```{r}
#remove variables
rm(list=ls())
#import needed libraries
library("bayesm")
library(tidyverse)
library(tidyr)
library(dplyr)
library(nnet)
#Load in the Data
data(margarine)
#create dataframes
df=as.data.frame(margarine$choicePrice)
df2=as.data.frame(margarine$demos)
df3=left_join(df, df2,by="hhid")

```

```
#Q1 Means
colMeans(df)
#Write SD function by column
colSd <- function (x, na.rm=FALSE) apply(X=x, MARGIN=2, FUN=sd, na.rm=na.rm)
#Column standard deviation
colSd(df)
#Groupby count of choices and get marketshare
temp=count(df,"hhid")[1,'n']
a=df %>% group_by(choice) %>% count("hhid")
a['n']=a['n']/temp
a
#to determine whether a person bought a product that was above the mean price for their options, create the average price per row and create an indicator for if the selected choice has a price that is above that average. Then create a groupby for both above and below
paramnames=colnames(df)[3:12]
#calculate row's mean
temp=rowMeans(df[,paramnames])
#create a variable for the price of the selected good
temp2=mat.or.vec(nrow(df),1)
for (i in 1:nrow(df))
{
  temp2[i]=df[i,paramnames[df[i,'choice']]]
}
df['price']=temp2
#generate a variable if above the mean
df['abovemean']=as.integer(df['price']>temp)
#create our groupby
a=df %>% group_by(choice, abovemean) %>% count("hhid")
#marketshare for above mean price
a2=a[a['abovemean']==1,]
a2['n']=a2['n']/sum(a2['n'])
a2
#market share for below mean prices
a3=a[a['abovemean']==0,]
a3['n']=a3['n']/sum(a3['n'])
a3
# Descriptives on which personal characteristics tend to be higher for each choice on average
aggregate(df3[,colnames(df2)[2:8]],list(df3$choice),mean)
```

```
        hhid       choice      PPk_Stk      PBB_Stk
2.125756e+06 3.242953e+00 5.184362e-01 5.432103e-01
     PFl_Stk     PHse_Stk     PGen_Stk     PImp_Stk
1.015020e+00 4.371477e-01 3.452819e-01 7.807785e-01
     PSS_Tub      PPk_Tub      PFl_Tub     PHse_Tub
8.250895e-01 1.077409e+00 1.189376e+00 5.686734e-01
        hhid       choice      PPk_Stk      PBB_Stk
1.426333e+04 2.587219e+00 1.505174e-01 1.203319e-01
     PFl_Stk     PHse_Stk     PGen_Stk     PImp_Stk
4.289519e-02 1.188312e-01 3.516605e-02 1.146461e-01
     PSS_Tub      PPk_Tub      PFl_Tub     PHse_Tub
6.121159e-02 2.972613e-02 1.405451e-02 7.245500e-02
```

choice <dbl>	"hhid" <chr>	n <dbl>
1	hhid	0.39507830
2	hhid	0.15637584
3	hhid	0.05436242
4	hhid	0.13266219
5	hhid	0.07046980
6	hhid	0.01655481
7	hhid	0.07136465
8	hhid	0.04541387
9	hhid	0.05033557
10	hhid	0.00738255

choice <dbl>	abovemean <int>	"hhid" <chr>	n <dbl>
3	1	hhid	0.25471698
6	1	hhid	0.04402516
7	1	hhid	0.25262055
8	1	hhid	0.21278826
9	1	hhid	0.23584906

choice <dbl>	abovemean <int>	"hhid" <chr>	n <dbl>
1	0	hhid	0.502275313
2	0	hhid	0.198805461
4	0	hhid	0.168657565
5	0	hhid	0.089590444
6	0	hhid	0.009101251
7	0	hhid	0.022184300
10	0	hhid	0.009385666

Group.1 <dbl>	Income <dbl>	Fs3_4 <dbl>	Fs5. <dbl>	Fam_Size <dbl>	college <dbl>	whtcollar <dbl>	retired <dbl>
1	26.71291	0.5107588	0.13703284	3.174972	0.3176670	0.5702152	0.19932050
2	26.06581	0.5150215	0.11158798	3.101574	0.3133047	0.5436338	0.24034335
3	30.70988	0.2551440	0.08230453	2.481481	0.4526749	0.5432099	0.53086420
4	27.64334	0.5025295	0.19898820	3.470489	0.2934233	0.5919056	0.15345700
5	26.44444	0.5936508	0.20000000	3.692063	0.2730159	0.7142857	0.14603175
6	39.15541	0.2432432	0.31081081	3.175676	0.4324324	0.5675676	0.37837838
7	25.32132	0.4921630	0.06269592	2.890282	0.3228840	0.5768025	0.14733542
8	34.24877	0.6009852	0.05418719	3.093596	0.2561576	0.5714286	0.09852217
9	31.90000	0.3022222	0.04888889	2.386667	0.2755556	0.5777778	0.36000000
10	29.46970	0.3636364	0.54545455	4.424242	0.4545455	0.9393939	0.12121212

```{r}
#Set random seed
set.seed(42)
#create a list of our parameter names
paramnames=colnames(df)[3:12]
#get the number of parameters, choices, and observations
numin=length(paramnames)
numout=n_distinct(df['choice'])
numobs=nrow(df)
#create a random start of size number of choices (10 intercepts) plus one more for price
start=runif(numout+1,-10,10)
#create a matrix of equal size to the observations and number of choices
ut = mat.or.vec(numobs,numout)
#create our likelihood function
flike=function(param,df){
  #loop for each choice
for (j in 1:numout)
{
  #replace in our empty matrix with the intercept for that choice
  #plus the beta for price times the price of the choice
  ut[,j]=param[j]+param[numout+1]*df[,paramnames[j]]
}
  #calculate the probabilities
prob   = exp(ut)            # exp(XB)
prob   = sweep(prob,MARGIN=1,FUN="/",STATS=rowSums(prob))
#select the probability in question
probc = NULL
for (i in 1:numobs)
  {
    probc[i] = prob[i,df[i,'choice']]
  }
  probc[probc>0.999999] = 0.999999
  probc[probc<0.000001] = 0.000001
  like = sum(log(probc))
  return(-like)
}
#run the optimization
res      = optim(start,fn=flike,method="BFGS",control=list(trace=6,maxit=1000),df=df)
#parameter on beta. All this states is (assuming it is significant) that increasing price, on average decreases the probability a particular good is purchased
res$par[11]
```

```
initial  value 15661.250327
iter  10 value 10993.101397
iter  20 value 9462.561007
final  value 9462.556704
converged
[1] -6.776574
```

```{r}
#take the output parameters
param=res$par
#calculate our counts and matrix
numin=length(paramnames)
numout=n_distinct(df['choice'])
numobs=nrow(df)
ut = mat.or.vec(numobs,numout)
#rerun our probabilities from before
for (j in 1:numout)
{
  ut[,j]=param[j]+param[numout+1]*df[,paramnames[j]]
}
prob   = exp(ut)            # exp(XB)
prob   = sweep(prob,MARGIN=1,FUN="/",STATS=rowSums(prob))

#create an empty matrix
paramc = mat.or.vec(numobs,numout)
#for each i and j calculate the effect of beta using the marginal effect formula
for (i in 1:numobs)
  {
    #find the selected choice
    temp=df[i,'choice']
    for (j in 1:numout)
    {
      if (temp==j)
      {
       paramc[i,j]=prob[i,j]*(1-prob[i,j])*param[11]
      }
      if (temp!=j)
      {
       paramc[i,j]=prob[i,j]*(-prob[i,j])*param[11]
      }
    }

}
#Marginal effects of increasing the price of a good on each of the 10 products. These tell us how an increase in price affects the probability that any one choice is chosen
  colMeans(paramc)
```

```
  [1]  3.975512e-02 -1.185805e-02  6.140651e-03 -3.487977e-04
  [5]  5.091597e-04  5.941763e-04  3.269976e-03 -2.808594e-08
  [9]  6.066944e-03  2.540738e-05
```

```{r}
#same as before
set.seed(42)
numout=n_distinct(df3['choice'])
numobs=nrow(df3)
start=runif(20,-1,1)
ut = mat.or.vec(numobs,numout)


flike=function(param,df3){
  #create 20 parameters, 10 constants and 10 betas (1 for each choice)
const=param[1:10]
pn1=param[11:20]
#loop through choices
for (j in 1:10)
{
  #add income times the beta parameter plus the corresponding constant
  ut[,j]=const[j]+pn1[j]*df3[,'Income']
}
#same as before
prob   = exp(ut)            # exp(XB)
prob   = sweep(prob,MARGIN=1,FUN="/",STATS=rowSums(prob))
probc = NULL
for (i in 1:numobs)
  {
    probc[i] = prob[i,df3[i,'choice']]
  }
  probc[probc>0.999999] = 0.999999
  probc[probc<0.000001] = 0.000001
  like = sum(log(probc))
  return(-like)
}
#same as before on optimization
res      = optim(start,fn=flike,method="BFGS",control=list(trace=6,maxit=1000),df3=df3)
#intercepts
res$par[1:10]
#coefficients on income. It says that income increases the probability of choosing certain products and decreases the probability of choosing other products (because there are 10, 1 one would be put into a main effect and the others would be incremental)
res$par[11:20]
```

```
initial  value 50530.937087
iter  10 value 36620.308644
iter  20 value 30106.811584
iter  30 value 27733.917991
iter  40 value 23149.870919
iter  50 value 21426.034178
iter  60 value 19094.875577
iter  70 value 17290.130962
iter  80 value 14919.807053
iter  90 value 10764.322604
iter 100 value 8852.799125
iter 110 value 8239.652783
iter 120 value 8237.833823
iter 130 value 8237.202784
iter 130 value 8237.202784
iter 140 value 8236.926763
iter 150 value 8236.797333
iter 150 value 8236.797257
final  value 8236.795320
converged
 [1]  2.43787151  1.57939274  0.02728317  1.21712296  0.73670850 -1.71198331  0.88572096 -0.41910310 -0.14442337
[10] -1.89289166
 [1] -1.102527 -1.105235 -1.087648 -1.097955 -1.103477 -1.071646 -1.108831 -1.079395 -1.084582 -1.090429
```

```{r}
set.seed(42)
numout=n_distinct(df3['choice'])
numobs=nrow(df3)
ut = mat.or.vec(numobs,numout)
param=res$par
const=param[1:10]
pn1=param[11:20]

for (j in 1:10)
{
  ut[,j]=const[j]+pn1[j]*df3[,'Income']
}
prob   = exp(ut)             # exp(XB)
prob   = sweep(prob,MARGIN=1,FUN="/",STATS=rowSums(prob))
probc = NULL
probc2=mat.or.vec(numobs,numout)
probc3=mat.or.vec(numobs,numout)
for (i in 1:numobs)
  {
    probc[i] = prob[i,df3[i,'choice']]
    probc2[i,]=prob[i,]*(const-sum(prob[i,]*const))
    probc3[i,]=prob[i,]*(pn1-sum(prob[i,]*pn1))
  }
  probc[probc>0.999999] = 0.999999
  probc[probc<0.000001] = 0.000001
  like = sum(log(probc))
  #marginal effects for intercepts
colMeans(probc2)
#marginal effects for coefficients. Increases in family income are associated with an increase in probability to pick some products and a decrease in probability to pick others
colMeans(probc3)
```

```
 [1]  0.39931371  0.02281071 -0.07442143 -0.02702555 -0.04889397 -0.05012973 -0.03932733 -0.08105004 -0.07713263
[10] -0.02414372
 [1] -1.165210e-03 -8.762562e-04  6.306713e-04  2.047898e-04 -2.731342e-04  4.437968e-04 -6.514827e-04  8.876515e-04
 [9]  7.339732e-04  6.520093e-05
```

```
set.seed(42)
#same as in conditional logit, but will need more parameters
paramnames=colnames(df)[3:12]
numout=n_distinct(df3['choice'])
numobs=nrow(df3)
start=runif(21,-1,1)
ut = mat.or.vec(numobs,numout)


flike=function(param,df3){
  #same parameters for family income
const=param[1:10]
pn1=param[11:20]
# loop over 10
for (j in 1:10)
{
  # combine the two models previously
  ut[,j]=const[j]+pn1[j]*df3[,'Income']+param[2*numout+1]*df3[,paramnames[j]]
}
#rest of the calculation is identical
prob   = exp(ut)            # exp(XB)
prob   = sweep(prob,MARGIN=1,FUN="/",STATS=rowSums(prob))
probc = NULL
for (i in 1:numobs)
  {
    probc[i] = prob[i,df3[i,'choice']]
  }
  probc[probc>0.999999] = 0.999999
  probc[probc<0.000001] = 0.000001
  like = sum(log(probc))
  return(-like)
}
#use the same optimization
res      = optim(start,fn=flike,method="BFGS",control=list(trace=6,maxit=1000),df3=df3)
#calculate the likelihood of the model
ll=res$value
```

```
initial  value 50744.677775
iter  10 value 34684.542490
iter  20 value 28646.918906
iter  30 value 20003.389121
iter  40 value 14399.424415
iter  50 value 9932.982790
iter  60 value 7918.721031
iter  70 value 7512.101083
iter  80 value 7424.850325
iter  90 value 7422.026063
iter 100 value 7420.249166
iter 110 value 7418.841031
iter 110 value 7418.841031
iter 110 value 7418.841031
final  value 7418.841031
converged
```

```r
#drop choice 10
set.seed(42)
df4=df3[df3['choice']!=10,]
#adjust parameters for the lost choice
paramnames=colnames(df)[3:11]
numout=n_distinct(df4['choice'])
numobs=nrow(df4)
#lower number of starting parameters for the missing choices
start=runif(19,-1,1)
ut = mat.or.vec(numobs,numout)
#define function
flike=function(param,df4){
  #similar to before, but now with 9 choices
const=param[1:9]
pn1=param[10:18]
#only iterate over 9 choices
for (j in 1:9)
{
  ut[,j]=const[j]+pn1[j]*df4[,'Income']+param[2*numout+1]*df4[,paramnames[j]]
}
#same calculation as before
prob   = exp(ut)            # exp(XB)
prob   = sweep(prob,MARGIN=1,FUN="/",STATS=rowSums(prob))
probc = NULL
for (i in 1:numobs)
  {
    probc[i] = prob[i,df4[i,'choice']]
  }
  probc[probc>0.999999] = 0.999999
  probc[probc<0.000001] = 0.000001
  like = sum(log(probc))
  return(-like)
}
#same optimization
res      = optim(start,fn=flike,method="BFGS",control=list(trace=6,maxit=1000),df4=df4)
l2=res$value
#create our test statistic for IIA
-2*(l1-l2)
#Clearly this is significant on a chi-squared distribution, and thus we can conclude that we violate IIA
```

```
initial  value 41363.403009
iter  10 value 22941.859340
iter  20 value 16390.832554
iter  30 value 13493.714470
iter  40 value 12517.442201
iter  50 value 11674.877375
iter  60 value 9600.114519
iter  70 value 9256.000299
iter  80 value 8912.440862
iter  90 value 7351.943421
iter 100 value 7240.718878
iter 100 value 7240.718813
iter 110 value 7240.279715
final  value 7240.279483
converged
[1] -357.1231
```