(SMM641) - Revenue Management & Pricing. Incorporating Buy-up and Buy-down Behaviours in Capacity Allocation, Overbooking - R Supplement

Oben Ceryan

Contents

1 Required Packages and Data

1.1 Use Package lpSolve

Type the following code to install and activate the **lpSolve** package:

```
install.packages("lpSolve",repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/rx/_gm3py093nd25flx8hwzm1bm0000gp/T//RtmpCSOL3z/downloaded_packages
```

```
library(lpSolve)
```

```
## Warning: package 'lpSolve' was built under R version 3.5.2
```

1.2 Data Files

Download and place in the R working directory the data file: “Restaurant.csv”

2 Network Revenue Management with Bid Price Heuristics

2.1 Problem Parameters

```
# Network Revenue Management with Dynamic Porgramming

N1=100; # Leg 1 seat availability
N2=120; # Leg 2 seat availability
TT=300; # Length of time horizon

arrivalprob=c(1/5, 4/15, 1/6, 4/15);

price=c(150,120,250,180);

totalarrivalprob=sum(arrivalprob);
noarrivalprob=1-totalarrivalprob;
```

2.2 Solving the corresponding LP with Expected Demand Values

```r
# Airline DP with LP Heuristics

# Objective Function Coefficients
obj.fun <- c(150,120,250,180);
expdemands <- arrivalprob*300;
AllocateLessThanDLcap<-c(1,0,1,1)
AllocateLessThanLEcap<-c(0,1,1,1)
AllocateLessThanDemand<-diag(1, 4, 4)

constr <- rbind(AllocateLessThanDLcap,AllocateLessThanLEcap,
                AllocateLessThanDemand);

# Constraint directions:
constr.dir <- c(rep("<=", 2),rep("<=", 4));

# Constraint Right Hand Side
rhs <- c(100,120,expdemands)

# Solving the LP:
optairline <- lp("max", obj.fun, constr, constr.dir, rhs, compute.sens=TRUE)

# Optimal Solution (Values for Decision Variables)
optairline$solution
```

```
## [1] 60 80 40  0
```

```r
# Optimal Objective Function Value
revenueLP<-optairline$objval
print(revenueLP)
```

```
## [1] 28600
```

2.3 Obtaining Bid Prices for Heuristics

```r
# Bid Prices for Capacity Constraints
bidprices<-optairline$duals[1:2]

# The Bid Price of the first flight leg
```

```r
print(paste("The Bid Price for the first flight leg:",bidprices[1]))
```

```
## [1] "The Bid Price for the first flight leg: 130"
```

```r
# The Bid Price of the second flight leg
print(paste("The Bid Price for the second flight leg:",bidprices[2]))
```

```
## [1] "The Bid Price for the second flight leg: 120"
```

```r
# We can set a heuristic accapetance rule as follows:
# Accept all Product 1 demand as price>=value of resources.
# Accept all Product 2 demand as price>=value of resources.
# Accept all Product 3 demand as price>=value of resources.
# Do not accept Product 4 demand as price<value of resources.
```

2.4 Implementing the Bid Price Heuristic

```r
# Defining arrays with correct dimensions:
v=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept1=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept2=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept3=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));
accept4=array(rep( 0, len=(N1+1)*(N2+1)*(TT+1)), dim=c(N1+1,N2+1,TT+1));

# Initialization / SettingTerminal Values:
for(i in 1:(N1+1)){
    for(j in 1:(N2+1)){
        v[i,j,1]=0;
    }
}

# The Value Function Recursions

for(t in 2:(TT+1)){ #2:TT+1
    for(i in 1:(N1+1)){ #1:N1+1
        for(j in 1:(N2+1)){ #1:N2+1

            # For no arrivals:
            vforarrival0=v[i,j,t-1];
```

```
# For Product 1 arrival:
# default not accept unless able to accept
vforarrival1=v[i,j,t-1];
accept1[i,j,t]=0;
# If resource available:
if(i>1){
    vforarrival1=price[1]+v[i-1,j,t-1];
    accept1[i,j,t]=1;
}

# For Product 2 arrival:
# default not accept unless able to accept
vforarrival2=v[i,j,t-1];
accept2[i,j,t]=0;
# If resource available:
if(j>1){
    vforarrival2=price[2]+v[i,j-1,t-1];
    accept2[i,j,t]=1;
}

# For Product 3 arrival:
# default not accept unless able to accept
vforarrival3=v[i,j,t-1];
accept3[i,j,t]=0;
# If resources available:
if(i>1){
    if(j>1){
        vforarrival3=price[3]+v[i-1,j-1,t-1];
        accept3[i,j,t]=1;
    }
}

# For Product 4 arrival:
# Do not accept
vforarrival4=v[i,j,t-1];
accept4[i,j,t]=0;

# Obtaining the overall value function from its parts:
v[i,j,t]=noarrivalprob*vforarrival0+
```

```
                arrivalprob[1]*vforarrival1+
                arrivalprob[2]*vforarrival2+
                arrivalprob[3]*vforarrival3+
                arrivalprob[4]*vforarrival4;
        }
    }
}

# Bid Price Heuristic Revenue
revenueBidPriceHeuristic<-v[101,121,301]
print(revenueBidPriceHeuristic)
```

```
## [1] 28150.06
```

3 Buy-up Behaviour in Capacity Allocation

3.1 Recall Example from Week 1

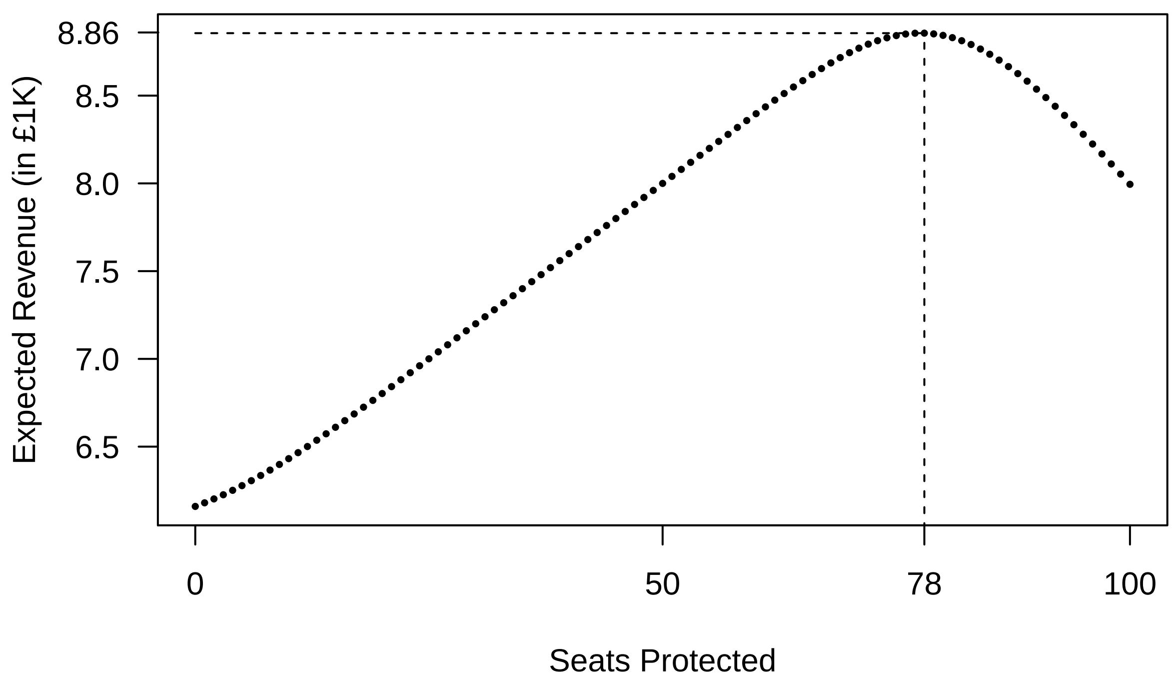
Please see class slides for the problem description.

```
# Base Case in Capacity Allocation

mL=100           # Mean Demand for Low-Fare, Poisson
mH=80           # Mean Demand for High-Fare, Poisson
pL=60           # Price for Low-Fare
pH=100          # Price for Low-Fare
capacity=100    # Capacity
ExpRevenue=rep(0,capacity+1)
for (i in 1:(capacity+1)){
    protect=i-1
    availforLowFare=capacity-protect;
    ExpRevenue[i]=0;
    for(dL in 50:150){
        soldLowFare=min(availforLowFare,dL)
        remainforHighFare=capacity-soldLowFare
        for(dH in 40:120){
            soldHighFare=min(remainforHighFare,dH)
            RevenueThisIter=pL*soldLowFare+pH*soldHighFare
            ExpRevenue[i]=ExpRevenue[i]+
```

```
                RevenueThisIter*dpois(dL,mL)*dpois(dH,mH)
        }
    }
}
Protectindexbest = which(ExpRevenue == max(ExpRevenue))
ProtectBest=Protectindexbest-1
OptimalExpRevenue=max(ExpRevenue)
print(paste("The Optimal Protection Level for High-Fare Demand:", ProtectBest))
```

```
## [1] "The Optimal Protection Level for High-Fare Demand: 78"
```

```
# Plotting Expected Revenue vs Protection Level
xaxis=0:capacity
plot(xaxis,ExpRevenue/1000,pch = 16, cex = 0.5,las=1, xaxt="n",
     xlab="Seats Protected",ylab="Expected Revenue (in £1K)")
xticks <- seq(0, capacity, by=50)
axis(side = 1, at = xticks)
axis(side = 1, at = ProtectBest)
lines(c(ProtectBest,ProtectBest),c(0, max(ExpRevenue)/1000),lty=2)
axis(side = 2, at = round(max(ExpRevenue)/1000,2),las=1)
lines(c(0,ProtectBest),c(max(ExpRevenue)/1000, max(ExpRevenue)/1000),lty=2)
```



3.2 Buy-up Behaviour in Capacity Allocation

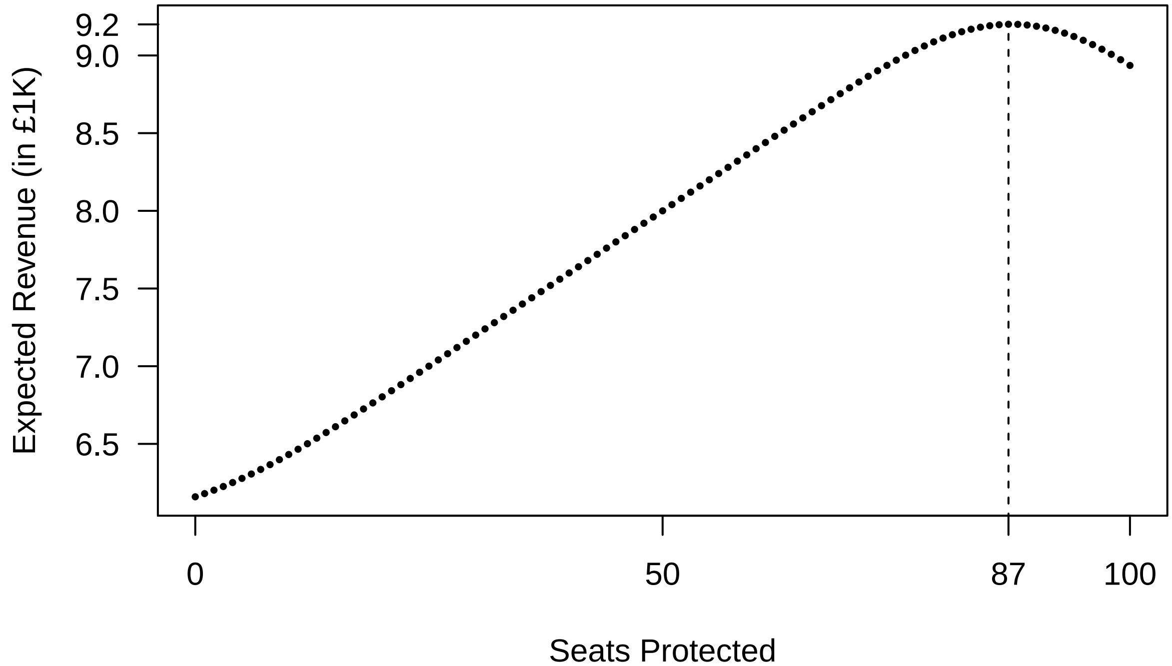
Please see class slides for the problem description.

```r
# Buy up Behaviour in Capacity Allocation

mL=100          # Mean Demand for Low-Fare, Poisson
mH=80           # Mean Demand for High-Fare, Poisson
pL=60           # Price for Low-Fare
pH=100          # Price for Low-Fare
capacity=100    # Capacity
qUp=0.1;          # Fraction Low Fare Buy Up
ExpRevenue=rep(0,capacity+1)
for (i in 1:(capacity+1)){
  protect=i-1
  availforLowFare=capacity-protect;
  ExpRevenue[i]=0;
  for(dL in 0:150){
    soldLowFare=min(availforLowFare,dL)
    unmetLowFare=dL-soldLowFare
    remainforHighFare=capacity-soldLowFare
    for(dH in 0:150){
      soldHighFare=min(remainforHighFare,dH+qUp*unmetLowFare)
      RevenueThisIter=pL*soldLowFare+pH*soldHighFare
      ExpRevenue[i]=ExpRevenue[i]+
        RevenueThisIter*dpois(dL,mL)*dpois(dH,mH)
    }
  }
}
Protectindexbest = which(ExpRevenue == max(ExpRevenue))
ProtectBest=Protectindexbest-1
OptimalExpRevenue=max(ExpRevenue)
print(paste("The Optimal Protection Level for High-Fare Demand:", ProtectBest))
```

```
## [1] "The Optimal Protection Level for High-Fare Demand: 87"
```

```r
# Plotting Expected Revenue vs Protection Level
xaxis=0:capacity
plot(xaxis,ExpRevenue/1000,pch = 16, cex = 0.5,las=1, xaxt="n",
     xlab="Seats Protected",ylab="Expected Revenue (in £1K)")
xticks <- seq(0, capacity, by=50)
```

```
axis(side = 1, at = xticks)
axis(side = 1, at = ProtectBest)
lines(c(ProtectBest,ProtectBest),c(0, max(ExpRevenue)/1000),lty=2)
axis(side = 2, at = round(max(ExpRevenue)/1000,2),las=1)
```



4 Buy-down Behaviour in Capacity Allocation

Please see class slides for the problem description.

4.1 If Firm Unaware of Buy-Down Behaviour

4.1.1 Iteration 0: Firm picks protection level 78 (original protection level)

```
# Suppose the firm continues uses the original protection level of 78
# but there are customers in the market who buy down

mL=100          # Mean Demand for Low-Fare, Poisson
mH=80           # Mean Demand for High-Fare, Poisson
pL=60           # Price for Low-Fare
pH=100          # Price for Low-Fare
capacity=175    # Capacity
qDown=0.4;      # Fraction Low Fare Buy Up
```

```r
ExpRevenue=rep(0,capacity+1)
ExpSoldHighFare=rep(0,capacity+1)
for (i in 79:79){ # incorporating protection level ignoring buy-down behaviour
    protect=i-1
    availforLowFare=capacity-protect;
    ExpRevenue[i]=0;
    ExpSoldHighFare[i]=0;
    for(dL in 0:150){
        for(dH in 0:150){
            lowBoughtLowFare=min(availforLowFare,dL)
            remainingCapatLowFare=availforLowFare-lowBoughtLowFare
            highBoughtLowFare=min(remainingCapatLowFare,qDown*dH)
            soldLowFare=lowBoughtLowFare+highBoughtLowFare

            remainingHighFareDemand=dH-highBoughtLowFare
            remainingCapforHighFare=capacity-soldLowFare
            soldHighFare=min(remainingHighFareDemand,remainingCapforHighFare)

            RevenueThisIter=pL*soldLowFare+pH*soldHighFare
            ExpRevenue[i]=ExpRevenue[i]+
                RevenueThisIter*dpois(dL,mL)*dpois(dH,mH)
            ExpSoldHighFare[i]=ExpSoldHighFare[i]+
                soldHighFare*dpois(dL,mL)*dpois(dH,mH)
        }
    }
}

print(paste("Expected Revenue:", ExpRevenue[i]))
```

```
## [1] "Expected Revenue: 13188.3472276875"
```

```r
print(paste("Expected Observed High Fare Demand", ExpSoldHighFare[i]))
```

```
## [1] "Expected Observed High Fare Demand 73.6839316608279"
```

4.1.2 Iteration 1: Firm picks protection level 72 (based on *new* High Fare Demand)

```r
# Suppose the firm now picks a protection level of 72
# observing a lower high fare demand of mean 74.
# The potection level of 72 is found by the code correspponding to the
```

```r
# base allocation model (see 2.1) but with an mean demand for high fare of 74.
# (You can update the capacity there to 175 but recall from week 1 that
# capacity does not impact the protection level so you can also keep 100.)

mL=100          # Mean Demand for Low-Fare, Poisson
mH=80           # Mean Demand for High-Fare, Poisson
pL=60           # Price for Low-Fare
pH=100          # Price for Low-Fare
capacity=175    # Capacity
qDown=0.4;      # Fraction Low Fare Buy Up
ExpRevenue=rep(0,capacity+1)
ExpSoldHighFare=rep(0,capacity+1)
for (i in 73:73){ # incorporating protection level ignoring buy-down behaviour
    protect=i-1
    availforLowFare=capacity-protect;
    ExpRevenue[i]=0;
    ExpSoldHighFare[i]=0;
    for(dL in 0:150){
        for(dH in 0:150){
            lowBoughtLowFare=min(availforLowFare,dL)
            remainingCapatLowFare=availforLowFare-lowBoughtLowFare
            highBoughtLowFare=min(remainingCapatLowFare,qDown*dH)
            soldLowFare=lowBoughtLowFare+highBoughtLowFare

            remainingHighFareDemand=dH-highBoughtLowFare
            remainingCapforHighFare=capacity-soldLowFare
            soldHighFare=min(remainingHighFareDemand,remainingCapforHighFare)

            RevenueThisIter=pL*soldLowFare+pH*soldHighFare
            ExpRevenue[i]=ExpRevenue[i]+
                RevenueThisIter*dpois(dL,mL)*dpois(dH,mH)
            ExpSoldHighFare[i]=ExpSoldHighFare[i]+
                soldHighFare*dpois(dL,mL)*dpois(dH,mH)
        }
    }
}

print(paste("Expected Revenue:", ExpRevenue[i]))
```

```
## [1] "Expected Revenue: 13038.8714757313"
```

```r
print(paste("Expected Observed High Fare Demand", ExpSoldHighFare[i]))
```

```
## [1] "Expected Observed High Fare Demand 68.592687628427"
```

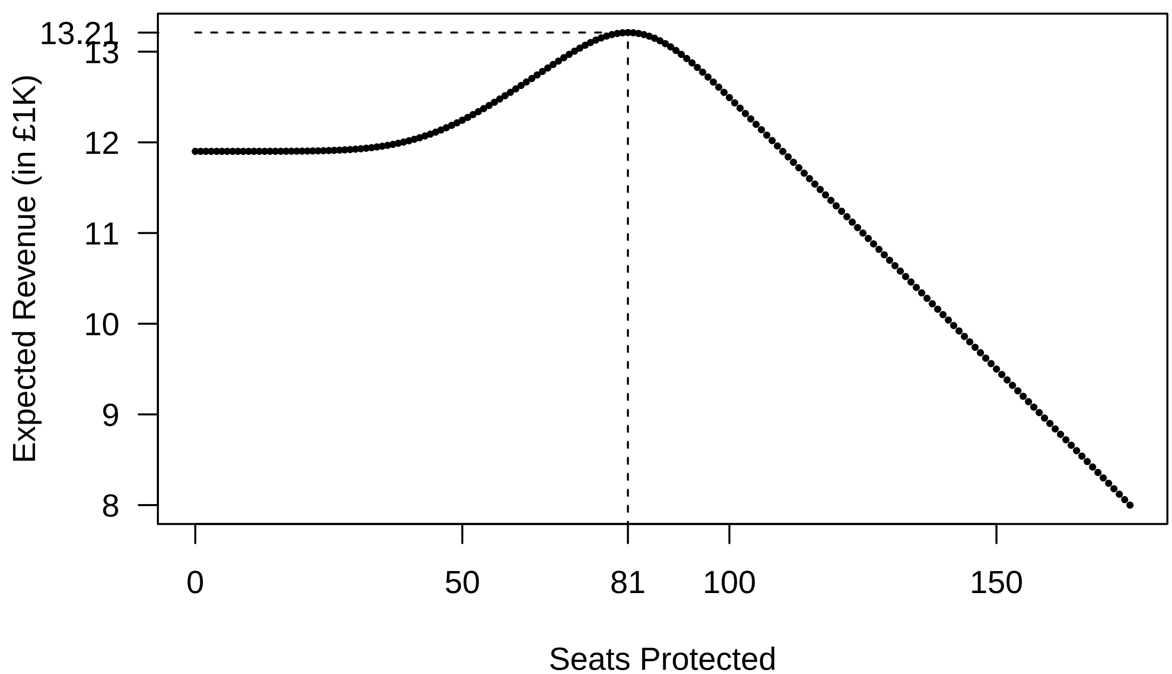
4.2 If Firm Aware of Buy-Down Behaviour

```r
# Buy Down

mL=100          # Mean Demand for Low-Fare, Poisson
mH=80           # Mean Demand for High-Fare, Poisson
pL=60           # Price for Low-Fare
pH=100          # Price for Low-Fare
capacity=175    # Capacity
qDown=0.4;      # Fraction Low Fare Buy Up
ExpRevenue=rep(0,capacity+1)
ExpSoldHighFare=rep(0,capacity+1)
for (i in 1:(capacity+1)){
    protect=i-1
    availforLowFare=capacity-protect;
    ExpRevenue[i]=0;
    ExpSoldHighFare[i]=0;
    for(dL in 0:150){
        for(dH in 0:150){
            lowBoughtLowFare=min(availforLowFare,dL)
            remainingCapatLowFare=availforLowFare-lowBoughtLowFare
            highBoughtLowFare=min(remainingCapatLowFare,qDown*dH)
            soldLowFare=lowBoughtLowFare+highBoughtLowFare

            remainingHighFareDemand=dH-highBoughtLowFare
            remainingCapforHighFare=capacity-soldLowFare
            soldHighFare=min(remainingHighFareDemand,remainingCapforHighFare)

            RevenueThisIter=pL*soldLowFare+pH*soldHighFare
            ExpRevenue[i]=ExpRevenue[i]+
                RevenueThisIter*dpois(dL,mL)*dpois(dH,mH)
            ExpSoldHighFare[i]=ExpSoldHighFare[i]+
                soldHighFare*dpois(dL,mL)*dpois(dH,mH)
        }
```

```
    }
}
Protectindexbest = which(ExpRevenue == max(ExpRevenue))
ProtectBest=Protectindexbest-1
OptimalExpRevenue=max(ExpRevenue)
print(paste("The Optimal Protection Level for High-Fare Demand:", ProtectBest))
```

```
## [1] "The Optimal Protection Level for High-Fare Demand: 81"
```

```
ExpSoldHighFareOptProtect=ExpSoldHighFare[Protectindexbest]




# Plotting Expected Revenue vs Protection Level
xaxis=0:capacity
plot(xaxis,ExpRevenue/1000,pch = 16, cex = 0.5,las=1, xaxt="n",
     xlab="Seats Protected",ylab="Expected Revenue (in £1K)")
xticks <- seq(0, capacity, by=50)
axis(side = 1, at = xticks)
axis(side = 1, at = ProtectBest)
lines(c(ProtectBest,ProtectBest),c(0, max(ExpRevenue)/1000),lty=2)
axis(side = 2, at = round(max(ExpRevenue)/1000,2),las=1)
lines(c(0,ProtectBest),c(max(ExpRevenue)/1000, max(ExpRevenue)/1000),lty=2)
```



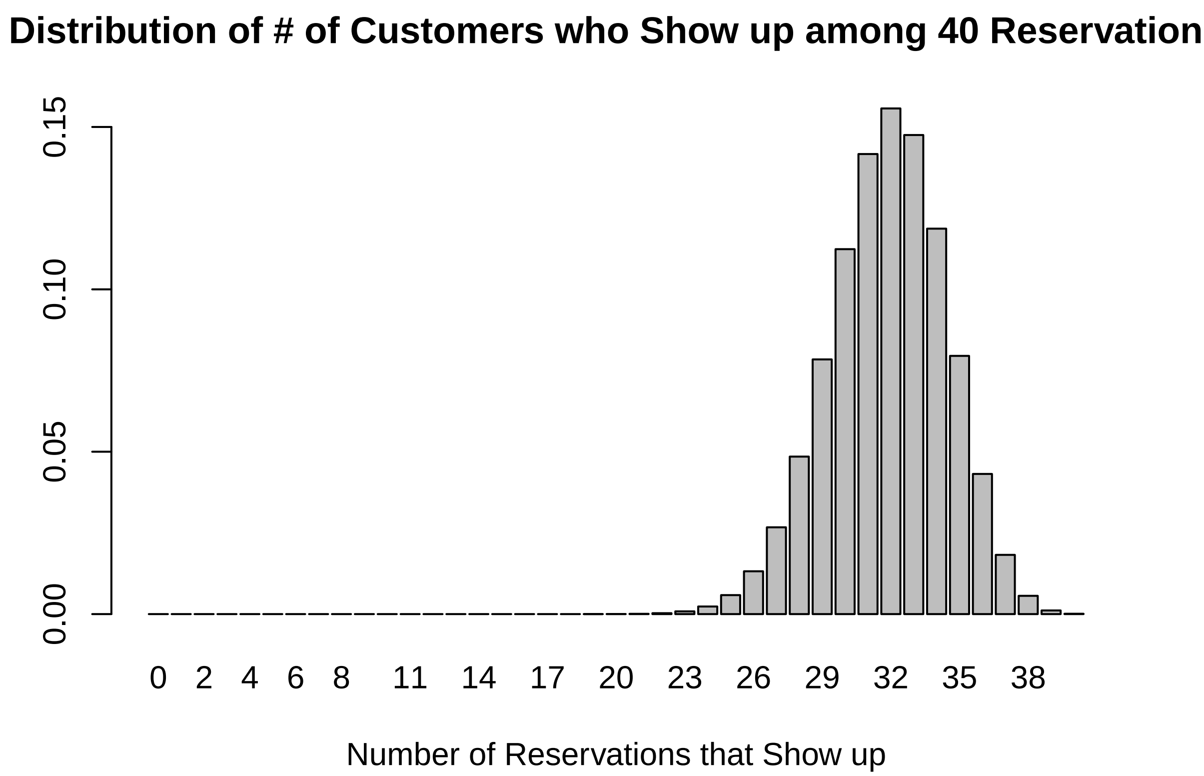
5 Overbooking - Restaurant Revenue Management Example

5.1 Importing Data

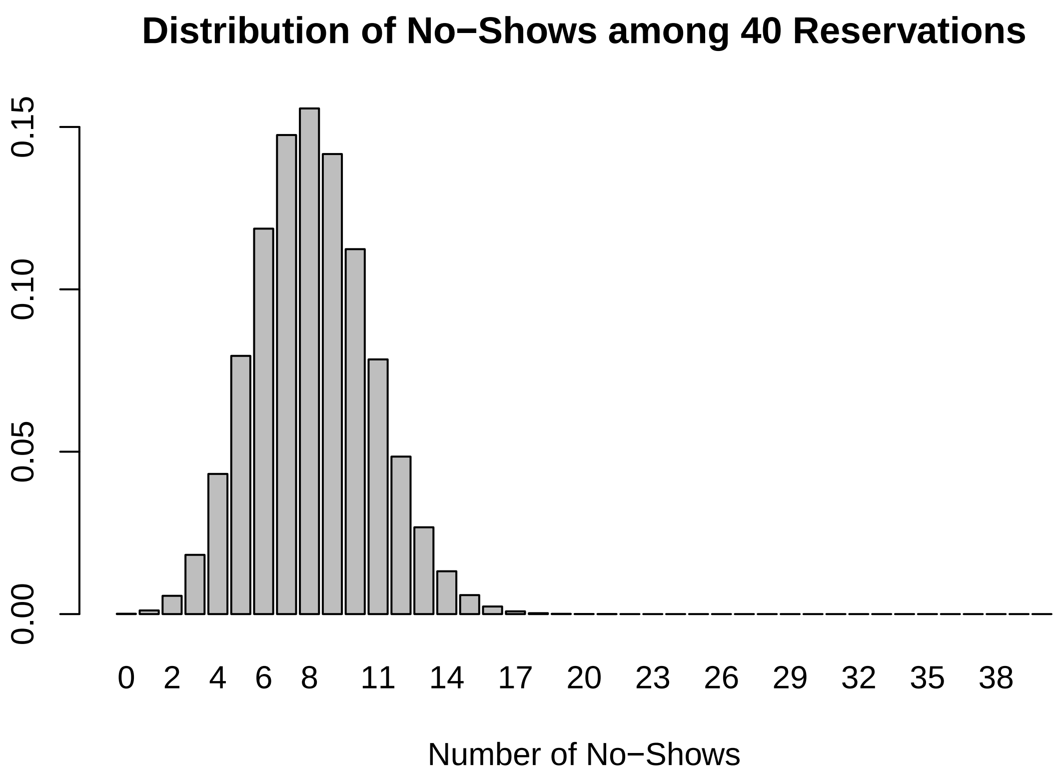
```r
# Overbooking where NoShows depend on the number of reservations taken

# Read reservations data
reservations <-read.csv("Restaurant.csv",header=T)
```

5.2 Preliminary Analysis

```r
cap=40
overbookmax=10
# quantify penalty of delay or refusing service
# e.g., loss of reputation, loss of repeat visits by customers,
# compensate by free drinks, appetizers, etc.
penalty=1000

# Estimating the probability that a reservation shows up
showProb=mean(reservations$Arrive)
# Estimating the probability that a reservation does not shows up (no-show)
noshowProb=1-showProb

# Plotting the distribution of customers who show up
x <- dbinom(0:40,size=40,prob=showProb)
barplot(x,names.arg=0:40,
        main="Distribution of # of Customers who Show up among 40 Reservations",
        xlab="Number of Reservations that Show up")
```




```r
# Plotting the distribution of customers who do not show up
x <- dbinom(0:40,size=40,prob=noshowProb)
barplot(x,names.arg=0:40,
        main="Distribution of No-Shows among 40 Reservations",
        xlab="Number of No-Shows")
```




```
# Estimating profit per Table
# Take the subset of data corresponding to reservations that show up
arrivedonlyData <- subset(reservations, reservations$Arrive >0)
# Calculate the average profit
avgPrice=mean(arrivedonlyData$Profit)
```

5.3 Optimal Overbooking Quantity

```
ExpProfit=rep(0,overbookmax+1)

for (i in 1:(overbookmax+1)){
  reservationsTaken=cap+i-1;
  ExpProfit[i]=0;
  for (j in 0:reservationsTaken){
      arrive=j
      serve=min(arrive,cap)
      delay=arrive-serve
      ProfitThisIter=avgPrice*serve-penalty*delay
      ExpProfit[i]=ExpProfit[i]+
          dbinom(j,reservationsTaken,showProb)*ProfitThisIter
```

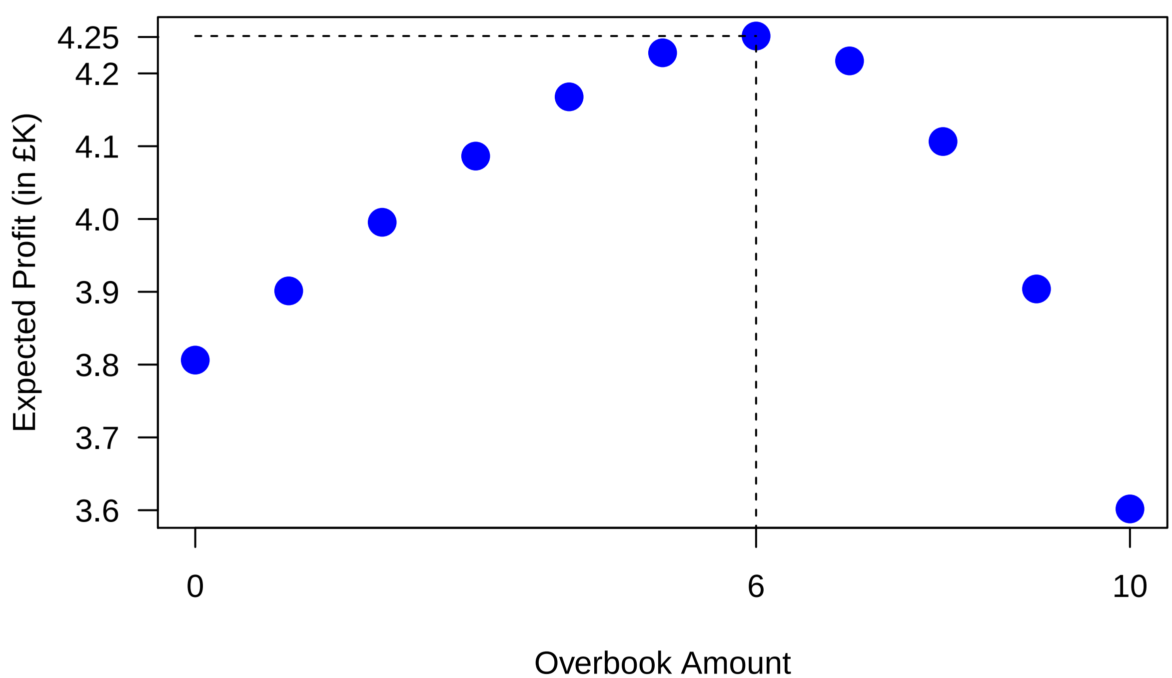
```
  }
}

overbookindexbest = which(ExpProfit == max(ExpProfit))
overbookBest=overbookindexbest-1
OptimalExpProfit=max(ExpProfit)
print(paste("The Optimal Overbooking Amount:", overbookBest))
```

```
## [1] "The Optimal Overbooking Amount: 6"
```

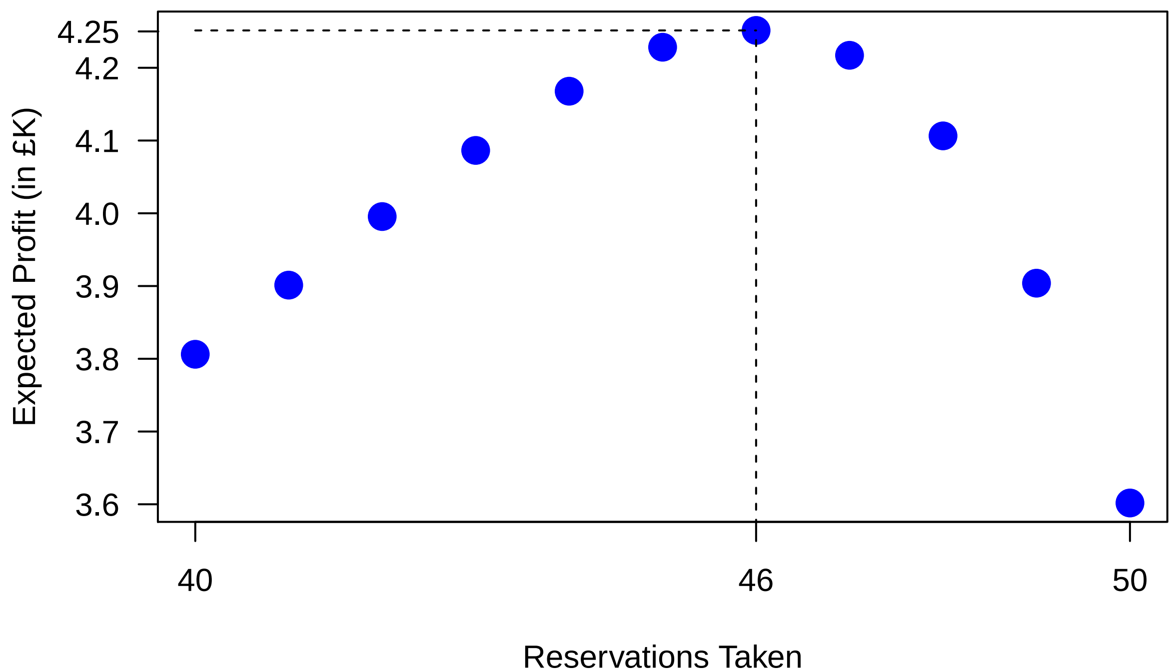
```
print(paste("The Optimal Number of Reservations to Take:",
            cap+overbookBest))
```

```
## [1] "The Optimal Number of Reservations to Take: 46"
```

```
# Plotting Expected Profit vs Overbook Amount
xaxis=0:overbookmax
plot(xaxis,ExpProfit/1000,pch = 16, col="blue",cex = 2,las=1, xaxt="n",
     xlab="Overbook Amount",ylab="Expected Profit (in £K)")
xticks <- seq(0, overbookmax, by=10)
axis(side = 1, at = xticks)
axis(side = 1, at = overbookBest)
lines(c(overbookBest,overbookBest),c(0, max(ExpProfit)/1000),lty=2)
axis(side = 2, at = round(max(ExpProfit)/1000,2),las=1)
lines(c(0,overbookBest),c(max(ExpProfit)/1000, max(ExpProfit)/1000),lty=2)
```



```r
# Plotting Expected Profit vs Reservations Taken
xaxis=cap:(cap+overbookmax)
plot(xaxis,ExpProfit/1000,pch = 16, col="blue", cex = 2,las=1, xaxt="n",
     xlab="Reservations Taken",ylab="Expected Profit (in £K)")
xticks <- seq(cap, cap+overbookmax, by=10)
axis(side = 1, at = xticks)
axis(side = 1, at = cap+overbookBest)
lines(c(cap+overbookBest,cap+overbookBest),c(0, max(ExpProfit)/1000),lty=2)
axis(side = 2, at = round(max(ExpProfit)/1000,2),las=1)
lines(c(cap,cap+overbookBest),c(max(ExpProfit)/1000, max(ExpProfit)/1000),lty=2)
```



5.4 Profit Comparison for Optimal Overbooking vs No Overbooking

```r
print(paste("Profit from Optimal Overbooking is",
            OptimalExpProfit))
```

```
## [1] "Profit from Optimal Overbooking is 4251.33436901254"
```

```r
print(paste("Profit from No Overbooking is",
            ExpProfit[1]))
```

```
## [1] "Profit from No Overbooking is 3806.1795"
```

```
print(paste("Optimal overbooking generates",
            round(100*(max(ExpProfit)-ExpProfit[1])/ExpProfit[1],1),
            "% more revenue compared to no overbooking."))
```

```
## [1] "Optimal overbooking generates 11.7 % more revenue compared to no overbooking."
```