrt(diag(b_cov))
se_ols
```

```
##    intercept           X1           X2           X3          eps
## 7.819496e-15 2.985431e-15 4.997326e-16 3.717346e-15 1.708077e-15
```

```r
# Bootstrap standard errors------------------------------------

# write a function to return standard error
boot_se = function(data){
  X = cbind(data$intercept, data$X1, data$X2, data$X3, data$eps)
  Y = as.matrix(data$Y)
  A = as.matrix(solve(t(X) %*% X))
  b_hat = A %*% t(X) %*% Y
  n = nrow(data)
  k = ncol(data) - 1
  u_hat = Y - X %*% b_hat
  sigsq_hat = as.numeric(t(u_hat) %*% u_hat / (n - k - 1))
  b_cov = sigsq_hat * A
  b_se = sqrt(diag(b_cov))
  return(b_se)
}

# write a for loop to resample 49 times
output = list()

for (x in 1:49) {
  df = sample_n(d1, nrow(d1), replace = T)
  output[[x]] = boot_se(df)
}

bt_se_ols = data.frame(t(sapply(output, c)))

# rename the columns and match to d1
names(bt_se_ols) = names(d1[, 1:5])

#c alculate the average standard error
bt_se_ols = bt_se_ols %>% summarise_all(mean)
bt_se_ols
```

```
##      intercept           X1           X2           X3          eps
## 1 5.261845e-15 2.011452e-15 3.363499e-16 2.500659e-15 1.147461e-15
```

```
# write a for loop to resample 499 times
output = list()

for (x in 1:499) {
  df = sample_n(d1, nrow(d1), replace = T)
  output[[x]] = boot_se(df)
}

bt_se_ols2 = data.frame(t(sapply(output, c)))

#rename the columns and match to d1
names(bt_se_ols2) = names(d1[, 1:5])

#calculate the average standard error
bt_se_ols2 = bt_se_ols2 %>% summarise_all(mean)
bt_se_ols2
```

```
##      intercept           X1           X2           X3          eps
## 1 4.913386e-15 1.874876e-15 3.139929e-16 2.334359e-15 1.072858e-15
```

Exercise 3 Numerical Optimization

```
x = d1[, 1:4] %>% as.matrix()
y = d1$ydum


L_probit = function(beta, X = x, Y = y){
  # get linear predictor
  z = X %*% beta
  # probability
  p = pnorm(z)
  p[p==1] = 0.9999
  p[p==0] = 0.0001
  # log-likelihood function
  L = Y * log(p) + (1 - Y) * log( 1 - p)
  # sum over sample n and take negative log-likelihood
  -sum(L)
}

# Steepest ascent optimization algorithm--------------------------------------
# gradient
alpha = 1e-6
h = 0.0001
output = list()
beta = c(0, 0, 0, 0)
c = 1

gradient =  function(fun,par){
    g = NULL
    for(i in 1:length(par)){
```

```r
   par.n = par
   par.n[i] = par.n[i] + h
   df = (fun(par.n) - fun(par))/h
   g = c(g, df)
    }
    return(g)
}


# compute beta
while (c > 0.0001){
  g = gradient(L_probit,beta)
  beta. = as.numeric(beta - alpha %*% g)
  c = abs((L_probit(beta.)-L_probit(beta)))/abs(L_probit(beta))
  beta = as.numeric(beta.)
}

beta
```

```
## [1]  0.7949055  1.2805594 -0.5777754  0.1856262
```

Exercise 4 Discrete Choice

```r
# Probit model----------------------------------------------------------------

# optimize the negative loglikelihood

L_probit = function(beta, X = x, Y = y){
  # get linear predictor
  z = X %*% beta
  # probability
  p = pnorm(z)
  p[p==1] = 0.9999
  p[p==0] = 0.0001
  # log-likelihood function
  L = Y * log(p) + (1 - Y) * log( 1 - p)
  # sum over sample n and take negative log-likelihood
  -sum(L)
}

fit1 = optim(par = c(0, 0, 0, 0), L_probit)
par1 = fit1$par

# Logit model------------------------------------------------------------
L_logit = function(beta, X = x, Y = y){
  z = X %*% beta
  p = exp(z)/(1 + exp(z))
  p[p==1] = 0.9999
  p[p==0] = 0.0001
  # log-likelihood function
  L = Y * log(p) + (1 - Y) * log( 1 - p)
  # sum over sample n and take negative log-likelihood
  -sum(L)
}

fit2 = optim(par = c(0, 0, 0, 0), L_logit)
par2 = fit2$par


# Linear model-----------------------------------------------------------

L_linear = function(beta, X = x, Y = y){
  z = X %*% beta
  z[z==1] = 0.9999
  z[z==0] = 0.0001
  L = Y * log(z) + (1 - Y) * log(1 - z)
  -sum(L)
}

fit3 = optim(par = c(0, 0, 0, 0), L_linear)
par3 = fit3$par

# Check with pkg output------------------------------------------------

f_probit = glm(ydum ~ X1 + X2 + X3,
               family = binomial(link = probit),
               data = d1)
f_logit = glm(ydum ~ X1 + X2 + X3,
              family = binomial(link = logit),
              data = d1)
f_linear = glm(ydum ~ X1 + X2 + X3,
               family = "gaussian",
               data = d1)

model.list = list(f_probit, f_logit, f_linear)
coef_pkg = sapply(model.list, coef) %>% as.data.frame()
names(coef_pkg)= c("probit", "logit", "linear")
coef_pkg
```

```
##                 probit     logit      linear
## (Intercept)  2.9654092  5.3212472  0.89111066
## X1           1.2343270  2.2190576  0.14521037
## X2          -0.9199287 -1.6529784 -0.10485191
## X3           0.1354812  0.2439212  0.01078067
```

```
coef_hand = cbind(par1, par2, par3) %>% as.data.frame()
names(coef_hand) = c("probit", "logit", "linear")
row.names(coef_hand) = c("(Intercept)", "X1", "X2", "X3")
coef_hand
```

```
##                 probit     logit      linear
## (Intercept)  2.9647599  5.3203555  0.08145489
## X1           1.2340560  2.2187041  0.28585030
## X2          -0.9196697 -1.6527338 -0.01597924
## X3           0.1356510  0.2437645  0.05539490
```

```
# Interpret the coefficients----------------------------------------------
# For probit and logit models, we expect X1, X2, and X3 to have
# positive relationships with ydum. We can not say much about the
# magnitude, however. For linear models, the coefficients are the
# expected changes in the probability for a one-unit change in
# X1, X2, X3 respectively, holding everything elese constant.
```

Exercise 5 Marginal Effects

```
# Compute the marginal effects of X on Y--------------------------

# probit model
x = d1[, 1:4] %>% as.matrix()

# marginal effect for each observation
d_probit = (dnorm(x %*% par1)) %*% par1

# average out marginal effect
marginal_probit = apply(d_probit, 2, mean)
marginal_probit
```

```
## [1]  0.3558299  0.1481112 -0.1103786  0.0162808
```

```
# logit model
d_logit = (exp(x %*% par2) / ((1 + exp(x %*% par2))^2)) %*% par2
marginal_logit = apply(d_logit, 2, mean)
marginal_logit
```

```
## [1]  0.3547761  0.1479494 -0.1102089  0.0162549
```

```
# check with package

probitmfx(formula = f_logit, data = d1, atmean = FALSE)
```

```
## Call:
## probitmfx(formula = f_logit, data = d1, atmean = FALSE)
##
## Marginal Effects:
##           dF/dx   Std. Err.         z     P>|z|
## X1   0.14811320  0.00444500   33.3213 < 2.2e-16 ***
## X2  -0.11038693  0.00041883 -263.5595 < 2.2e-16 ***
## X3   0.01617924  0.00557116    2.9041  0.003683 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## dF/dx is for discrete change for the following variables:
##
## [1] "X3"
```

```
# Compute the standard deviations using the Delta method---------------

# probit model

# marginal effect
```

```r
marginal_probit = function(x = x, beta = par1){
  marginal_probit = (dnorm(x %*% beta)) %*% beta
  return(marginal_probit)
  }

# jacobian matrix
j_probit = jacobian(marginal_probit, par1)

# var(ME)
var_probit = vcov(f_probit)

# standard error
se_probit_delta = diag((j_probit) %*% var_probit %*% t(j_probit))
se_probit_delta
```

```
## [1] 8.580962e-54 1.486711e-54 8.256969e-55 1.796399e-56
```

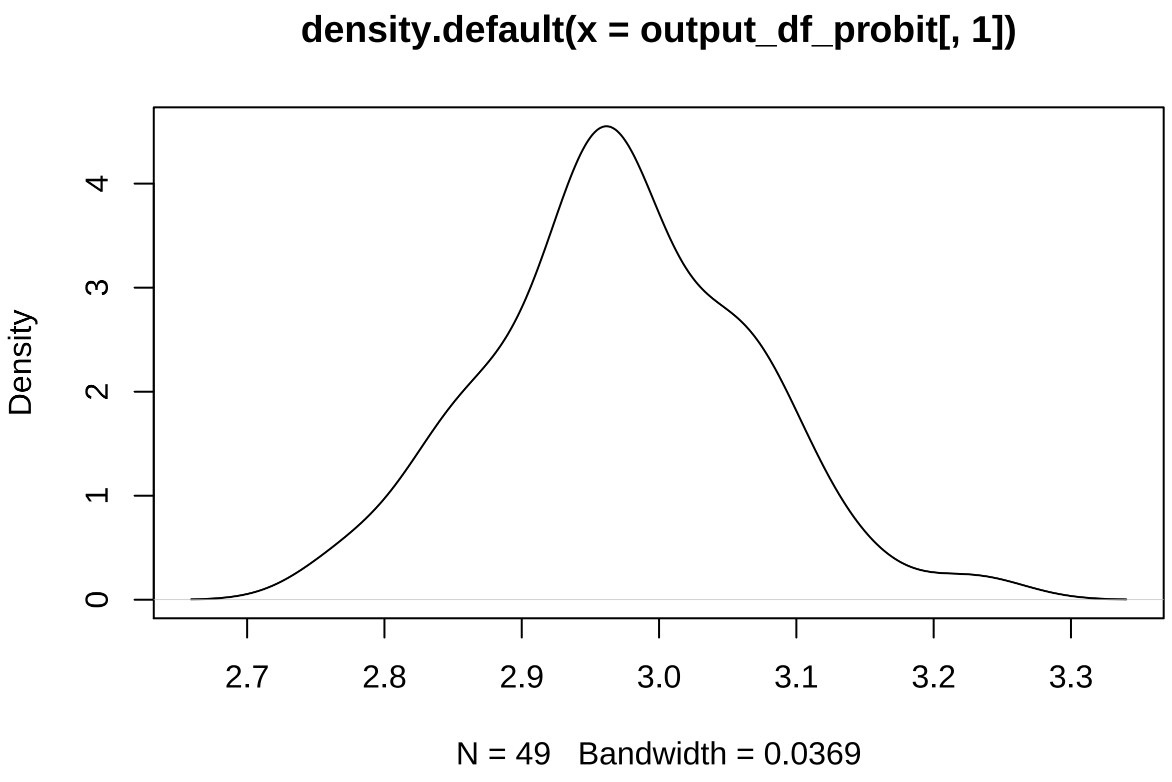
```r
# logit model

#marginal effect
marginal_logit = function(x = x, beta = par2){
  marginal_logit = (dlogis(x %*% beta)) %*% beta
  return(marginal_logit)
  }

# jacobian matrix
j_logit = jacobian(marginal_logit, par2)

# var(ME)
var_logit = vcov(f_logit)

#standard error
se_logit_delta = diag(j_logit %*% var_logit %*% t(j_logit))
se_logit_delta
```

```
## [1] 1.522846e-30 2.648339e-31 1.469537e-31 3.196802e-33
```

```r
# Bootstrap the standard errors------------------------------------------

# probit model

L_probit = function(beta, X, Y){
  z = X %*% beta
  p = pnorm(z)
  p[p==1] = 0.9999
  p[p==0] = 0.0001
  L = Y * log(p) + (1 - Y) * log( 1 - p)
  -sum(L)
}

coef_est_probit = function(data){
  X_s = as.matrix(data[, 1:4])
  Y_s = data[, 7]
  fit1 = optim(par = c(0, 0, 0, 0), fn = L_probit, X = X_s, Y = Y_s)
  par1 = fit1$par
}

output = list()
n_sample = 49

for (i in 1:n_sample){
  df = sample_n(d1, nrow(d1), replace = T)
  output[[i]] = coef_est_probit(data = df)
  message(i, " of ", n_sample)
}

output_df_probit = do.call(rbind, output)
output_df_probit
```

```
##           [,1]     [,2]       [,3]       [,4]
##  [1,] 3.124663 1.236725 -0.9442136 0.09324005
##  [2,] 2.950373 1.258391 -0.9321346 0.14329728
##  [3,] 2.934380 1.278638 -0.9199625 0.05532119
##  [4,] 2.898371 1.255187 -0.9134697 0.08475615
##  [5,] 2.845560 1.165375 -0.8805488 0.16267021
##  [6,] 2.999382 1.281233 -0.9457855 0.15172724
##  [7,] 2.842162 1.267611 -0.9160042 0.18972664
##  [8,] 3.091059 1.207458 -0.9290001 0.19533358
##  [9,] 2.950094 1.124530 -0.8731889 0.16356925
## [10,] 3.229576 1.202872 -0.9547018 0.10772289
## [11,] 2.957918 1.268739 -0.9235366 0.04951397
## [12,] 2.868926 1.207540 -0.8996735 0.21247536
## [13,] 2.895121 1.200357 -0.9001946 0.19629457
## [14,] 2.905841 1.216460 -0.9018122 0.14315333
## [15,] 2.922851 1.246808 -0.9183272 0.12118396
## [16,] 2.956202 1.234779 -0.9188392 0.09054457
## [17,] 2.999532 1.317887 -0.9498415 0.09726819
## [18,] 3.055578 1.255565 -0.9354604 0.10734394
## [19,] 2.944813 1.243720 -0.9128840 0.04124518
## [20,] 3.056045 1.181700 -0.9156933 0.13143176
## [21,] 2.961778 1.273593 -0.9263415 0.09662544
## [22,] 3.080985 1.172927 -0.9163464 0.08130173
## [23,] 2.876548 1.220149 -0.8970858 0.18596699
## [24,] 3.014495 1.194349 -0.9149425 0.14842107
## [25,] 2.964717 1.235787 -0.9178806 0.21541402
## [26,] 3.046958 1.216232 -0.9274126 0.16532734
## [27,] 3.088301 1.204966 -0.9217246 0.04343201
## [28,] 2.769964 1.210938 -0.8790412 0.15162529
## [29,] 2.858360 1.174213 -0.8789133 0.11734108
## [30,] 2.976803 1.281623 -0.9422740 0.15194533
## [31,] 3.074067 1.303358 -0.9647416 0.16673393
## [32,] 3.073290 1.343749 -0.9833639 0.15863682
## [33,] 2.962333 1.275953 -0.9326856 0.12309555
## [34,] 2.974519 1.162013 -0.9013122 0.10939579
## [35,] 3.055452 1.224502 -0.9384737 0.15434495
## [36,] 2.955905 1.216755 -0.9186739 0.15249370
## [37,] 2.923243 1.165052 -0.8839917 0.17608921
## [38,] 2.846809 1.231793 -0.9022306 0.15882503
## [39,] 2.968112 1.212079 -0.9263143 0.19807564
## [40,] 3.143129 1.287800 -0.9658663 0.17956446
## [41,] 2.958893 1.287766 -0.9402827 0.10568309
## [42,] 3.039453 1.321044 -0.9639728 0.15174986
## [43,] 2.832647 1.236524 -0.9080119 0.20206472
## [44,] 3.009536 1.255364 -0.9311835 0.10675503
## [45,] 2.981074 1.220685 -0.9158814 0.12598406
## [46,] 2.783469 1.299805 -0.9088955 0.08517572
## [47,] 3.042518 1.235287 -0.9317072 0.05387081
## [48,] 2.956118 1.146434 -0.8919099 0.10723899
## [49,] 2.974929 1.212616 -0.9094285 0.12742064
```

```
plot(density(output_df_probit[, 1]))
```

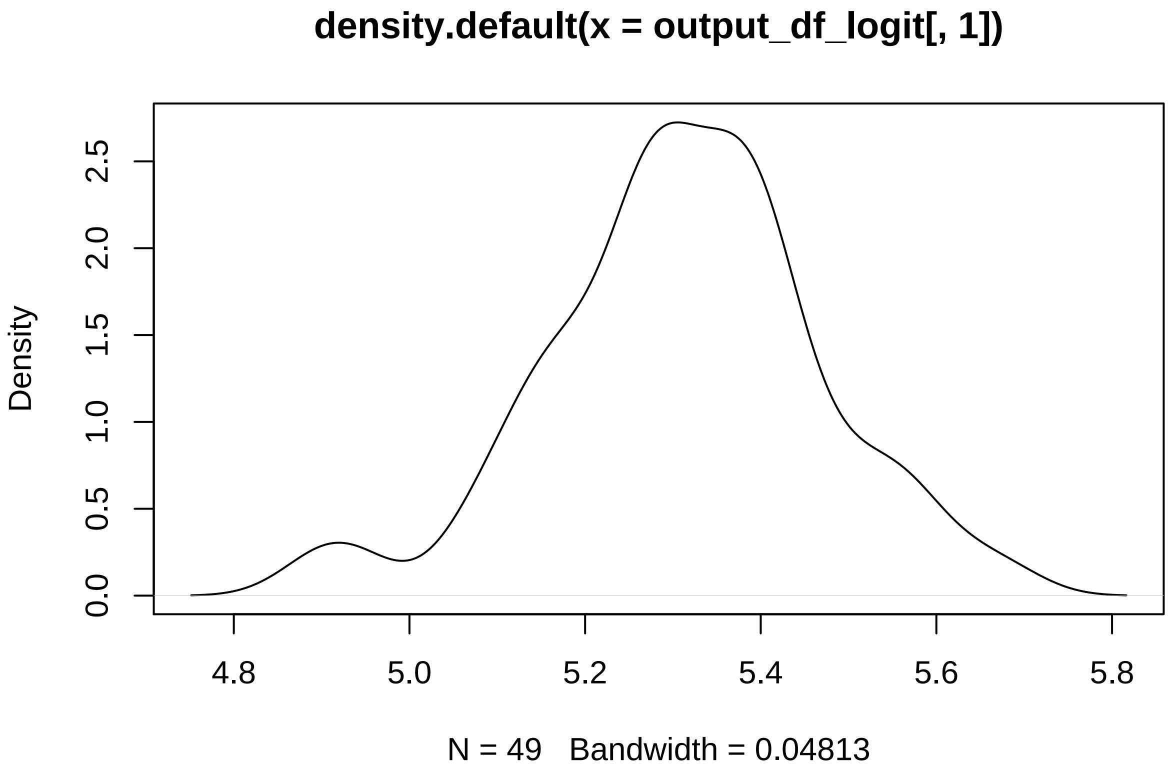


```
# logit model

L_logit = function(beta, X, Y){
  z = X %*% beta
  p = exp(z)/(1 + exp(z))
  p[p==1] = 0.9999
  p[p==0] = 0.0001
  L = Y * log(p) + (1 - Y) * log( 1 - p)
  -sum(L)
}

coef_est_logit = function(data){
  X_s = as.matrix(data[, 1:4])
  Y_s = data[, 7]
  fit2 = optim(par = c(0, 0, 0, 0), fn = L_logit, X = X_s, Y = Y_s)
  par2 = fit2$par
}
```

```r
output = list()
n_sample = 49
for (i in 1:n_sample){
  df = sample_n(d1, nrow(d1), replace = T)
  output[[i]] = coef_est_logit(data = df)
  message(i, " of ", n_sample)
}

output_df_logit = do.call(rbind, output)
output_df_logit
```

```
##           [,1]     [,2]      [,3]       [,4]
##  [1,] 5.362981 2.299242 -1.686775 0.16968268
##  [2,] 5.142184 2.220524 -1.626190 0.26180921
##  [3,] 5.120797 2.250737 -1.647219 0.34531295
##  [4,] 5.396895 2.220766 -1.662423 0.20512577
##  [5,] 4.941104 2.118629 -1.558777 0.27637937
##  [6,] 5.381430 2.216394 -1.673278 0.20171416
##  [7,] 5.073105 2.190213 -1.600616 0.31241735
##  [8,] 5.312900 2.217707 -1.655061 0.06139579
##  [9,] 5.671835 2.291701 -1.733750 0.27960177
## [10,] 5.472239 2.206244 -1.671907 0.29561729
## [11,] 5.345566 2.314149 -1.683482 0.27287428
## [12,] 5.284696 2.389214 -1.700683 0.29114799
## [13,] 5.262862 2.164821 -1.637600 0.33135329
## [14,] 5.265081 2.193141 -1.629758 0.13885276
## [15,] 5.307028 2.136356 -1.634088 0.32736108
## [16,] 5.158042 2.249316 -1.638131 0.26576462
## [17,] 5.417900 2.179752 -1.653448 0.24222992
## [18,] 5.283647 2.475122 -1.723380 0.21746294
## [19,] 5.183268 2.237766 -1.644144 0.24800441
## [20,] 5.546193 2.142636 -1.662284 0.11773586
## [21,] 5.190505 2.288776 -1.655614 0.23276724
## [22,] 5.541018 2.124860 -1.656745 0.36555206
## [23,] 5.430836 2.292937 -1.697888 0.26415783
## [24,] 5.300877 2.200844 -1.639409 0.30906847
## [25,] 5.599137 2.053994 -1.629348 0.24243364
## [26,] 5.249540 2.258476 -1.659095 0.29344942
## [27,] 5.080413 2.227085 -1.616307 0.22480883
## [28,] 5.389279 2.127986 -1.629543 0.24777709
## [29,] 5.277349 2.185829 -1.628251 0.15503496
## [30,] 5.494058 2.171517 -1.669421 0.20327609
## [31,] 5.260497 2.369536 -1.687471 0.25353879
## [32,] 5.334091 2.273420 -1.667573 0.23612414
## [33,] 5.458551 2.145443 -1.652477 0.22219477
## [34,] 5.399635 2.155298 -1.637241 0.23453248
## [35,] 5.414102 2.261160 -1.682972 0.22035672
## [36,] 5.196736 2.221603 -1.617918 0.07739983
## [37,] 5.257553 2.381986 -1.684989 0.16628678
## [38,] 5.296578 2.198273 -1.626469 0.22527634
## [39,] 5.145220 2.173534 -1.600361 0.16964862
## [40,] 5.364895 2.169922 -1.640310 0.25693324
## [41,] 5.387692 2.132587 -1.639623 0.34294780
## [42,] 5.385082 2.396220 -1.728788 0.24894794
## [43,] 5.349722 2.182273 -1.653985 0.27455588
## [44,] 4.895855 2.322836 -1.606969 0.14735802
## [45,] 5.171784 2.191597 -1.620241 0.22264549
## [46,] 5.240814 2.317981 -1.677836 0.36012635
## [47,] 5.563555 2.176301 -1.679305 0.18052806
## [48,] 5.278342 2.219133 -1.654426 0.30752208
## [49,] 5.391459 2.162451 -1.635693 0.13964320
```

```
plot(density(output_df_logit[, 1]))
```


density.default(x = output_df_logit[, 1])
Density
N = 49 Bandwidth = 0.04813