Kleisle-Murphy Lab 4

Isaac Kleisle-Murphy

February 23, 2020

First, we re-run the synthesis model for `CEsample`, just as we did in Labs 2-3.

```
suppressMessages(require(dplyr))
suppressMessages(require(ggplot2))
suppressMessages(require(runjags))
suppressMessages(require(coda))
suppressMessages(require(tidyr))
suppressMessages(require(fastDummies))
options(warn = -1)
setwd("~/Documents/Swat_2020/Data_Privacy/Data-Confidentiality/datasets")

CEsample = read.csv("CEsample.csv")%>%
  mutate(logInc = log(TotalIncomeLastYear),
         logExp = log(TotalExpLastQ))


the_data = list("logInc" = CEsample$logInc,
                "logExp" = CEsample$logExp,
                "r" = CEsample$Race,
                "R" = max(CEsample$Race),
                "u" = CEsample$UrbanRural,
                "U" = max(CEsample$UrbanRural),
                "N" = nrow(CEsample),
                "mu_b0" = 0,
                "mu_b1" = 0,
                "prec_b0" = 1,
                "prec_b1" = 1
                )


the_formula = "

model{

#model
for (i in 1:N){
  logInc[i] ~ dnorm(B_0[r[i], u[i]] + B_1[r[i], u[i]]*logExp[i], inv_sigma_sq[r[i]])
}

#priors
for (rc in 1:R){
  for (ur in 1:U){
    B_0[rc, ur] ~ dnorm(b_0, tau_sq_0)
```

```
    B_1[rc, ur] ~ dnorm(b_1, tau_sq_1)
  }
  inv_sigma_sq[rc] ~ dgamma(1,1)
  sigma[rc] <- sqrt(pow(inv_sigma_sq[rc], -1))
}

#hyperpriors
b_0 ~ dnorm(mu_b0,prec_b0)
b_1 ~ dnorm(mu_b1,prec_b1)
tau_sq_0 ~ dgamma(1,1)
tau_sq_1 ~ dgamma(1,1)
}

"




initsfunction <- function(chain){
  .RNG.seed <- c(1,2)[chain]
  .RNG.name <- c("base::Super-Duper",
                 "base::Wichmann-Hill")[chain]
  return(list(.RNG.seed=.RNG.seed,
              .RNG.name=.RNG.name))
}


posterior_jags  <- run.jags(the_formula,
                            n.chains = 2,
                            data = the_data,
                            monitor = c("B_0", "B_1", "sigma"),
                            adapt = 1000,
                            burnin = 5000,
                            sample = 2500,
                            thin = 100,
                            inits = initsfunction)



#extract, store in dictionary for quicker retrieval
posterior_df = data.frame(as.mcmc(posterior_jags))
posterior_list = lapply(colnames(posterior_df), function(x) posterior_df%>%pull(x))%>%
  `names<-`(colnames(posterior_df))
```

As before, we also re-initialize our synthesis helper functions. Note that `synth_helper()` generates a single synthetic dataset, while

```
synth_helper <- function(posterior_list, df, ii = NULL){

  #randomly select posterior draw to use

  if (is.null(ii)){
    ii = sample(1:nrow(df), 1)
  }
```

```r
  beta_suffixes = paste0(".", df$Race, ".", df$UrbanRural, ".")
  sigma_suffixes = paste0(".", df$Race, ".")

  b0 = sapply(beta_suffixes, function(x) posterior_list[[paste0("B_0", x)]][ii])%>%as.vector()
  b1 = sapply(beta_suffixes, function(x) posterior_list[[paste0("B_1", x)]][ii])%>%as.vector()


  mu_post = b0+b1*df%>%pull(logExp)
  sig_post = sapply(sigma_suffixes, function(x) posterior_list[[paste0("sigma", x)]][ii])%>%as.vector()

  result = rnorm(nrow(df), mu_post, sig_post)

  return(result)
}


synth_logInc <- function(posterior_list, df, m=1){

  set.seed(4*m-1)
  sample_idx = sample(1:length(posterior_list[[1]]), m, replace = F)

  lapply(sample_idx,
         function(y)
           synth_helper(posterior_list, df, ii = y)
         )%>%
    return()
}
```

Using these functions, we take $m = 20$ synthetic draws of `logInc`.

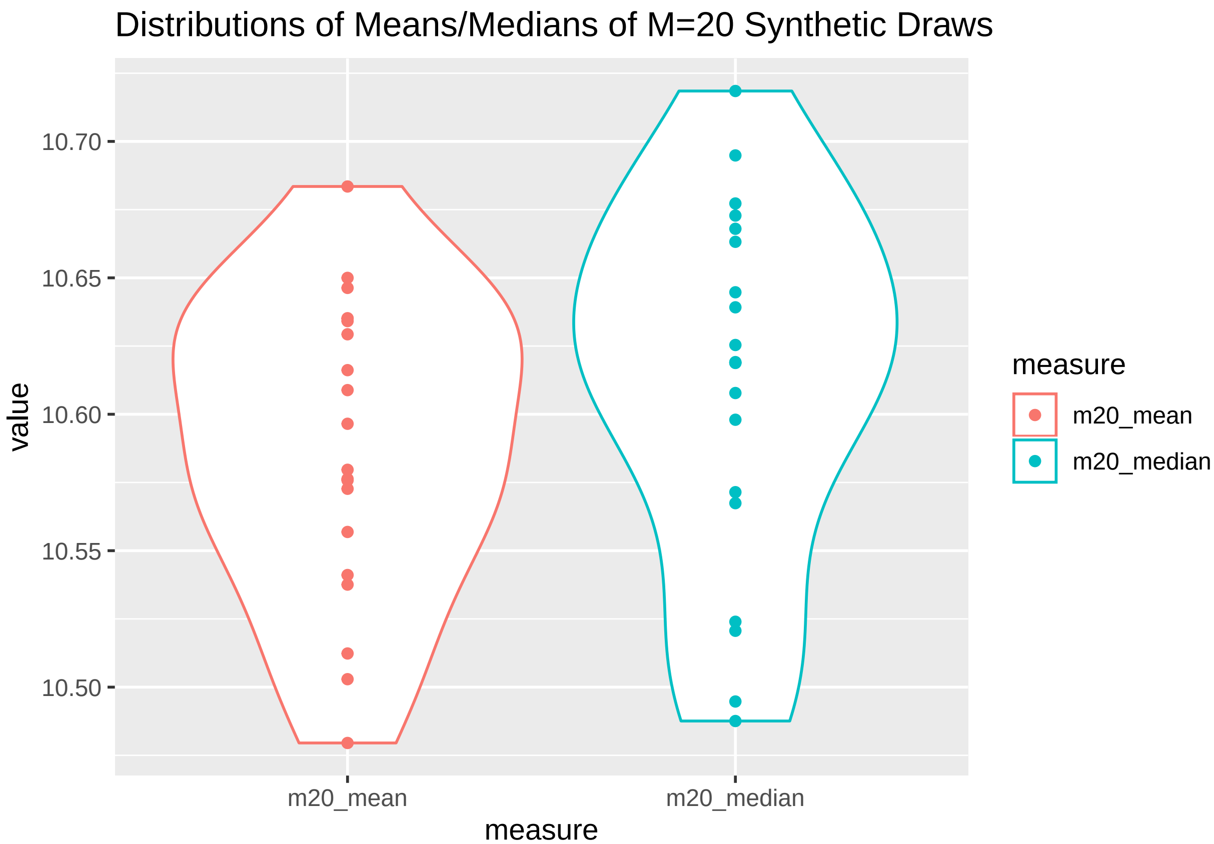
```r
m20 = synth_logInc(posterior_list, CEsample, m = 20)
```

ii/iii.)

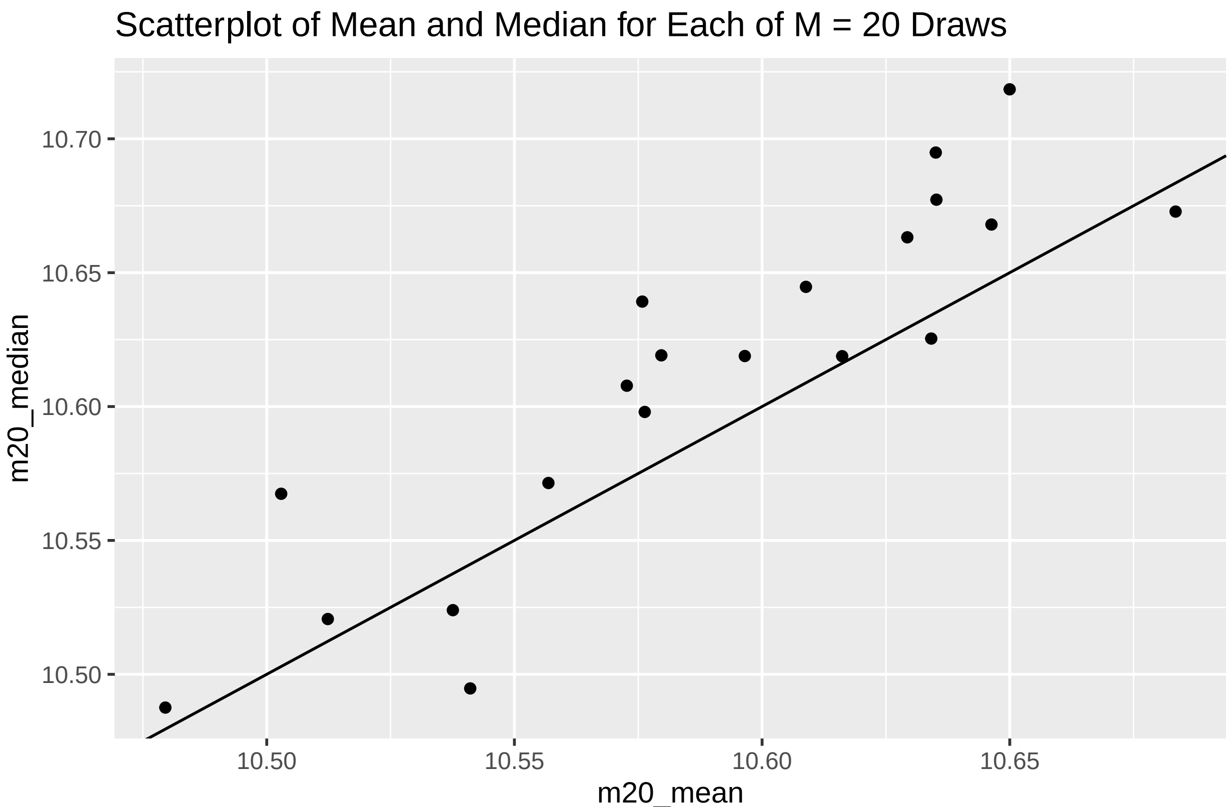
For each of these synthetic draws, we then compute the following analysis-specific utility measures: mean, median, ρ, and β_0 and β_1 regression coefficients

```r
m20_mean = lapply(m20, mean)%>%unlist()
m20_median = lapply(m20, median)%>%unlist()
m20_rho = lapply(m20, cor, CEsample$logInc)%>%unlist()
m20_var = lapply(m20, var)%>%unlist()

ggplot(data.frame(m20_mean, m20_median)%>%
         gather(measure,value),
       aes(x = measure, y = value, color = measure))+
  geom_violin() +
  geom_point() +
  labs(title = "Distributions of Means/Medians of M=20 Synthetic Draws")
```

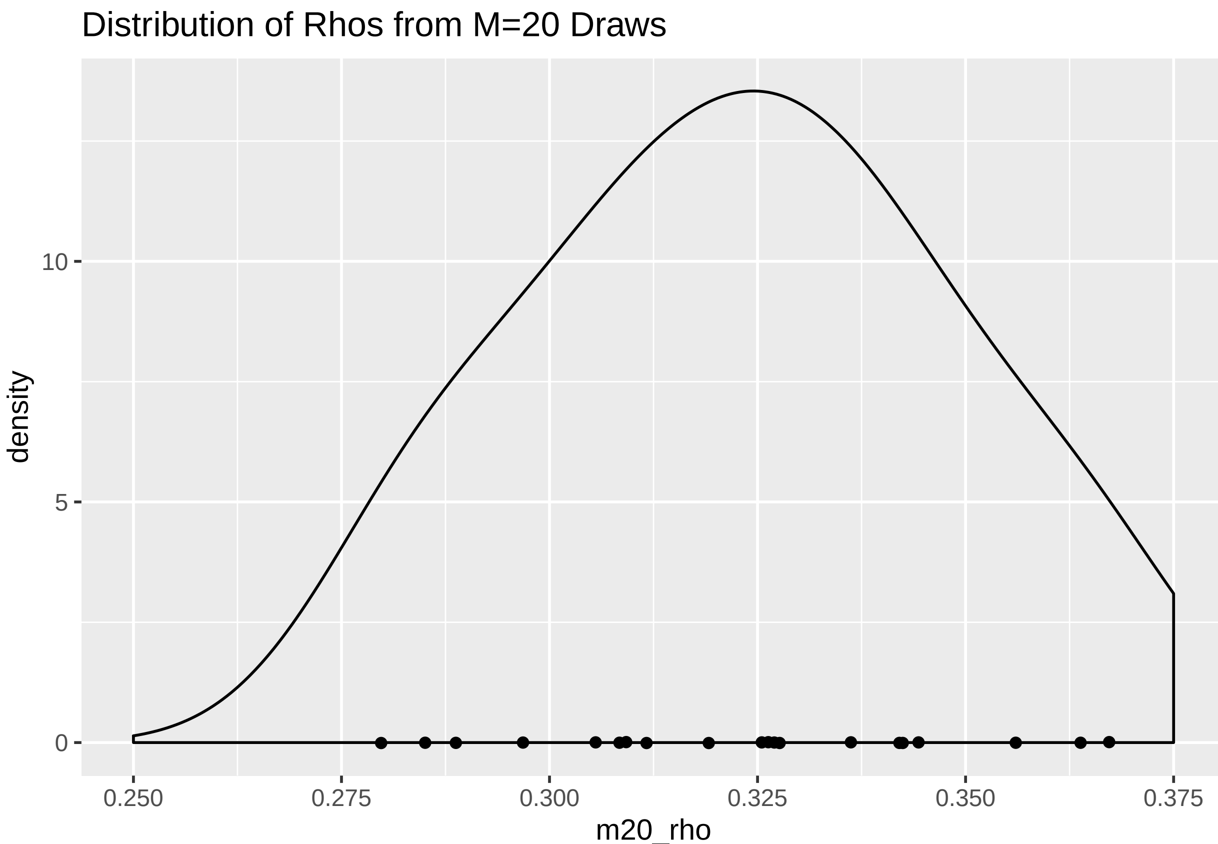



```
ggplot(data.frame(m20_mean, m20_median),
       aes(x = m20_mean, y = m20_median))+
  geom_point() +
  labs(title = "Scatterplot of Mean and Median for Each of M = 20 Draws",
       caption = "Each draw represents the mean and median pair from one of the 20 draws. Line y=x for r
  geom_abline(slope  =1)
```



Each draw represents the mean and median pair from one of the 20 draws. Line y=x for reference.

```
ggplot(data.frame(m20_rho = m20_rho, y = 0),
       aes(x = m20_rho))+
  geom_density()+xlim(c(.25, .375))+
  geom_jitter(aes(m20_rho,y), height = 0.01) +
  labs(title = "Distribution of Rhos from M=20 Draws")
```



Each dot represents the mean or median of one of the 20 synthetic draws. Interestingly, we see that the medians of a synthetic draw was often greater than the mean of that draw. Additionally, ρ values tend to fall in the $.27 - .35$ range.

Further, if the measure Q is mean, we have:

```
q_m = mean(m20_mean);
b_m = (1/length(m20_mean))*sum((m20_mean - q_m)^2);
u_m = mean(m20_var)
q_m; b_m; u_m
```

```
## [1] 10.58849
```

```
## [1] 0.00284826
```

```
## [1] 1.394619
```

This yields a synthetic confidence interval of:

```
t = b_m/length(m20) + u_m;
cat("CI: ", q_m - sqrt(t), q_m + sqrt(t))
```

```
## CI:  9.407491 11.76949
```

```
ci_synth_mean = c(q_m - sqrt(t), q_m + sqrt(t))
```

Likewise, if the measure Q is median, we have:

```
q_m = mean(m20_median);
b_m = (1/length(m20_median))*sum((m20_median - q_m)^2);
u_m = mean(m20_var)
q_m; b_m; u_m
```

```
## [1] 10.61161
## [1] 0.004160784
## [1] 1.394619
```

This yields a confidence interval of:

```
t = b_m/length(m20) + u_m;
cat("CI: ", q_m - sqrt(t), q_m + sqrt(t))
```

```
## CI:  9.430586 11.79264
```

```
ci_synth_median = c(q_m - sqrt(t), q_m + sqrt(t))
```

And finally, if the measure Q is the regression coefficient for an SLR in which Y is the true `logInc` and X is a synthetic draw of `logInc`, we have

```
m20_lms = lapply(m20, function(x) lm(CEsample$logInc ~ x)%>%summary())
#estimates
m20_b1 = lapply(m20_lms, function(x) x$coefficients[[2]])%>%unlist()
#variances of estimate
m20_b1_var = lapply(m20_lms, function(x) x$coefficients[[4]]^2)%>%unlist()

q_m = mean(m20_b1);
b_m = (1/length(m20_b1))*sum((m20_b1 - q_m)^2);
u_m = mean(m20_b1_var)
q_m; b_m; u_m
```

```
## [1] 0.3156762
## [1] 0.0004544441
## [1] 0.0008637687
```

This yields a confidence interval of:

```
t = b_m/length(m20) + u_m;
cat("CI: ", q_m - sqrt(t), q_m + sqrt(t))
```

```
## CI:  0.2859022 0.3454502
```

3.)

Having generated CIs, via Dreschler's method, based upon our $m = 20$ synthetic draws, we turn to interval overlap measures. We first need to get CIs for the estimands (mean, median, slope) based on the true data, so we compute:

```
ci_mean_true = c(mean(CEsample$logInc) + qnorm(.025)*sd(CEsample$logInc),
                 mean(CEsample$logInc) + qnorm(.975)*sd(CEsample$logInc))

ci_median_true = c(median(CEsample$logInc) + qnorm(.025)*sd(CEsample$logInc),
                   median(CEsample$logInc) + qnorm(.975)*sd(CEsample$logInc))
```

Notably, since the regression coefficient above is between the synthetic and true datasets, we cannot perform this analysis for regression coefficient (as the true data regressed against itself would give slope = 1 with sd = 0), so we omit this estimand from interval overlap analysis.

Next, we compute interval overlaps for mean and median, in accordance with Dreschler et al.'s methods:

```r
intOv <- function(synthCI, obsCI){

  intCI = c(max(synthCI[1], obsCI[1]), min(synthCI[2], obsCI[2]))

  I = (intCI[2]-intCI[1])/(2*(synthCI[2] - synthCI[1])) + (intCI[2]-intCI[1])/(2*(obsCI[2] - obsCI[1]))

  return(I)
}
```

For the mean, we have:

```r
intOv(ci_synth_mean,
      ci_mean_true)
```

```
## [1] 0.7611913
```

For the median, we have:

```r
intOv(ci_synth_median,
      ci_mean_true)
```

```
## [1] 0.7611975
```

Both of these scores are thoroughly mediocre – a good reflection of my model!

4.)

From before, we retrieve the Age and Disability data set from the City of Seattle (specifically focusing on neighborhoods in the immediate city). In this model, I will attempt to synthesize the following variables that otherwise could reveal (or narrow down) the identity of someone receiving disability services from the City: `AgeRange`, `Veteran`, and `RaceCode`. Notably, I do not synthesize neighborhood (as these are generally pretty big) – however, neighborhood helps drive the prior for each model I fit.

```r
setwd("~/Downloads")
ageDisData <- read.csv("Aging_and_Disability_Services_-_Client_Level_Data_June_2016.csv")
suppressMessages(require(stringr))

cat_vars = c("GeographicLocation", "AgeRange", "Veteran", "HouseholdWithChildren")
code_vars = c("RaceCode")
count_vars = c("NumberofChildren")

ageDisData_wr = ageDisData%>%
  filter(grepl("Seattle Neighborhoods", GeographicLocation),
         grepl("to", AgeRange),
         !is.na(RaceCode),
         !is.na(NumberofChildren))%>%
  mutate_at(cat_vars, as.character)%>%
  mutate_at(cat_vars, function(x) ifelse(x == " "|x=="", "U", x))%>%
  dplyr::select(c("ClientID", code_vars, count_vars, cat_vars))%>%
  mutate(GeographicLocation = str_replace_all(GeographicLocation, "Seattle Neighborhoods: ", ""),
         GeographicLocation = str_replace_all(GeographicLocation, " ", ""))%>%
  group_by(ClientID)%>%
  slice(1)%>%
  ungroup()
```

The model for race given neighborhood is

$$P(Race = r|Neighborhood = j) \sim Multinom(N_j, \vec{p_j}),$$

with dirichlet priors

$$\vec{p_j} \sim Dirichlet(\langle 1, \ldots, 1 \rangle).$$

In refiend versions of this model, the dirichlet priors for each of the $\vec{p_j}$ could be parametrized by the counts of each race, $N_{1j}, \ldots, N_{Rj}$, in neighborhood j. This would give us a more informed prior. However, for now, we will use a more weakly informed prior of $\langle 1, \ldots, 1 \rangle$.

We first transform the data to prepare for multinomial-dirichlet modeling.

```
loc_dict = data.frame(GeographicLocation = unique(sort(ageDisData_wr$GeographicLocation)),
                      loc = 1:length(unique(ageDisData_wr$GeographicLocation)))

race_mod_df = ageDisData_wr%>%
  group_by(GeographicLocation, RaceCode)%>%
  summarise(N = n())%>%
  ungroup()%>%
  arrange(RaceCode)%>%
  mutate(RaceCode = as.factor(RaceCode),
         GeographicLocation = as.factor(GeographicLocation))%>%
  pivot_wider(names_from = RaceCode, names_prefix = "race", values_from = N)%>%
  left_join(loc_dict, by = c("GeographicLocation"))

race_mod_df[is.na(race_mod_df)] = 0
race_mod_df$Nj = rowSums(race_mod_df%>%dplyr::select(race0:race8))
  #fastDummies::dummy_cols(., c("GeographicLocation", "RaceCode"), remove_first_dummy = T, remove_selec

race_mod_df%>%head(15)
```

```
## # A tibble: 12 x 12
##    GeographicLocat... race0 race1 race2 race3 race4 race5 race6 race7 race8   loc
##    <fct>              <int> <dbl> <int> <int> <dbl> <dbl> <int> <int> <int> <int>
##  1 Ballard                9     9    24    26     0     0   204     7     3     1
##  2 CapitolHill           19     0    26    54     0     8   144     4     3     2
##  3 CentralSeattle        22     0   116   141     0     0    80     3     6     3
##  4 Delridge              26    12   290   175    13    19   211    18     7     4
##  5 Downtown              18    43   748   209     0    66   427    31    14     5
##  6 Duwamish              13     0   507   114     0     9    45    16     1     6
##  7 LakeUnion             13     0    40    11     0     0   124     2     1     7
##  8 NESeattle             25     0    61    29     0     6   246     4     6     8
##  9 NorthSeattle          32     0   101    60     0     5   257    13     7     9
## 10 QueenAnne             16     0    33    37     0    12   185     4     2    10
## 11 SESeattle             54     9  1296   479    28    22   216    37    16    11
## 12 SWSeattle             18     0    24    14     0     0   121     3     4    12
## # ... with 1 more variable: Nj <dbl>
```

We then fit the model.

```
the_data = list()
the_data$Nj = race_mod_df$Nj
the_data$loc = race_mod_df$loc
the_data$J = max(race_mod_df$loc)
the_data$Njr = race_mod_df%>%dplyr::select(race0: race8)%>%as.matrix()%>%unname()
the_data$ones = rep(1, 9)
```

```r
model_str = "

model{

for (j in 1:J){

Njr[j, 1:9] ~ dmulti(p[j, 1:9], Nj[j])
p[j, 1:9] ~ ddirch(ones[])

}


}

"

jags_fit_race_mod = run.jags(model = model_str, data = the_data, n.chains = 2, monitor = c("p"))
```

```
## Warning: No initial values were provided - JAGS will use the same initial values
## for all chains

## Calling the simulation...
## Welcome to JAGS 4.3.0 on Tue Feb 25 12:22:11 2020
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 12
##    Unobserved stochastic nodes: 12
##    Total graph size: 59
##
## WARNING: Unused variable(s) in data table:
## loc
##
## . Initializing model
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 4000
## -------------------------------------------------| 4000
## ************************************************** 100%
## . . Updating 10000
## -------------------------------------------------| 10000
## ************************************************** 100%
## . . . . . Updating 0
## . Deleting model
## .
## Note: the model did not require adaptation
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Note: Summary statistics were not produced as there are >50 monitored
## variables
```

```
## [To override this behaviour see ?add.summary and ?runjags.options]
## FALSEFinished running the simulation
```

```
post_race_mod_df = data.frame(as.mcmc(jags_fit_race_mod))
```

```
## Warning in as.mcmc.runjags(jags_fit_race_mod): Combining the 2 mcmc chains
## together
```

```
#plot(jags_fit_race_mod_df)
```

Now, our $Race|Neighborhood$ model has been fit. We proceed outwards, to include `veteran` in the joint density. Since Seattle – at least the City itself – does not have much of a military presence, we will assume that `veteran` is independent of neighborhood. However, we will use `Race` as a covariate in veteran status, thus giving (note that $v \in \{0,1,2\}$, where 0 is non-veteran, 1 is veteran, and 2 is unknown)

$$P(Veteran = v|Race = r, Neighborhood = j) = P(Veteran = v|Race = r).$$

For this, we will use a multinomial logistic regression where X denotes race, i.e.

$$log(p_{iv}/p_{i1}) \sim B_{0v} + B_{1v}X_i$$

with priors

$$B_{0v} \sim N(0,1), B_{1v} \sim N(0,1).$$

Note that non-veteran is the holdout category.

As before, we transform the data to accomodate this model.

```
cat_vars = c("GeographicLocation", "AgeRange", "Veteran")
code_vars = c("RaceCode")
count_vars = c("NumberofChildren")

vet_mod_df = ageDisData_wr%>%
  mutate(vet_code = ifelse(Veteran == "N", 0,
                           ifelse(Veteran == "Y", 1, 2)))%>%
  arrange(RaceCode)%>%
  dplyr::select(RaceCode, vet_code)%>%
  fastDummies::dummy_cols(., c("RaceCode"), remove_first_dummy = T, remove_selected_columns = F)%>%
  arrange(vet_code)%>%
  fastDummies::dummy_cols(., c("vet_code"), remove_first_dummy = F, remove_selected_columns = F)
```

We then fit the model in JAGS.

```
the_data = list()
the_data$N = nrow(vet_mod_df)
the_data$Y = vet_mod_df%>%dplyr::select(vet_code_0:vet_code_2)%>%as.matrix()%>%unname()
the_data$C = max(vet_mod_df$vet_code)+1 #because indexed starting at 0
the_data$ones = rep(1, max(vet_mod_df$RaceCode)) #note race 0 is held out
the_data$zeros = rep(0, max(vet_mod_df$RaceCode))
the_data$I = diag(max(vet_mod_df$RaceCode))
the_data$X = vet_mod_df%>%dplyr::select(RaceCode_1:RaceCode_8)%>%as.matrix()%>%unname()

model_str = "
model {

for (i in 1:N){
```

```
Y[i,1:C] ~ dmulti(p[i,1:C],1)

for (c in 1:C){
p[i,c] <- q[i,c]/sum(q[i,1:C])
log(q[i,c]) <- beta0[c] + sum(beta1[c, 1:8]*X[i,1:8])
}

}


beta0[1] <- 0
beta1[1,1:8] <- zeros

for (c in 2:C){

beta0[c] ~ dnorm(0, 1)
beta1[c, 1:8] ~ dmnorm(zeros, I)

}
}

"


vet_mod_jags = run.jags(model_str, data = the_data, n.chains = 2, monitor = c("beta0", "beta1"),
                        burnin = 1000, sample = 500)
post_vet_mod_df = data.frame(as.mcmc(vet_mod_jags))

saveRDS(vet_mod_jags, "~/Documents/swat_images/vet_mod_jags.RDS")
#plot(vet_mod_jags)
```

Again, move outwards, such that our model now includes `AgeRange` in the joint density. Since one's age could plausibly tied/predicted by one's race in a neighborhood, as well as whether or not they are a veteran, we are in search of

$$P(AgeRange = k|Race = r, Neighborhood = j, Veteran = v).$$

Again, we will use a multinomial logistic regression. Importantly, Y describes an indicator vector for `AgeRange`, X_1 describes an indicator vector for `GeographicLocation` (i.e. neighborhood), X_2 describes an indicator vector for `Race`, and X_3 describes an indicator vector for `Veteran`. Further, we do not use interaction terms, because a.) the interaction of 9 races and 12 locations adds a lot of variables, and b.) we risk overfit, and too granular of interaction could replicate the vulnerable combinations of these variables in the dataset in synthetic draws as the model. As such, this is a first order multinomial logistic regression, with no interaction.

$$Y_i \sim Multinom(\vec{p_i}, 1) log(p_{ik}/p_{i1}) \sim B_{0k} + B_{1k}X_{1i} + B_{2k}X_{2i} + +B_{3k}X_{2i}.$$

with priors

$$B_{0v} \sim N(0,1) B_{1k} \sim N(0,1), B_{2k} \sim N(0,1), B_{3k} \sim N(0,1).$$

Note that non-veteran is the holdout category.

As before, we transform the data to accomodate this model.

```
age_dict = data.frame(AgeRange = sort(unique(ageDisData_wr$AgeRange)),
                      age_code = 1:length(unique(ageDisData_wr$AgeRange)))
```

```
age_mod_df = ageDisData_wr%>%
  mutate(vet_code = ifelse(Veteran == "N", 0,
                           ifelse(Veteran == "Y", 1, 2)))%>%
  left_join(loc_dict, by = c("GeographicLocation"))%>%
  left_join(age_dict, by = c("AgeRange"))%>%
  arrange(RaceCode)%>%
  fastDummies::dummy_cols(., c("RaceCode"), remove_first_dummy = T, remove_selected_columns = F)%>%
  arrange(vet_code)%>%
  fastDummies::dummy_cols(., c("vet_code"), remove_first_dummy = T, remove_selected_columns = F)%>%
  arrange(loc)%>%
  fastDummies::dummy_cols(., c("loc"), remove_first_dummy = T, remove_selected_columns = F)%>%
  arrange(age_code)%>%
  fastDummies::dummy_cols(., c("age_code"), remove_first_dummy = F, remove_selected_columns = F)
```

```
## Warning: Column `GeographicLocation` joining character vector and factor,
## coercing into character vector

## Warning: Column `AgeRange` joining character vector and factor, coercing into
## character vector
```

And then we fit:

```
the_data = list()
the_data$N = nrow(age_mod_df)
the_data$Y = age_mod_df%>%dplyr::select_if(grepl("age_code_", colnames(.)))%>%as.matrix()%>%unname()
the_data$C = max(age_mod_df$age_code) #because indexed starting at 0
the_data$ones = rep(1, 100) #note race 0 is held out
the_data$zeros = rep(0, 100)
the_data$I = diag(100)

the_data$J = max(age_mod_df$loc)-1 #because of dropout (unlike RaceCode, which is indexed at 1)
the_data$X1 = age_mod_df%>%dplyr::select(paste0("loc_", 2:(max(age_mod_df$loc))))%>%as.matrix()%>%unname

the_data$R = max(age_mod_df$RaceCode) #indexed at 0, so already dropped
the_data$X2 = age_mod_df%>%dplyr::select(paste0("RaceCode_", 1:(max(age_mod_df$RaceCode))))%>%as.matrix

the_data$V = max(age_mod_df$vet_code) #already indexed 0:2
the_data$X3 = age_mod_df%>%dplyr::select(paste0("vet_code_", 1:(max(age_mod_df$vet_code))))%>%as.matrix


model_str = "
model {

for (i in 1:N){

Y[i,1:C] ~ dmulti(p[i,1:C],1)

for (c in 1:C){
p[i,c] <- q[i,c]/sum(q[i,1:C])
log(q[i,c]) <- beta0[c] + sum(beta1[c, 1:J]*X1[i,1:J]) + sum(beta2[c, 1:R]*X2[i,1:R]) + sum(beta3[c, 1:
}

}
```

```
beta0[1] <- 0
beta1[1,1:J] <- zeros[1:J]
beta2[1,1:R] <- zeros[1:R]
beta3[1,1:V] <- zeros[1:V]

for (c in 2:C){

beta0[c] ~ dnorm(0, 1)
beta1[c, 1:J] ~ dmnorm(zeros[1:J], I[1:J, 1:J])
beta2[c, 1:R] ~ dmnorm(zeros[1:R], I[1:R, 1:R])
beta3[c, 1:V] ~ dmnorm(zeros[1:V], I[1:V, 1:V])

}
}

"


age_mod_jags = run.jags(model_str, data = the_data, n.chains = 2, monitor = c("beta0", "beta1", "beta2"
                        burnin = 1000, sample = 250)
post_age_mod_df = data.frame(as.mcmc(age_mod_jags))

saveRDS(age_mod_jags, "~/Documents/swat_images/age_mod_jags.RDS")
#plot(vet_mod_jags)
```

Lastly, we add `HouseholdWithChildren` – a three-valued categorical variable which, when paired with veteran status, age, race, and neighborhood, poses disclosure risks – to our expanding joint density. Since this variable might be closely tied to age and neighborhood in thie data, we use these two as covariates in this final multinomial logistic regression. Importantly, race and veteran status are not covariates in this model, lest we unintentionally replicate certain identifiable rare age/race/child or age/veteran/child combinations within a particular neighborhood.

As such, this model is, where $Y \in \{0 = No, 1 = Yes, 2 = Unknown\}$ is `HouseholdWithChildren`, where X_1 describes an indicator vector for `GeographicLocation` (i.e. neighborhood), and where X_2 describes an indicator vector for `AgeRange`:

$$P(HouseholdWithChildren == c|AgeRange = k, GeographicLocation = j) \implies Y_i \sim Multinom(\vec{p_i}, 1) log(p_{ic}/p_{i1}) \sim B_{0c}+$$

We wrangle/dummify in the same manner as before:

```
child_mod_df = ageDisData_wr%>%
  mutate(child_code = ifelse(HouseholdWithChildren == "N", 0,
                             ifelse(HouseholdWithChildren == "Y", 1, 2)))%>%
  left_join(loc_dict, by = c("GeographicLocation"))%>%
  left_join(age_dict, by = c("AgeRange"))%>%
  arrange(loc)%>%
  fastDummies::dummy_cols(., c("loc"), remove_first_dummy = T, remove_selected_columns = F)%>%
  arrange(age_code)%>%
  fastDummies::dummy_cols(., c("age_code"), remove_first_dummy = T, remove_selected_columns = F)%>%
  arrange(child_code)%>%
  fastDummies::dummy_cols(., c("child_code"), remove_first_dummy = F, remove_selected_columns = F)
```

```
## Warning: Column `GeographicLocation` joining character vector and factor,
## coercing into character vector
```

```
## Warning: Column `AgeRange` joining character vector and factor, coercing into
## character vector
```

And then fit:

```
the_data = list()
the_data$N = nrow(child_mod_df)
the_data$Y = child_mod_df%>%dplyr::select_if(grepl("child_code_", colnames(.)))%>%as.matrix()%>%unname()
the_data$C = max(child_mod_df$child_code)+1 #because indexed starting at 0
the_data$ones = rep(1, 100) #note race 0 is held out
the_data$zeros = rep(0, 100)
the_data$I = diag(100)

the_data$J = max(child_mod_df$loc)-1 #because of dropout (unlike RaceCode, which is indexed at 1)
the_data$X1 = child_mod_df%>%dplyr::select(paste0("loc_", 2:(max(child_mod_df$loc))))%>%as.matrix()%>%un

the_data$K = max(child_mod_df$age_code)-1 #because of dropout (unlike RaceCode, which is indexed at 1)
the_data$X2 = child_mod_df%>%dplyr::select(paste0("age_code_", 2:(max(child_mod_df$age_code))))%>%as.ma



model_str = "
model {

for (i in 1:N){

Y[i,1:C] ~ dmulti(p[i,1:C],1)

for (c in 1:C){
p[i,c] <- q[i,c]/sum(q[i,1:C])
log(q[i,c]) <- beta0[c] + sum(beta1[c, 1:J]*X1[i,1:J]) + sum(beta2[c, 1:K]*X2[i,1:K])
}

}


beta0[1] <- 0
beta1[1,1:J] <- zeros[1:J]
beta2[1,1:K] <- zeros[1:K]

for (c in 2:C){

beta0[c] ~ dnorm(0, 1)
beta1[c, 1:J] ~ dmnorm(zeros[1:J], I[1:J, 1:J])
beta2[c, 1:K] ~ dmnorm(zeros[1:K], I[1:K, 1:K])

}
}

"


child_mod_jags = run.jags(model_str, data = the_data, n.chains = 2, monitor = c("beta0", "beta1", "beta2
                          burnin = 1000, sample = 250)
```

```r
saveRDS(child_mod_jags, "~/Documents/swat_images/child_mod_jags.RDS")
```

Finally, so as to blow up this submission at the end, here are the traceplots for the multinomial logistic synthetic models in Problem 4. Note that mixing for these models is generally poor, but that is because I ran them in short lengths and with minimal thinning, so that I could knit and submit in time. For the final project, I intend to let these chains run longer.

Veteran Model

```r
plot(vet_mod_jags)
```

Age Model

```r
plot(age_mod_jags)
```

Child Model

```r
plot(child_mod_jags)
```