NHISExploration

Sarah Boese

2/16/2020

```
library(ProbBayes)
library(dplyr)
library(ggplot2)
require(gridExtra)
library(reshape)
library(runjags)
library(coda)
library(tidyverse)
library(fastDummies)
crcblue <- "#2905a1"
```

I wnated to look into survey data pertaining to public health. I found the National Health Interview Suvey as an example. I don't beleive that any of the information here has been altered. However, some information can be left out for the protection of a survey participants privacy. For example, the only geographic information released about a participant is REGION which can take four arguments: NORTHEAST, MIDWEST, SOUTH, WEST.

```
NHISSampleAdult<-read.csv("samadult.csv")
```

Each piece of data from the SampleAdult sample is unique. Here we can see this because n_unique is equal to $number_rows$. We can attribute this phenomenon to the fineness of detail within the survey (the data has 742 variables).

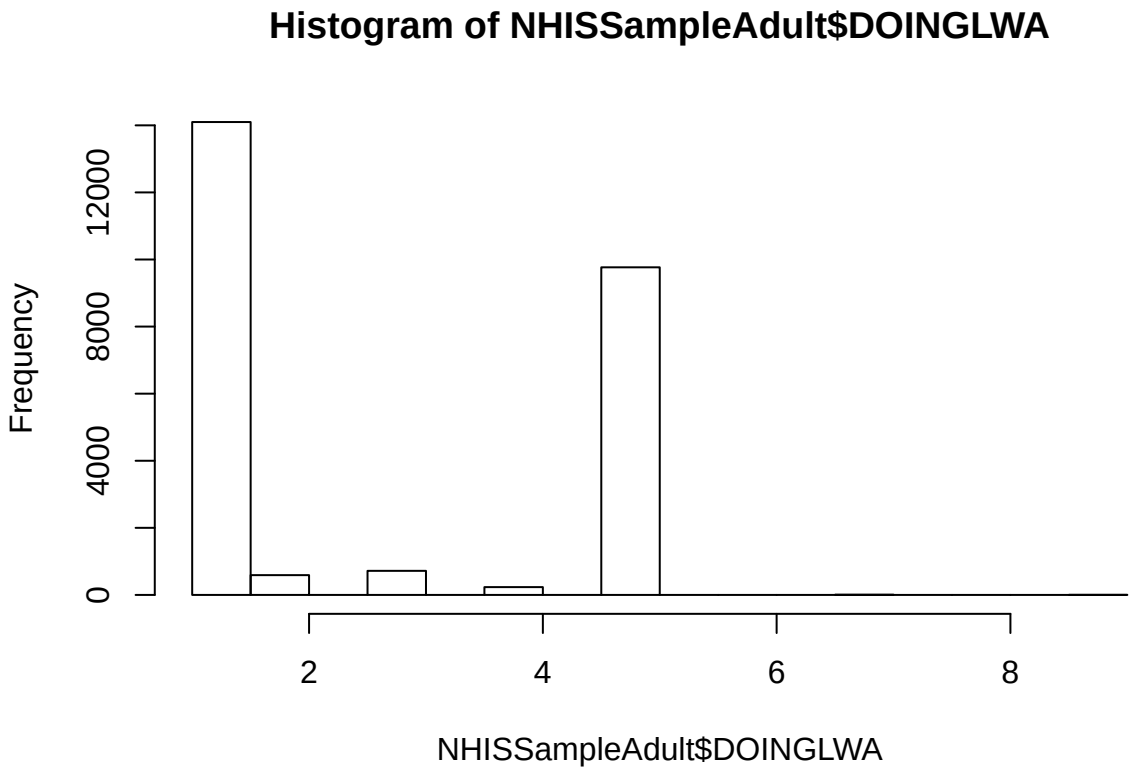
```
n_unique <- n_distinct(NHISSampleAdult)
n_unique
```

```
## [1] 25417
```

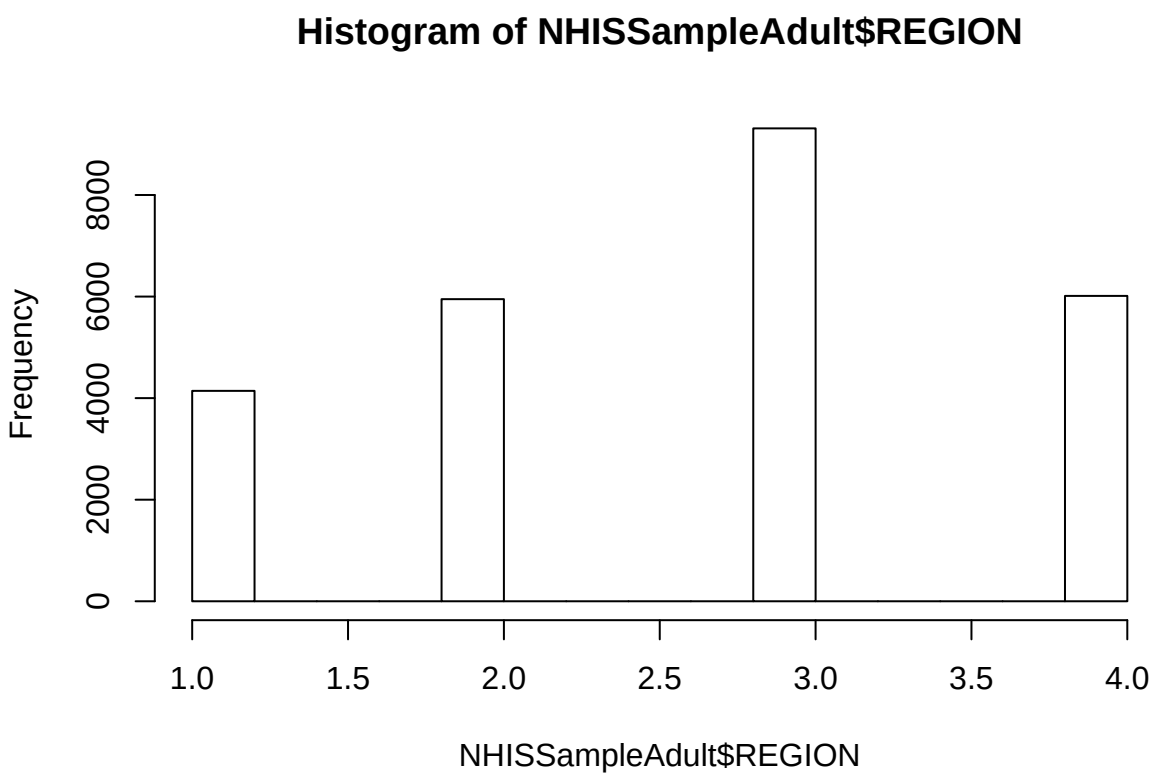
```
number_rows<- nrow(NHISSampleAdult)
number_rows
```

```
## [1] 25417
```

```
hist(NHISSampleAdult$DOINGLWA)
```




```
hist(NHISSampleAdult$REGION)
```



First, I wanted to see given certain information about demographics of participants in the survey if an intruder could find information about the health of specific participants. Here we have only two participants

who identify as Native American, Mexican-American women from the south. Clearly, there is some high level of attribute disclosure risk inthis data.

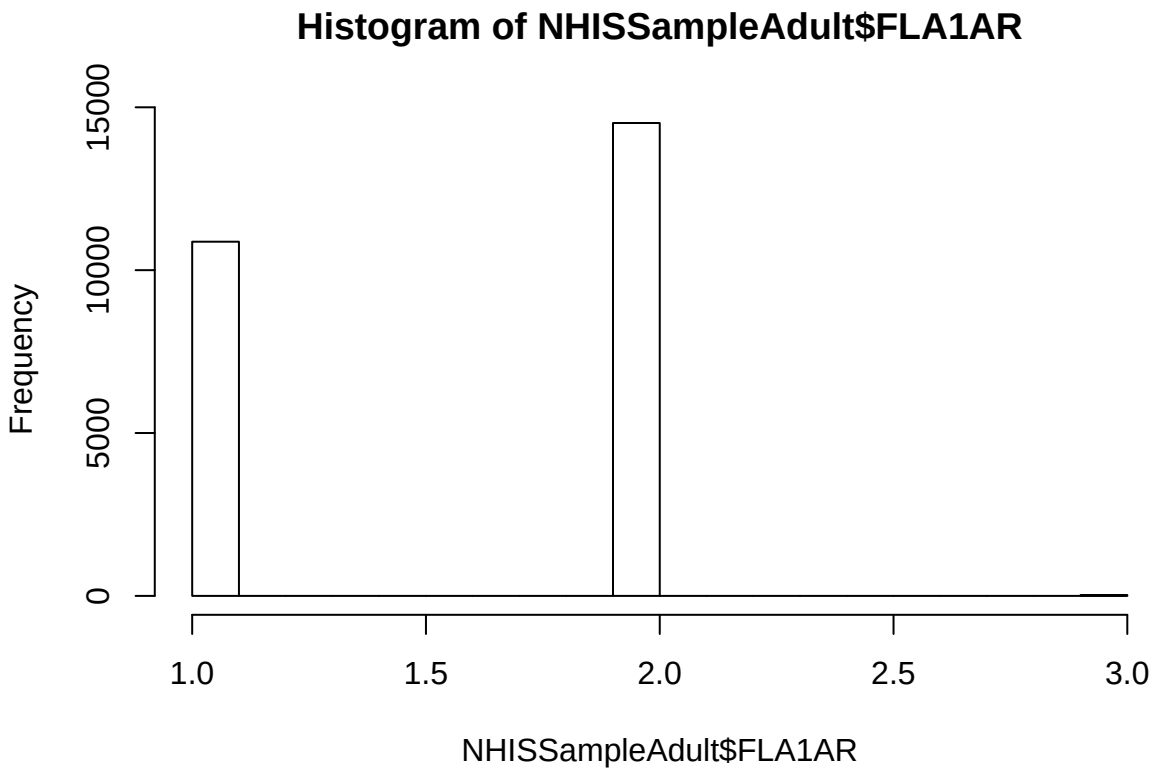
```
AIAN<- NHISSampleAdult %>%
  filter(RACERPI2 == 1 & REGION==1 & SEX == 2 & HISPAN_I == 3)
nrow(AIAN)
```

```
## [1] 2
```

```
nrow(AIAN)/nrow(NHISSampleAdult)
```

```
## [1] 7.868749e-05
```

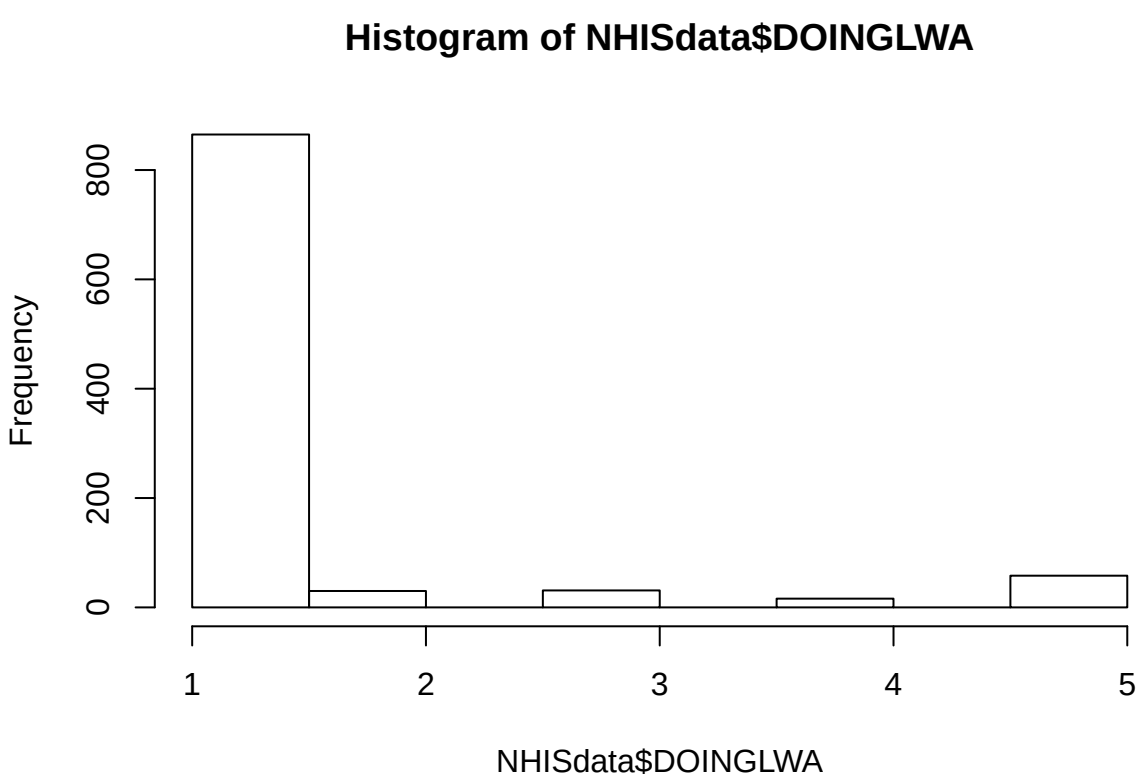
```
hist(NHISSampleAdult$FLA1AR)
```



I have decided to contain my synthesis model to a small portion of the 742 fields within the NHIS Sample adult survey.

```
NHISdata<- NHISSampleAdult %>%
  select(HHX, FPX, REGION, SEX, HISPAN_I, RACERPI2, AGE_P, DOINGLWA, WRKCATA, WKDAYR, FLA1AR, ALCNDRT) %
  filter(FLA1AR!=3 & DOINGLWA!=7 & DOINGLWA!=9 & WKDAYR<367) %>%
  sample_n(size=1000)
sig_WKDAYR<-sd(NHISdata$WKDAYR)^2
mu_WKDAYR<-mean(NHISdata$WKDAYR)
NHISdata<-NHISdata %>%
  mutate(WKDAYR_STD=(WKDAYR-mu_WKDAYR)/sig_WKDAYR)
```

```r
hist(NHISdata$DOINGLWA)
```




```r
sum(NHISdata$FLA1AR==2)/nrow(NHISdata)
```

```
## [1] 0.695
```

```r
sum(NHISdata$ALCNDRT==1)/nrow(NHISdata)
```

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

```r
sum(NHISdata$DOINGLWA==9)
```

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

```r
synthesize_lim <- function(index, n, post_lim){
  synthetic_lim <- rbinom(n, 1, post_lim[index,"p"])
  return(data.frame(synthetic_lim))
}

modelString_lim <-"
model {
## sampling
for (i in 1:N){
y[i] ~ dbern(p)
}
## priors
```

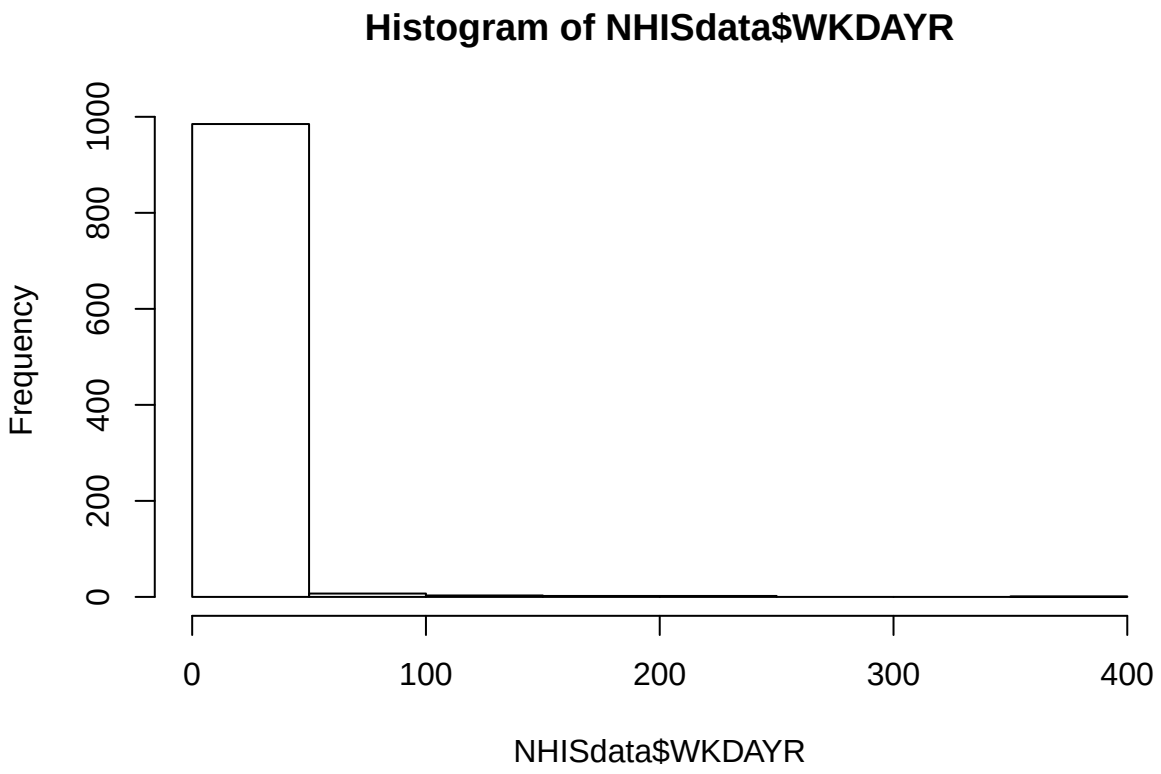
```
p ~ dbeta(57, 43)
}
"

limitations <- NHISdata$FLA1AR - 1
y_lim = as.vector(limitations)
N = length(y_lim)

the_data_lim <- list("y" = y_lim,
                     "N" = N, "mu0"=0)

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))
}

synthesize_lim_func<-function(){
posterior_lim <- run.jags(modelString_lim,
                      n.chains = 1,
                      data = the_data_lim,
                      monitor = c("p"),
                      adapt = 1000,
                      burnin = 5000,
                      sample = 5000,
                      thin = 1,
                      inits = initsfunction)


post_lim <- as.mcmc(posterior_lim)

n <- dim(NHISdata)[1]
syn_lim <- synthesize_lim(1, n, post_lim)
names(syn_lim)=c("synthesized_func_lim")
return(syn_lim)
}
```

```
 modelString_stat <-"
model {

## sampling
for (i in 1:N){
    y[i] ~ dcat(p[i, 1:C])
    p[i,1:C] ~ ddirch(alpha)
}

## priors
for (c in 1:C){
  alpha[c]<-1
}
```

```
}
"
```

```
y<-as.vector(NHISdata$DOINGLWA)
N_stat<-nrow(NHISdata)

the_data_stat <- list("y" = y,
                      "N" = N_stat, "C"=5)

synthesize_stat<-function(index, n_syn, p){
  synthetic_stat <- rmultinom(n=n_syn, size=1, prob = c(p[index,1], p[index, 2], p[index, 3], p[index, 4
  return(data.frame(t(synthetic_stat)))
}

synthesis_stat_func = function(){
posterior_stat <- run.jags(modelString_stat,
                           n.chains = 1,
                           data = the_data_stat,
                           monitor = c("p"),
                           adapt = 1000,
                           burnin = 5000,
                           sample = 1000,
                           thin = 1,
                           inits = initsfunction)

summary(posterior_stat)
post_stat <- as.mcmc(posterior_stat)
p<-as.matrix(post_stat)
syn_stat <- synthesize_stat(1000, N_stat, p)
names(syn_stat)=c("syn_emp_stat_1", "syn_emp_stat_2", "syn_emp_stat_3", "syn_emp_stat_4", "syn_emp_stat_
return(syn_stat)
}
```

```
hist(NHISdata$WKDAYR)
```




```
y_wrk=as.vector(NHISdata$WKDAYR)
sd(y_wrk)^2/mean(y_wrk)
```

```
## [1] 96.78255
```

```
synthesize_wrk <- function(X, index, n, post_wrk){
  lambda <- exp(post_wrk[index, "beta0"] + X$x_lim * post_wrk[index, "beta1"] + X$x_stat_1 * post_wrk[in
  + X$x_stat_4 * post_wrk[index, "beta5"]
  + X$x_stat_5 * post_wrk[index, "beta6"])
  synthetic_Y <- rpois(n, lambda)
  data.frame(synthetic_Y)
}

modelString_wrk <-"
model {
## sampling
for (i in 1:N){
y[i] ~ dpois(lambda[i])
log(lambda[i]) <- beta0 + beta1*x_lim[i] +
  beta2*x_stat_1[i] + beta3*x_stat_2[i] +
  beta4*x_stat_3[i] + beta5*x_stat_4[i] +
  beta6*x_stat_5[i]
}
## priors
beta0 ~ dnorm(mu0, g0)
beta1 ~ dnorm(mu1, g1)
beta2 ~ dnorm(mu2, g2)
beta3 ~ dnorm(mu3, g3)
```

```
beta4 ~ dnorm(mu4, g4)
beta5 ~ dnorm(mu5, g5)
beta6 ~ dnorm(mu6, g6)
}
"

synthesis_wrk_func<- function(syn_lim, syn_stat){
  y_wrk=as.vector(NHISdata$WKDAYR)
  x_no_lim<-syn_lim[,1]
  syn_lim = syn_lim %>%
    mutate(x_lim_data = if_else(synthesized_func_lim==1, 0, 1))
  x_lim<-as.vector(syn_lim$x_lim_data)
  x_stat<-data.matrix(syn_stat)
  x_stat_1<-x_stat[,1]
  x_stat_2<-x_stat[,2]
  x_stat_3<-x_stat[,3]
  x_stat_4<-x_stat[,4]
  x_stat_5<-x_stat[,5]
  N_wrk=nrow(NHISdata)


  the_data_wrk <- list("y" = y_wrk, "x_lim" = x_lim,
    "x_stat_1" = x_stat_1,"x_stat_2" = x_stat_2,"x_stat_3" = x_stat_3,
    "x_stat_4" = x_stat_4, "x_stat_5" = x_stat_5,
                 "mu0" = 0, "g0" = 0.01, "mu1" = 0, "g1" = 0.01,
                 "mu2" = 0, "g2" = 0.01, "mu3" = 0, "g3" = 0.01,
                 "mu4" = 0, "g4" = 0.01, "mu5" = 0, "g5" = 0.01,
                 "mu6" = 0, "g6" = 0.01, "N" = N_wrk)


  posterior_wrk <- run.jags(modelString_wrk,
                    n.chains = 1,
                    data = the_data_wrk,
                    monitor = c("beta0", "beta1", "beta2",
                                "beta3", "beta4", "beta5",
                                "beta6"),
                      adapt = 1000,
                      burnin = 5000,
                      sample = 5000,
                      thin = 1,
                      inits = initsfunction)

post_wrk <- as.mcmc(posterior_wrk)

params<-data.frame(x_lim, x_stat_1, x_stat_2, x_stat_3, x_stat_4, x_stat_5)
n <- dim(NHISdata)[1]
syn_wrk <- synthesize_wrk(params, 1, n, post_wrk)
names(syn_wrk)=c("synthesized_lost_wrk")
return(syn_wrk)
}
```

```r
syn_lim<-synthesize_lim_func()
```

```
## Calling the simulation...
## Welcome to JAGS 4.3.0 on Tue Mar 31 12:54:20 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: 1000
##    Unobserved stochastic nodes: 1
##    Total graph size: 1005
##
## WARNING: Unused variable(s) in data table:
## mu0
##
## . Reading parameter file inits1.txt
## . Initializing model
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 5000
## -------------------------------------------------| 5000
## ************************************************** 100%
## . . Updating 5000
## -------------------------------------------------| 5000
## ************************************************** 100%
## . . . . Updating 0
## . Deleting model
## .
## Note: the model did not require adaptation
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Calculating summary statistics...

## Warning: Convergence cannot be assessed with only 1 chain

## Finished running the simulation
```

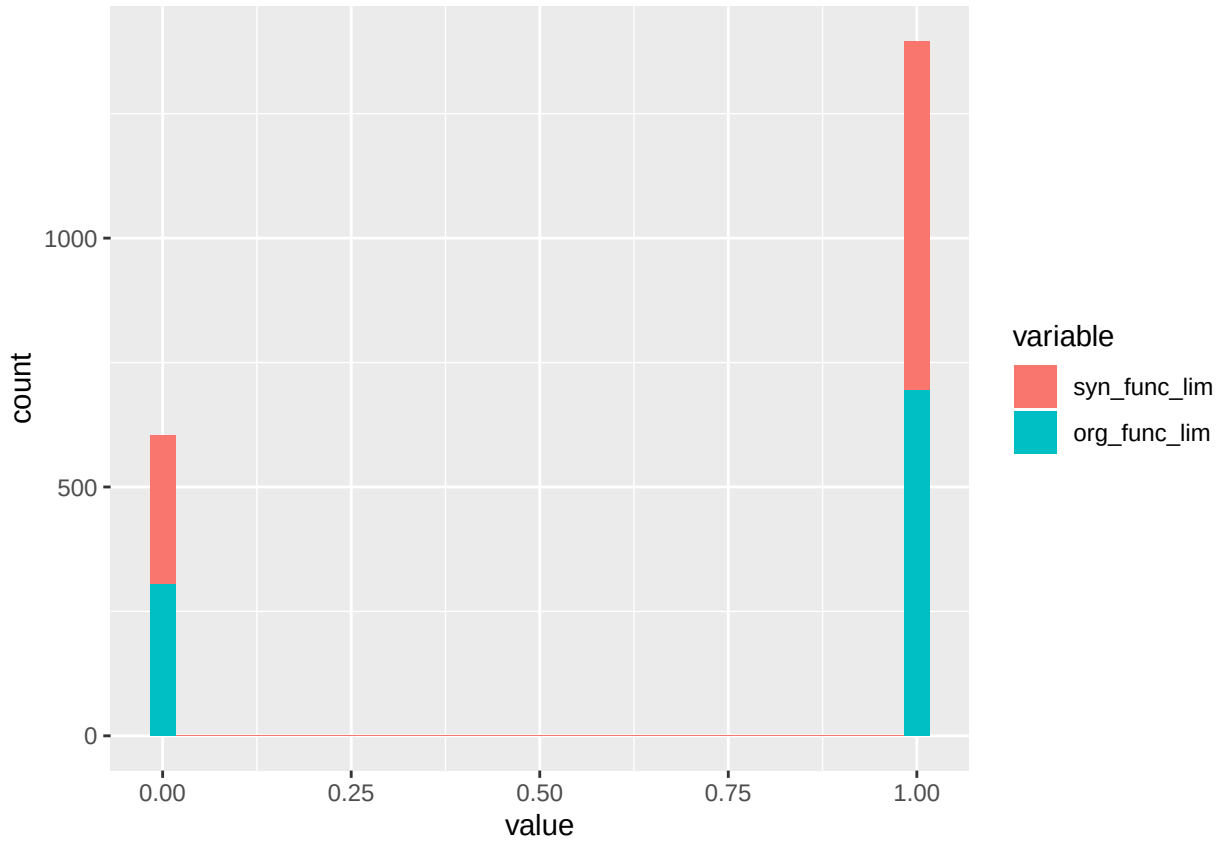
```r
SyntheticData_lim <- data.frame(syn_lim$synthesized_func_lim, NHISdata$FLA1AR-1)
names(SyntheticData_lim) = c("syn_func_lim", "org_func_lim")
```

```r
data<- melt(SyntheticData_lim)
```

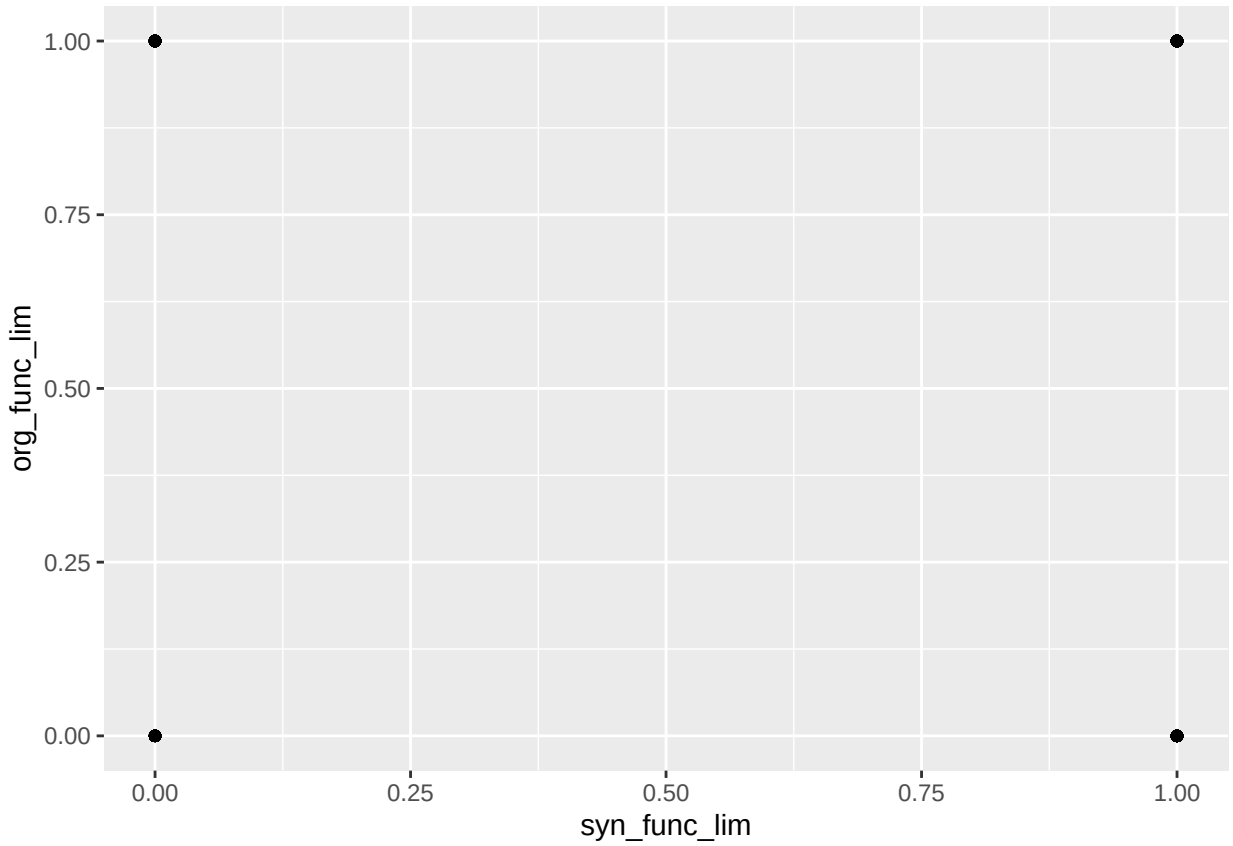
```
## Using  as id variables
```

```r
ggplot(data,aes(x=value, fill=variable)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(SyntheticData_lim, aes(x=syn_func_lim, y=org_func_lim)) +
  geom_point()
```




```
syn_stat<-synthesis_stat_func()
```

```
## Calling the simulation...
## Welcome to JAGS 4.3.0 on Tue Mar 31 12:54:25 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: 1000
##    Unobserved stochastic nodes: 1000
##    Total graph size: 2004
## . Reading parameter file inits1.txt
## . Initializing model
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 5000
## -------------------------------------------------| 5000
## ************************************************** 100%
## . . Updating 1000
## -------------------------------------------------| 1000
## ************************************************** 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
## Calculating summary statistics...

## Warning: Convergence cannot be assessed with only 1 chain
```

```
syn_stat_1<- syn_stat %>%
  mutate(syn_DOINGLWA=if_else(syn_emp_stat_1==1, 1,
                      if_else(syn_emp_stat_2==1, 2,
                      if_else(syn_emp_stat_3==1, 3,
                      if_else(syn_emp_stat_4==1, 4,
                      if_else(syn_emp_stat_5==1, 5, 9))))))
```

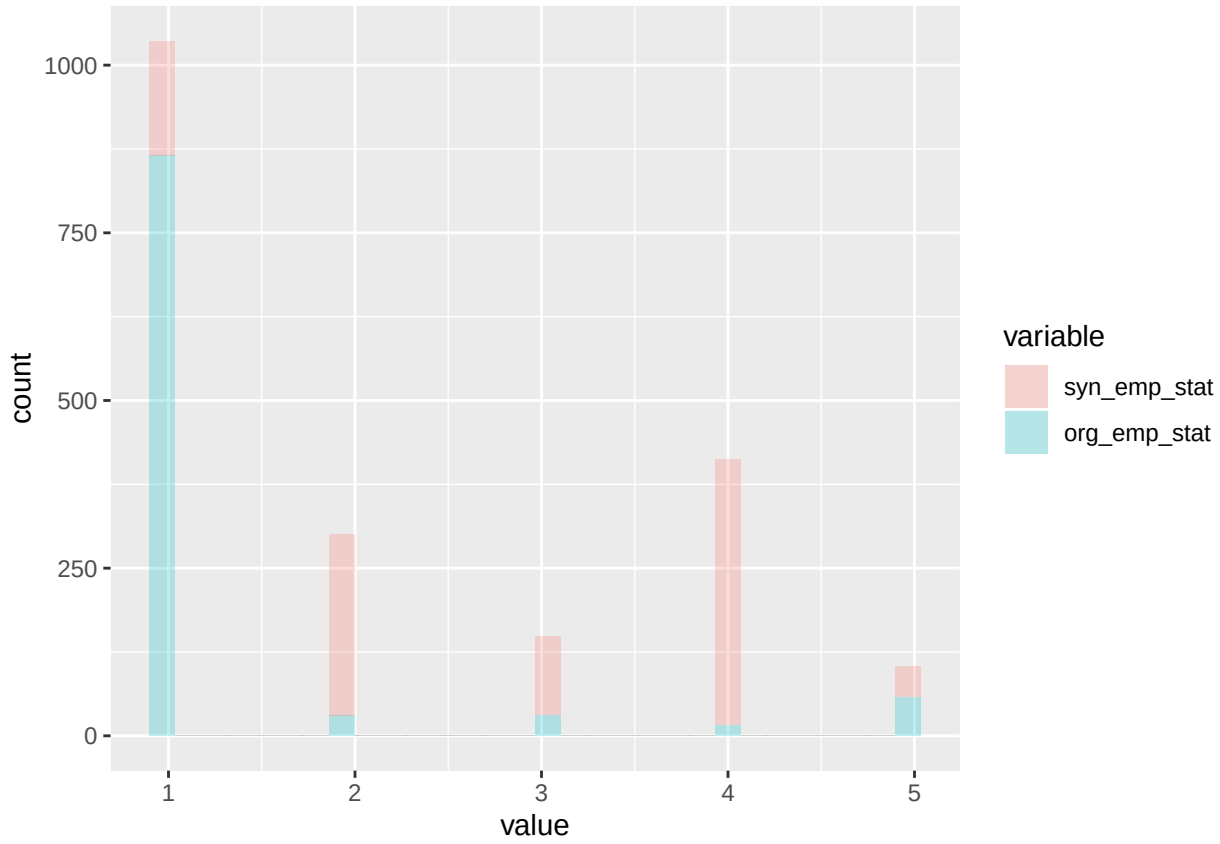
```
SyntheticData_stat <- data.frame(syn_stat_1$syn_DOINGLWA, NHISdata$DOINGLWA)
names(SyntheticData_stat) = c("syn_emp_stat", "org_emp_stat")
```

```
data<- melt(SyntheticData_stat)
```

```
## Using  as id variables
```

```
ggplot(data,aes(x=value, fill=variable)) + geom_histogram(alpha=0.25)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
org_stat<-NHISdata %>%
  select(Stat1, Stat2, Stat3, Stat4, Stat5)
```

```
syn_wrk<-synthesis_wrk_func(syn_lim, syn_stat)
```

```
## Calling the simulation...
## Welcome to JAGS 4.3.0 on Tue Mar 31 12:57:48 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: 1000
##    Unobserved stochastic nodes: 7
##    Total graph size: 7054
## . Reading parameter file inits1.txt
## . Initializing model
## . Adapting 1000
## -------------------------------------------------| 1000
## +++++++++++++++++++++++++++++++++++++++++++++++++ 100%
## Adaptation successful
## . Updating 5000
## -------------------------------------------------| 5000
```

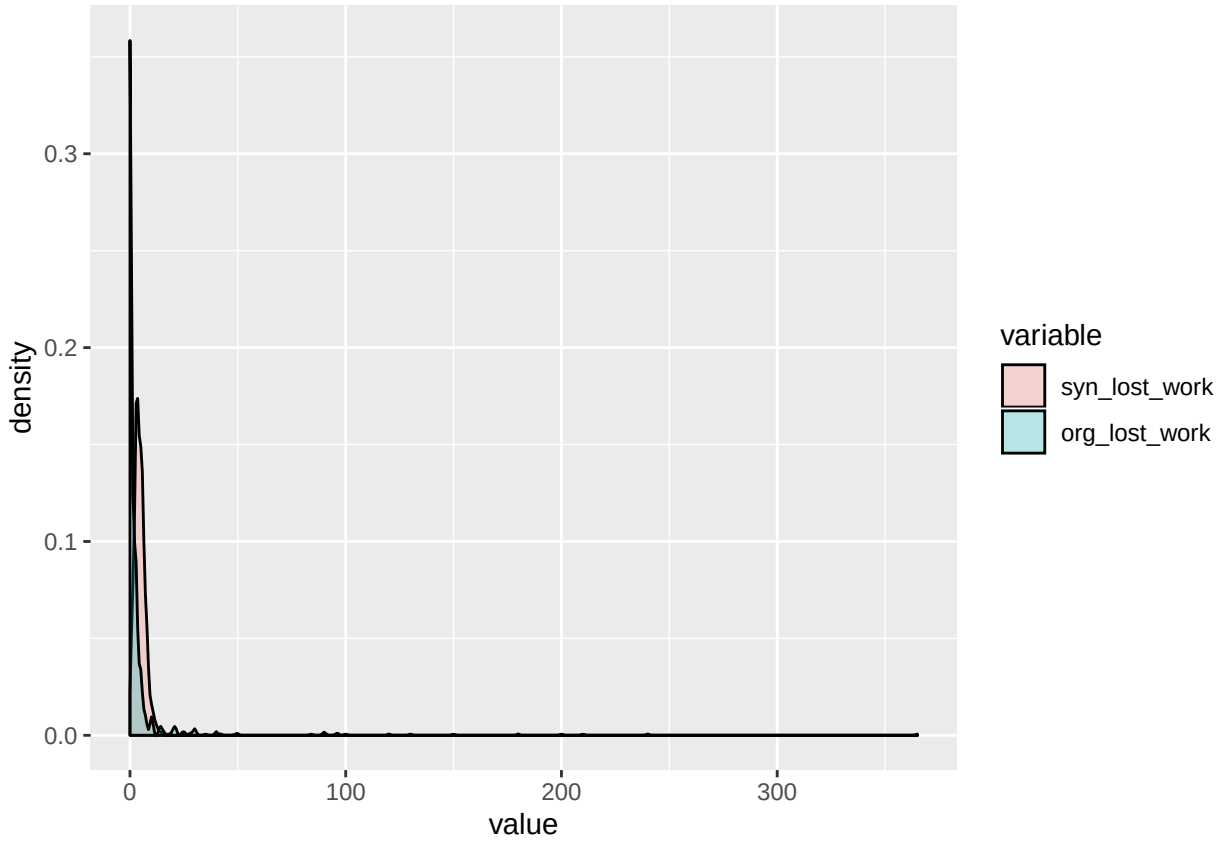
```
## ************************************************** 100%
## . . . . . . . . Updating 5000
## ----------------------------------------------------| 5000
## ************************************************** 100%
## . . . . Updating 0
## . Deleting model
## .
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Calculating summary statistics...

## Warning: Convergence cannot be assessed with only 1 chain

## Finished running the simulation
```

```
SyntheticData_wrk <- data.frame(syn_wrk$synthesized_lost_wrk, NHISdata$WKDAYR)
names(SyntheticData_wrk) = c("syn_lost_work", "org_lost_work")
```

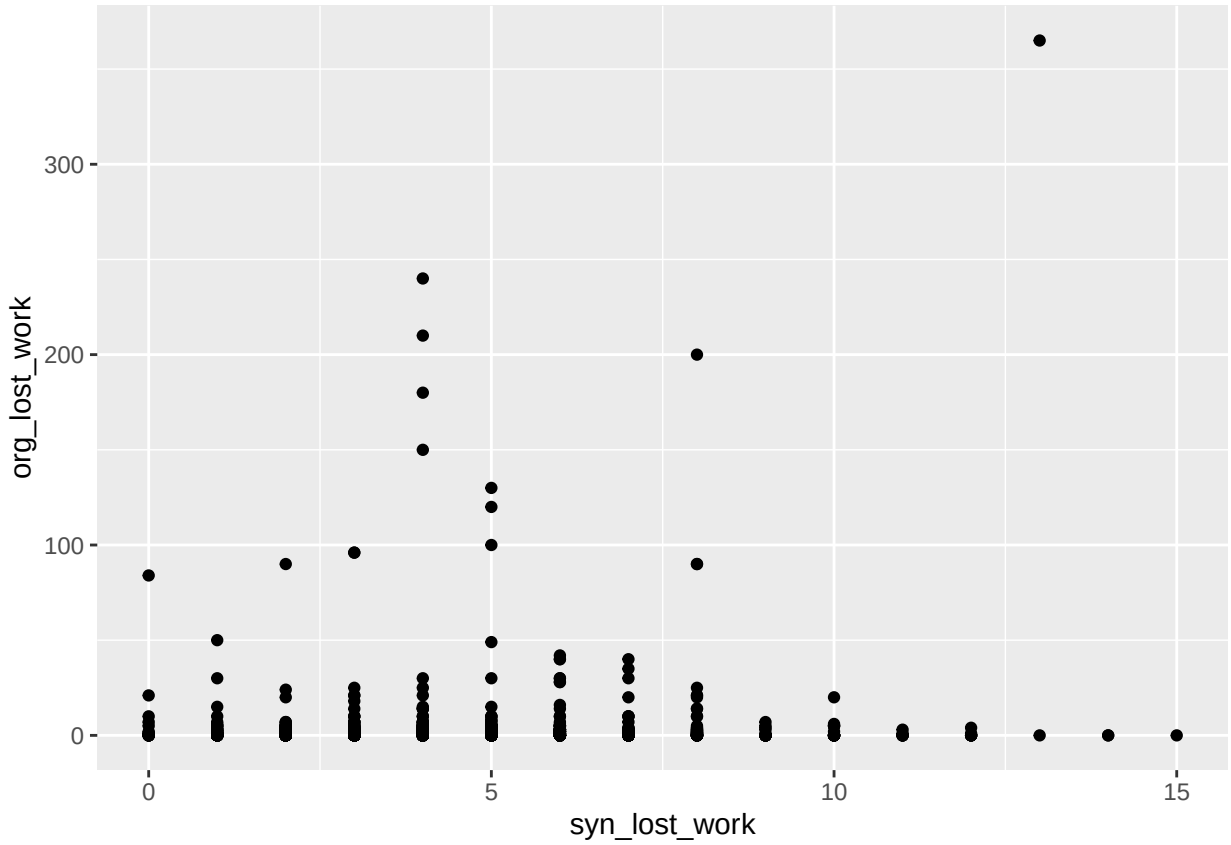
```
data<- melt(SyntheticData_wrk)
```

```
## Using  as id variables
```

```
ggplot(data,aes(x=value, fill=variable)) + geom_density(alpha=0.25)
```



```r
ggplot(SyntheticData_wrk, aes(x=syn_lost_work, y=org_lost_work)) +
  geom_point()
```



##Utility Measures:

I will begin by implimenting some of the universal utility measures discussed in Woo et. all. I begin with calculating the propensity score for each set for variables FLA1AR, DOINGLWA and WKDAYR.

##Propensity Scores

```r
n<-nrow(NHISdata)

create_T<- function(org,syn,n, variable){
  original_T<-data.frame(org, integer(length=n))
  names(original_T)= c(variable, "T")

  synthetic_T<-data.frame(syn, integer(length=n)+1)
  names(synthetic_T)= c(variable, "T")
  merged_T<- bind_rows(original_T, synthetic_T)
}

merged_data_lim<-create_T(SyntheticData_lim$org_func_lim, SyntheticData_lim$syn_func_lim, n, "FLA1AR")

merged_data_stat <- create_T(SyntheticData_stat$org_emp_stat, SyntheticData_stat$syn_emp_stat, n, "DOIN

merged_data_wrk <- create_T(SyntheticData_wrk$org_lost_work, SyntheticData_wrk$syn_lost_work, n, "WKDAY
```

```
n<-nrow(NHISdata)

mylogit_lim <- glm(T ~ FLA1AR, data = merged_data_lim, family = "binomial")
coefs_lim<-coef(mylogit_lim)
merged_data_lim<- merged_data_lim %>%
  mutate(p_hat=1/(1+exp(-1*(coefs_lim[1]+coefs_lim[2]*FLA1AR)))) %>%
  mutate(p_hat_c2=(p_hat-.5)^2)
U_p_lim<-(1/(2*n))*sum(merged_data_lim$p_hat_c2)
U_p_lim
```

```
## [1] 1.067383e-05
```

```
mylogit_stat <- glm(T ~ DOINGLWA, data = merged_data_stat, family = "binomial")
coefs_stat<-coef(mylogit_stat)
merged_data_stat<- merged_data_stat %>%
  mutate(p_hat=1/(1+exp(-1*(coefs_stat[1]+coefs_stat[2]*DOINGLWA)))) %>%
  mutate(p_hat_c2=(p_hat-.5)^2)
U_p_stat<-(1/(2*n))*sum(merged_data_stat$p_hat_c2)
U_p_stat
```

```
## [1] 0.07700381
```

```
mylogit_wrk <- glm(T ~ WKDAYR, data = merged_data_wrk, family = "binomial")
coefs_wrk<-coef(mylogit_wrk)
merged_data_wrk<- merged_data_wrk %>%
  mutate(p_hat=1/(1+exp(-1*(coefs_wrk[1]+coefs_wrk[2]*WKDAYR)))) %>%
  mutate(p_hat_c2=(p_hat-.5)^2)
U_p_wrk<-(1/(2*n))*sum(merged_data_wrk$p_hat_c2)
U_p_wrk
```

```
## [1] 5.737965e-07
```

This is Prof. Hu's method for calculating the propensity score for the variable FLA1AR. These methods return values that are close together.

```
log_reg<-glm(T ~ FLA1AR, data = merged_data_lim, family = "binomial")
pred <- predict(log_reg, data = merged_data_lim)
probs <- exp(pred)/(1+exp(pred))
Up <- 1/(2*n)*sum((probs - 1/2)^2)
```

##Function using clustering algorithim to calculate cluster analysis utility measure.

```
calc_Uc<-function(merged_data){
  clusters <- hclust(dist(merged_data[, 1:2]), method = 'average')
  G <- 5
  clusterCut <- cutree(clusters, G)
  cluster_S <- as.data.frame(cbind(clusterCut, merged_data$T))
  names(cluster_S) <- c("cluster", "S")
  table(cluster_S)
```

```
  n_gS <- table(cluster_S)[, 1]
  n_g <- rowSums(table(cluster_S))
  w_g <- n_g / (2*n)

  Uc <- (1/G) * sum(w_g * (n_gS/n_g - 1/2)^2)
  return(Uc)
}
```

##Calculating cluster analysis measure Uc

```
Uc_lim<-calc_Uc(merged_data_lim)
Uc_lim
```

```
## [1] 0.05
```

```
Uc_stat<-calc_Uc(merged_data_stat)
Uc_stat
```

```
## [1] 0.04754932
```

```
Uc_wrk<-calc_Uc(merged_data_wrk)
Uc_wrk
```

```
## [1] 0.0003778338
```

```
calc_cdf<-function(org, syn){
  ecdf_orig <- ecdf(org)
  ecdf_syn <- ecdf(syn)

 percentile_orig <- ecdf_orig(org)
 percentile_syn <- ecdf_syn(syn)

  ecdf_diff <- percentile_orig - percentile_syn
  Um <- max(abs(ecdf_diff))
  Ua <- mean(ecdf_diff^2)
  emp_cdf<-data.frame(Um,Ua)
  names(emp_cdf)=c("Um", "Ua")
  return(c(Um,Ua))
}
```

```
cdf_lim<-calc_cdf(SyntheticData_lim$org_func_lim, SyntheticData_lim$syn_func_lim)
cdf_stat<-calc_cdf(SyntheticData_stat$org_emp_stat, SyntheticData_stat$syn_emp_stat)
cdf_wrk<-calc_cdf(SyntheticData_wrk$org_lost_work, SyntheticData_wrk$syn_lost_work)
```

```
CalculateKeyQuantities <- function(origdata, syndata, known.vars, syn.vars, n){
  origdata <- origdata
  syndata <- syndata
  n <- n
  c_vector <- rep(NA, n)
  T_vector <- rep(NA, n)
```

```
  for (i in 1:n){
    match <- (eval(parse(text=paste("origdata$",syn.vars,"[i]==
      syndata$",syn.vars,sep="",collapse="&")))&
      eval(parse(text=paste("origdata$",known.vars,"[i]==
      syndata$",known.vars,sep="",collapse="&"))))
    match.prob <- ifelse(match, 1/sum(match), 0)
    if (max(match.prob) > 0){
      c_vector[i] <- length(match.prob[match.prob == max(match.prob)])
    } else {
    c_vector[i] <- 0
    T_vector[i] <- is.element(i, rownames(origdata)[match.prob == max(match.prob)])
    }
  }

  K_vector <- (c_vector * T_vector == 1)
  F_vector <- (c_vector * (1 - T_vector) == 1)
  s <- length(c_vector[c_vector == 1 & is.na(c_vector) == FALSE])
  res_r <- list(c_vector = c_vector, T_vector = T_vector, K_vector = K_vector,
                known.vars, F_vector = F_vector,s=s)
  return(res_r)
}
```

```
known.vars <- c("SEX", "RACE", "MAR")
syn.vars <- c("LANX", "WAOB", "DIS", "HICOV") n <- dim(ACSdata_org)[1]
KeyQuantities <- CalculateKeyQuantities(ACSdata_org, ACSdata_syn, known.vars, syn.vars, n)
```

```
IdentificationRisk <- function(c_vector, T_vector, K_vector, F_vector, s, N){ nonzero_c_index <- which(
exp_match_risk <- sum(1/c_vector[nonzero_c_index]*T_vector[nonzero_c_index]) true_match_rate <- sum(na.
false_match_rate <- sum(na.omit(F_vector))/s
res_r <- list(exp_match_risk = exp_match_risk, true_match_rate = true_match_rate,
)
return(res_r) }
```