```
- Conjunct Analysis and Market Simulator
```

In [ ]:

```
filenm = "ClassExampleConjointData"
load(paste(filenm,".Rdata",sep=""))

##list the objects and make sense of them
ls()
atts = c("Low Price","Tall Size","Rocking","Glamour")
colnames(desmat) = atts

####Solution to analysis
##regressions
##aggregate
summary(lm(ratings~desmat))

##by apriori segment age
summary(lm(ratings~desmat*ageD))
summary(lm(ratings~desmat*genderD)); ##run the regression with interactions for segment dummies
##note if significant. can run separately for two categories
summary(lm(ratings~desmat,subset=ageD==1)) # older kids
summary(lm(ratings~desmat,subset=ageD==0)) # young kids

##by individual
desmatf = cbind(rep(1,nrow(desmat)),desmat); ##add column for constant
partworths = matrix(nrow=sampsize,ncol=ncol(desmatf))
for(i in 1:sampsize){ #for each individual run the regression
  partworths[i,]=lm(ratings~desmat,subset=ID==i)$coef
}
colnames(partworths) = c("Intercept",atts)

##segmenting individuals
toClust = partworths;
##use code from last week to do the post-hoc segmentation


##NEED TO PREDICT MISSING CELLS
##predict missing cells (preparing for market simulation)
##repeat individual level partworths for multiplication
partworths.full = matrix(rep(partworths,each=16),ncol=5)
pratings = rowSums(desmatf*partworths.full)
finalratings = ifelse(is.na(ratings),pratings,ratings); #combining actual when available and predicted ratings
```

```
##market simulation
##Define scenarios!
  ##a scenario is a set of products, each with a set of levels.
  ## create a vector with the indexes for the product profiles from the

##Generate profits given market simulation

## prices are the prices for all products, vcosts are the variable costs
## fcosts are the fixed costs for the firm (need to calculate in already the number of products)
simProfit = function(inputmat,scen, myProds, prices, vcosts,fcosts,mktsize=1){
  mktshr = simFCShare(inputmat,scen);
  vprofit = mktshr * (prices-vcosts)*mktsize;
  sum(vprofit[myProds])-fcosts
}
simProf0 = simProfit(simDecInput,scen0,c(2),c(139,139),c(99,99),20000,2000)
simProf1 = simProfit(simDecInput,scen1,c(2,3),c(139,139,139),c(89,89,99),40000,2000)
simProf2 = simProfit(simDecInput,scen2,c(2,3),c(119,139,139),c(89,89,99),40000,2000)


##For Designing Tasks to Present to Respondents
##THIS SECTION NOT REQUIRED FOR ASSIGNMENT
install.packages("AlgDesign"); #install package AlgDesign if not already installed
library(AlgDesign); #load the library AlgDesign

set.seed(123)
levels.design= c(2,2,3,4); #set the number of levels for each variable
f.design <- gen.factorial(levels.design,factors="all"); #construct full orthogonal design
##Note: the argument factor="all" indicates that all of the variables are factors (i.e., categorical variables)
fract.design <- optFederov(frml=~X1+X2+X3+X4,data=f.design,nTrials=12,approximate=FALSE);

simScenarios = function(scenarios,data){
}

simScenarios = function(scenarios,data)

simScenarios = function(scenarios,data,...){
    res = matrix(nrow=length(scenarios),ncol=length(data)) #sets everything to NA by default
    for(i in 1:length(scenarios)){ ##loop over scenarios
    res[i, scenarios[[i]] ] = simFCShares(scenarios[[i]],data,...)
    res = as.data.frame(res); names(res) = names(data) #setting type and names
    }
res = as.data.frame(res); names(res) = names(data)
simFCShares = function(scen,data)
rep(1/length(scen),length(scen)) #dummied result where all shares are equal in scenario to allow simScenarios to run
}
#test the dummied worker function
simFCShares(scens[[1]],data[,2:6])
## [1] 0.3333333 0.3333333 0.3333333
#test the top-level function using the dummied simFCShares
simScenarios(scens,data[,2:6])
## Kolander.s.Regular Fisherman.s.Delight Kolander.s.Creamy Cape.Cod
## 1 NA 0.3333333 0.3333333 0.3333333
## 2 NA 0.3333333 NA 0.3333333
## 3 NA 0.2500000 0.2500000 0.2500000
## 4 0.2500000 0.2500000 0.2500000 0.2500000
## 5 0.2500000 0.2500000 NA 0.2500000
## 6 0.2000000 0.2000000 0.2000000 0.2000000
## 7 0.3333333 0.3333333 NA 0.3333333
## Kolander.s.Extra.Creamy
## 1 NA
## 2 0.3333333
## 3 0.2500000
## 4 NA
## 5 0.2500000
## 6 0.2000000
## 7 NA

##Arguments:
## scen indicates which columns of data are included in the scenario
## data is full set of data
##Returns:
## vector of shares with same length as data containing the shares
simFCShares = function(scen,data){
inmkt = data[,scen] #construct the subsetted matrix of options
bestOpts = apply(inmkt,1,which.min) #identify which option is best = min value
decs = as.data.frame(model.matrix(~0+as.factor(bestOpts))) #fill decisions to be 0 or 1 for all products
shs = colSums(decs)/sum(decs) #assumes that total decisions is market size
names(shs) = names(inmkt) #attach labels
shs
}
##Test the code
#test the full worker function
simFCShares(scens[[1]],data[,2:6])
## Fisherman.s.Delight Kolander.s.Creamy Cape.Cod
## 0.40 0.26 0.34
#test the top-level function using the full simFCShares
```

```
simScenarios(scens,data[,2:6])
## Kolander.s.Regular Fisherman.s.Delight Kolander.s.Creamy Cape.Cod
## 1 NA 0.40 0.260 0.340
## 2 NA 0.55 NA 0.200
## 3 NA 0.40 0.255 0.095
## 4 0.295 0.11 0.255 0.340
## 5 0.290 0.26 NA 0.200
## 6 0.290 0.11 0.255 0.095
## 7 0.295 0.26 NA 0.445
## Kolander.s.Extra.Creamy
## 1 NA
## 2 0.25
## 3 0.25
## 4 NA
## 5 0.25
## 6 0.25
## 7 NA

simFCShares = function(scen,data,bestValueIsLow=TRUE){
if(bestValueIsLow==FALSE) { #best value is high
data = -data #make values opposite sign so e.g,. 5 become -5 and now finding the min still works.
}
inmkt = data[,scen] #construct the subsetted matrix of options
bestOpts = apply(inmkt,1,which.min) #identify which option is best = min
decs = as.data.frame(model.matrix(~0+as.factor(bestOpts))) #fill to set of options marked 0 or 1
shs = colSums(decs)/sum(decs) #assumes that total decisions is market size
names(shs) = names(inmkt) #attach labels
shs
}
#test without passing the argument
simFCShares(scens[[1]],data[,2:6])
## Fisherman.s.Delight Kolander.s.Creamy Cape.Cod
## 0.40 0.26 0.34
#test bestValueIsLow=TRUE still works same
simFCShares(scens[[1]],data[,2:6],bestValueIsLow=TRUE)
3
## Fisherman.s.Delight Kolander.s.Creamy Cape.Cod
## 0.40 0.26 0.34
#test bestValueIsLow=FALSE still works same
simFCShares(scens[[1]],data[,2:6],bestValueIsLow=FALSE)
## Fisherman.s.Delight Kolander.s.Creamy Cape.Cod
## 0.44 0.01 0.55
#test without passing the argument
```

```
simScenarios(scens,data[,2:6])
## Kolander.s.Regular Fisherman.s.Delight Kolander.s.Creamy Cape.Cod
## 1 NA 0.40 0.260 0.340
## 2 NA 0.55 NA 0.200
## 3 NA 0.40 0.255 0.095
## 4 0.295 0.11 0.255 0.340
## 5 0.290 0.26 NA 0.200
## 6 0.290 0.11 0.255 0.095
## 7 0.295 0.26 NA 0.445
## Kolander.s.Extra.Creamy
## 1 NA
## 2 0.25
## 3 0.25
## 4 NA
## 5 0.25
## 6 0.25
## 7 NA
simScenarios(scens,data[,2:6],bestValueIsLow=TRUE)
## Kolander.s.Regular Fisherman.s.Delight Kolander.s.Creamy Cape.Cod
## 1 NA 0.40 0.260 0.340
## 2 NA 0.55 NA 0.200
## 3 NA 0.40 0.255 0.095
## 4 0.295 0.11 0.255 0.340
## 5 0.290 0.26 NA 0.200
## 6 0.290 0.11 0.255 0.095
## 7 0.295 0.26 NA 0.445
## Kolander.s.Extra.Creamy
## 1 NA
## 2 0.25
## 3 0.25
## 4 NA
## 5 0.25
## 6 0.25
## 7 NA

simScenarios(scens[-c(4,6)],data[,2:6],bestValueIsLow=FALSE)
## Kolander.s.Regular Fisherman.s.Delight Kolander.s.Creamy Cape.Cod
## 1 NA 0.440 0.010 0.55
## 2 NA 0.395 NA 0.01
```