Market Simulator

Mitchell J. Lovett

11/19/2020

We begin with the design of the top-level function for the software. We specify the output we want (or side-effects, but none in this case) as well as the arguments. This tells us what we need to accomplish.

```
##Arguments:
##  scenarios are the set of scenarios we are interested in evaluating
##  data is the data containing the rank order information
##Return:
##  data.frame containing shares with columns corresponding to the columns in the data
simScenarios = function(scenarios,data){
}
```

The next step is to get clearer about the algorithm in the top-level function. Here, we need to create the return table, then loop over scenarios that we pass. For each scenario (for loop!), we calculate the market shares and then insert that information into the row of the result table. After we complete all scenarios, we return the result table.

Notice that we have put off most of the work for a still to be defined step inside the loop! We will now write that step.

```
##Arguments:
##  scenarios are the set of scenarios we are interested in evaluating
##  data is the data containing the rank order information
##Return:
##  data.frame containing shares with columns corresponding to the columns in the data
simScenarios = function(scenarios,data){
  ##create result table
  ##loop over scenarios
  ##  calculate market shares and save into result table in appropriate indexes for the scenario
  ##return result table
}
```

Now we fill in the specification of the function.

```
##Arguments:
##  scenarios is a list of scenarios, which are vectors that index into data
##  data is a data.frame containing the rank order information
##  ... - an argument that accepts anything else and just passes it along
##Return:
##  data.frame containing shares with columns corresponding to the columns in the 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,...)
    ##  calculate market shares and save to right columns in res for the scenario
  }
```

```
  res = as.data.frame(res); names(res) = names(data) #setting type and names
  res ##return result table
}
```

Now we turn to the work of the simScenarios, csimFCShares, which calculates the market shares. We start by dummying this function out with the specs and returning a generic object that allows the top-level function to execute.

```
##Arguments: scen indicates which columns of data are included in the scenario
##data is full set of data Returns: vector of shares with length equal to number
##of products in scenario
simFCShares = function(scen,data){
  #subset matrix of options to correspond to products in the scenario
  #make consumers decisions by selecting the best ranked option available in the market
  #calculate vector of shares by summing decisions and dividing by rows in data
  #return shares
  rep(1/length(scen),length(scen)) #dummied result where all shares are equal in scenario to allow simS
}
#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
```

We now see our program works with the dummied data. Now we create a version with the algorithm.

```
##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 produ
  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
```

We have a very nice function now for simulating market shares for multiple scenarios given rank-order preferences! So now we can think about extending the function. In this example, there are two extensions dealing with rank data.

First, what if the highest number is best instead of the lowest? To handle this, we can just make a minor modification of the simFCShares function. We will add an argument bestValueIs that accepts either "low" or "high". If "high" is given, we

```
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)
```

```
## 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
```

```
##But this doesn't work! Why is this so? Some scenarios get noone choosing a product inmkt
#simScenarios(scens,data[,2:6],bestValueIsLow=FALSE)
#Specifically, scenarios 4 and 6 are problems. For example:
#simFCShares(scens[[4]],data[,2:6],bestValueIsLow=FALSE)
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
```

```
## 3                  NA               0.390             0.005     0.01
## 4               0.490               0.005                NA     0.01
## 5               0.545               0.005                NA     0.45
##   Kolander.s.Extra.Creamy
## 1                      NA
## 2                   0.595
## 3                   0.595
## 4                   0.495
## 5                      NA
```

So, the second issue is a bit bigger of an issue. What if not all brands are chosen by some consumer? If this happens, the current algorithm breaks. Why? Because the algorithm relies on bestOpts having some of each possible value. That is needed to get the same sized result vector each time.

To fix this, we have to think of a different algorithm. Ideally, this would be a more robust one. How else could you make the consumer decisions?