whiskey

October 31, 2022

[ ]:
```
using JuMP
#Import HiGHS solver
using HiGHS

#Create a JuMP model named picframe1 that will be solved using the HiGHS solver
wiskey=Model(HiGHS.Optimizer);
wistype = [:std,:cho,:pri];
blends = [:scot,:johnny];
wisAvilable=Dict(zip(wistype,[1200,2500,2000]));
wisCost=Dict(zip(wistype,[4,5,7]));
demand=Dict(zip(blends,[1000,600]));
wisPrice=Dict(zip(blends,[6.8,5.7]));

#variable for money spend on advertise and quantity used for each type
@variable(wiskey,x[wistype,blends]>=0);
@variable(wiskey,money[blends]>=0);

#Sup and Inf for specific type of liquor
@constraint(wiskey,PSInf,x[:pri,:scot]>=0.6*sum(x[i,:scot] for i in wistype));
@constraint(wiskey,SCSup,x[:std,:scot]<=0.2*sum(x[i,:scot] for i in wistype));
@constraint(wiskey,PJInf,x[:pri,:johnny]>=0.15*sum(x[i,:johnny] for i in␣
 ↪wistype));
@constraint(wiskey,SJSup,x[:std,:johnny]<=0.6*sum(x[i,:johnny] for i in␣
 ↪wistype));

#selling constant with advertise
@constraint(wiskey,prodcutInf[j in blends],sum(x[i,j] for i in␣
 ↪wistype)<=demand[j]+1.25*money[j]);

#avilable liquor
@constraint(wiskey, WisConstraint[i in wistype], sum(x[i,j] for j in␣
 ↪blends)<=wisAvilable[i]);

#Objetive with revenue-cost-money spend on advertise
@objective(wiskey,Max, sum(x[i,j] for i in wistype for j in blends)*wisPrice[j]-
sum(x[i,j] for i in wistype for j in blends)*wisCost[i]-sum(money[i] for i in␣
 ↪blends));
```

```
print(wiskey);
```

```
Max 2.8 x[std,scot] + 1.7000000000000002 x[std,johnny] + 1.7999999999999998
x[cho,scot] + 0.7000000000000002 x[cho,johnny] - 0.20000000000000018 x[pri,scot]
- 1.2999999999999998 x[pri,johnny] - money[scot] - money[johnny]
Subject to
 PSInf : -0.6 x[std,scot] - 0.6 x[cho,scot] + 0.4 x[pri,scot]  0.0
 PJInf : -0.15 x[std,johnny] - 0.15 x[cho,johnny] + 0.85 x[pri,johnny]  0.0
 SCSup : 0.8 x[std,scot] - 0.2 x[cho,scot] - 0.2 x[pri,scot]  0.0
 SJSup : 0.4 x[std,johnny] - 0.6 x[cho,johnny] - 0.6 x[pri,johnny]  0.0
 prodcutInf[scot] : x[std,scot] + x[cho,scot] + x[pri,scot] - 1.25 money[scot]
1000.0
 prodcutInf[johnny] : x[std,johnny] + x[cho,johnny] + x[pri,johnny] - 1.25
money[johnny]  600.0
 WisConstraint[std] : x[std,scot] + x[std,johnny]  1200.0
 WisConstraint[cho] : x[cho,scot] + x[cho,johnny]  2500.0
 WisConstraint[pri] : x[pri,scot] + x[pri,johnny]  2000.0
 x[std,scot]  0.0
 x[cho,scot]  0.0
 x[pri,scot]  0.0
 x[std,johnny]  0.0
 x[cho,johnny]  0.0
 x[pri,johnny]  0.0
 money[scot]  0.0
 money[johnny]  0.0
```

[ ]:
```
optimize!(wiskey)
@show objective_value(wiskey)
@show value.(x);
```

```
Presolving model
9 rows, 8 cols, 26 nonzeros
9 rows, 8 cols, 26 nonzeros
Presolve : Reductions: rows 9(-0); columns 8(-0); elements 26(-0)
Solving the presolved LP
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0    -1.1199987222e+01 Ph1: 7(10.8); Du: 4(11.2) 0s
          7    -1.6133333333e+03 Pr: 0(0) 0s
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Simplex   iterations: 7
Objective value     :  1.6133333333e+03
HiGHS run time      :          0.00
objective_value(wiskey) = 1613.3333333333335
value.(x) = 2-dimensional DenseAxisArray{Float64,2,…} with index sets:
```

```
    Dimension 1, [:std, :cho, :pri]
    Dimension 2, [:scot, :johnny]
And data, a 3×2 Matrix{Float64}:
 199.99999999999997  1000.0
 200.00000000000006   416.66666666666674
 600.0                250.0
```