Ex5_Scoring

Naga Soundari Balamurugan

November 5, 2018

Team: Naga Soundari Balamurugan, Aditya Wakade, Manasi Kulkarni, Mervin christo Daniel

```
#Install required libraries
library(kableExtra)
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#Read both the estimation and holdout file
estimationList <- read.csv("Data_Estimation_R.csv", header = TRUE)
holdOutList <- read.csv("Data_Holdout_R.csv", header = TRUE)
```

1. Predict y (i.e., the decision to join the club) as a function of the available scoring variables (gender and hl.) using a logistic regression approach. Include an intercept term to account for a base response rate. Keep all coefficients (i.e., do not eliminate coefficients which seems to be statistically insignificant).

```
#Create the training model
logisModel <- glm(y ~ as.factor(gender) + hl1 + hl2 + hl3 + hl5 + hl6,
                  family = binomial(link = "logit"),
                  data = estimationList)

summary(logisModel)
```

```
##
## Call:
## glm(formula = y ~ as.factor(gender) + hl1 + hl2 + hl3 + hl5 +
##     hl6, family = binomial(link = "logit"), data = estimationList)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6854  -0.9444  -0.6260   1.1212   2.1831
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -0.928404   0.361689  -2.567  0.01026 *
## as.factor(gender)1 -0.016632   0.344941  -0.048  0.96154
```

```
## hl1                  0.005733   0.001840   3.115  0.00184 **
## hl2                 -0.045830   0.026570  -1.725  0.08455 .
## hl3                 -0.068239   0.017004  -4.013 5.99e-05 ***
## hl5                  0.004349   0.026228   0.166  0.86830
## hl6                 -0.004919   0.017404  -0.283  0.77746
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 261.37  on 199  degrees of freedom
## Residual deviance: 234.26  on 193  degrees of freedom
## AIC: 248.26
##
## Number of Fisher Scoring iterations: 4
```

2. Based on your score function, score all individuals on the holdout-list (you can do this manually or adapt the R code from class). Using your model, compute (for each individual): (a) predicted response rate, (b) consequent lift (divide the predicted response rate by the average response rate in the estimation-list).

```r
#Dividing and preparing training data
x_train <- estimationList[c("id", "gender", "hl1", "hl2", "hl3", "hl5", "hl6")]
y_train <- estimationList[c("y")]

logisModel.train <- data.frame(ID = x_train$id,
    BinaryLogitProbability =  predict(logisModel, x_train, type = c("response")),
    BinaryLogitPredict = round(predict(logisModel, x_train, type =

#Calculate average response rate of the training model
avgResponseRate_train <- mean(logisModel.train$BinaryLogitProbability)
```

```r
#Dividing and preparing test data
x_test <- holdOutList[c("id", "gender", "hl1", "hl2", "hl3", "hl5", "hl6")]
y_test <- holdOutList[c("y")]

logisModel.predict <- data.frame(ID = x_test$id,
  BinaryLogitProbability = predict(logisModel, x_test, type = c("response")),
  BinaryLogitPredict = round(predict(logisModel, x_test, type =

#Calculate the Consequent lift
logisModel.predict$conseqLift <- logisModel.predict$BinaryLogitProbability/avgResponseRate_train

#Consequent lift for top 10 rows
kable(head(logisModel.predict, 10), "latex") %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

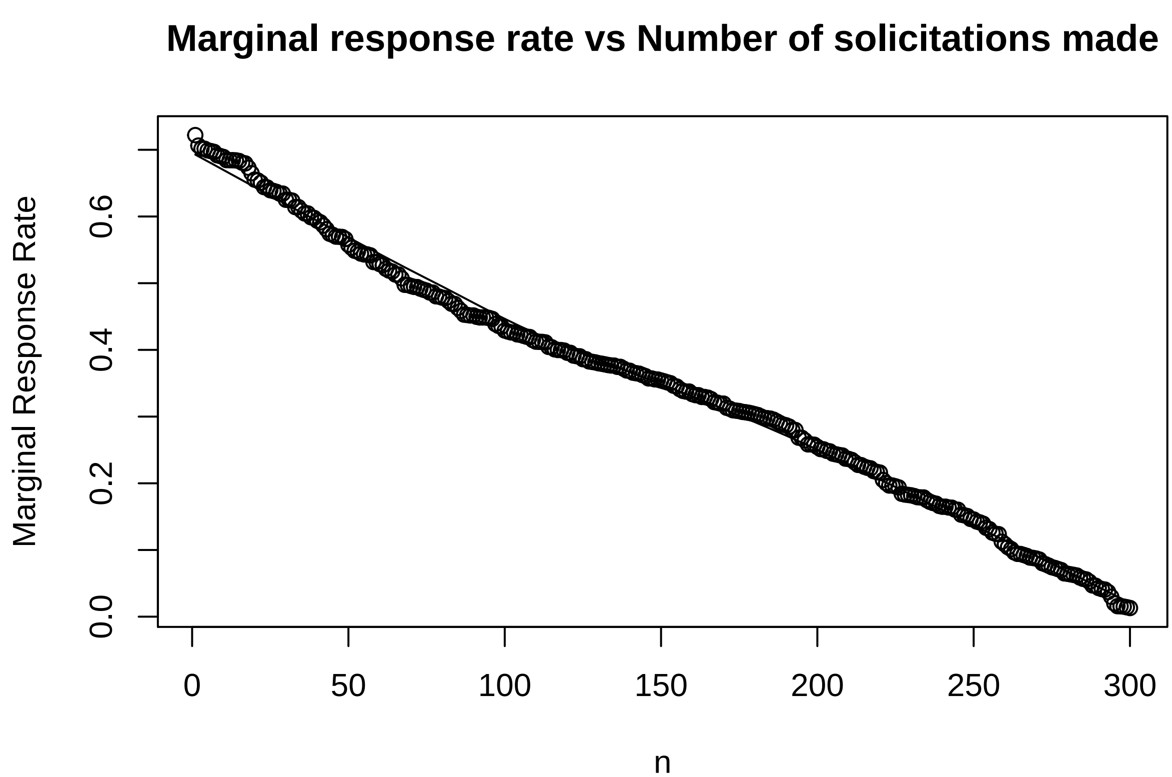
ID	BinaryLogitProbability	BinaryLogitPredict	conseqLift
201	0.4783915	0	1.3288654
202	0.5317304	1	1.4770288
203	0.2980727	0	0.8279797
204	0.6968386	1	1.9356628
205	0.2443930	0	0.6788694
206	0.3819674	0	1.0610205
207	0.5696267	1	1.5822965
208	0.3056892	0	0.8491368
209	0.4225614	0	1.1737816
210	0.4485039	0	1.2458442

3. Sort the holdout-list in decreasing order of lift

```r
#Sort the data in decresing order of consequent lift
sortedList <- logisModel.predict[order(-logisModel.predict$conseqLift),]
```

4. Plot marginal response rate vs. number of solicitations made

```r
#Marginal response rate vs solicitations
scatter.smooth(sortedList$BinaryLogitProbability, xlab = "n", ylab = "Marginal Response Rate",
               main = "Marginal response rate vs Number of solicitations made")
```



5. We know that average CLV is $30 and the solicitation cost is $12. Based on the marginal cost rule determine who the CD club should send invitations to.

```r
avg_clv <- 30
sol_cost <- 12
margin <- sol_cost/avg_clv

max_prob <- max(sortedList$BinaryLogitProbability[sortedList$BinaryLogitProbability > margin])
sortedList[sortedList$BinaryLogitProbability > margin,]
```

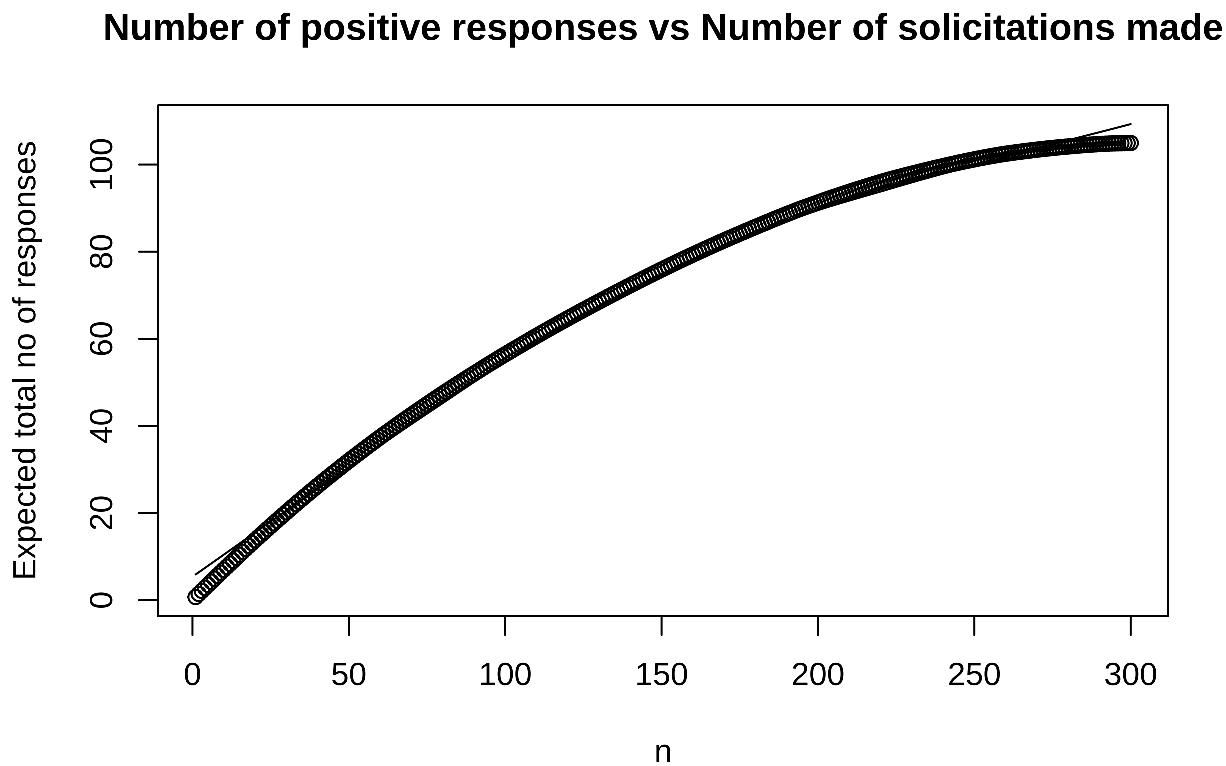
```
##      ID BinaryLogitProbability BinaryLogitPredict conseqLift
## 131 331              0.7220329                  1   2.005647
## 192 392              0.7063219                  1   1.962005
## 20  220              0.7016614                  1   1.949059
## 160 360              0.7015425                  1   1.948729
## 101 301              0.6991360                  1   1.942045
## 285 485              0.6980484                  1   1.939023
## 4   204              0.6968386                  1   1.935663
## 291 491              0.6918510                  1   1.921808
## 298 498              0.6906274                  1   1.918409
## 132 332              0.6892722                  1   1.914645
## 142 342              0.6845919                  1   1.901644
## 43  243              0.6844692                  1   1.901303
## 244 444              0.6844410                  1   1.901225
## 109 309              0.6841595                  1   1.900443
## 246 446              0.6832297                  1   1.897860
## 201 401              0.6806751                  1   1.890764
## 19  219              0.6796192                  1   1.887831
## 120 320              0.6731997                  1   1.869999
## 238 438              0.6649412                  1   1.847059
## 200 400              0.6553232                  1   1.820342
## 69  269              0.6540271                  1   1.816742
## 255 455              0.6507475                  1   1.807632
## 51  251              0.6440742                  1   1.789095
## 58  258              0.6435766                  1   1.787713
## 100 300              0.6394187                  1   1.776163
## 113 313              0.6382989                  1   1.773053
## 256 456              0.6368399                  1   1.769000
## 143 343              0.6344203                  1   1.762279
## 15  215              0.6343185                  1   1.761996
## 156 356              0.6250933                  1   1.736370
## 243 443              0.6249601                  1   1.736000
## 75  275              0.6238567                  1   1.732935
## 293 493              0.6141497                  1   1.705971
## 27  227              0.6140758                  1   1.705766
## 41  241              0.6087012                  1   1.690837
## 157 357              0.6045968                  1   1.679436
## 262 462              0.6045968                  1   1.679436
## 17  217              0.5988902                  1   1.663584
## 25  225              0.5979038                  1   1.660844
## 138 338              0.5937618                  1   1.649338
## 68  268              0.5913365                  1   1.642601
## 282 482              0.5864341                  1   1.628984
## 64  264              0.5809027                  1   1.613618
```

```
## 217 417               0.5741295             1   1.594804
## 137 337               0.5724321             1   1.590089
## 179 379               0.5698706             1   1.582974
## 7   207               0.5696267             1   1.582297
## 241 441               0.5692544             1   1.581262
## 73  273               0.5663275             1   1.573132
## 57  257               0.5574652             1   1.548514
## 176 376               0.5530723             1   1.536312
## 294 494               0.5486044             1   1.523901
## 130 330               0.5472562             1   1.520156
## 93  293               0.5443815             1   1.512171
## 127 327               0.5431771             1   1.508825
## 215 415               0.5431322             1   1.508701
## 300 500               0.5417579             1   1.504883
## 2   202               0.5317304             1   1.477029
## 180 380               0.5315889             1   1.476636
## 166 366               0.5287005             1   1.468612
## 124 324               0.5270612             1   1.464059
## 129 329               0.5215843             1   1.448845
## 125 325               0.5189094             1   1.441415
## 30  230               0.5174264             1   1.437295
## 228 428               0.5129949             1   1.424986
## 263 463               0.5129949             1   1.424986
## 33  233               0.5072640             1   1.409067
## 14  214               0.4976086             0   1.382246
## 92  292               0.4973733             0   1.381592
## 61  261               0.4957980             0   1.377217
## 154 354               0.4945004             0   1.373612
## 197 397               0.4941449             0   1.372625
## 22  222               0.4918532             0   1.366259
## 151 351               0.4902079             0   1.361688
## 24  224               0.4888204             0   1.357834
## 260 460               0.4860576             0   1.350160
## 90  290               0.4859040             0   1.349733
## 219 419               0.4803707             0   1.334363
## 89  289               0.4798224             0   1.332840
## 1   201               0.4783915             0   1.328865
## 194 394               0.4775025             0   1.326396
## 56  256               0.4732196             0   1.314499
## 258 458               0.4691465             0   1.303185
## 227 427               0.4688826             0   1.302452
## 77  277               0.4628997             0   1.285833
## 144 344               0.4585221             0   1.273672
## 220 420               0.4528606             0   1.257946
## 168 368               0.4523203             0   1.256445
## 250 450               0.4516263             0   1.254517
## 12  212               0.4515982             0   1.254440
## 288 488               0.4496308             0   1.248974
## 99  299               0.4486795             0   1.246332
## 278 478               0.4486795             0   1.246332
## 10  210               0.4485039             0   1.245844
## 270 470               0.4478455             0   1.244015
## 203 403               0.4463810             0   1.239947
## 274 474               0.4387737             0   1.218816
```

```
## 261 461                 0.4360303                    0   1.211195
## 159 359                 0.4346761                    0   1.207433
## 259 459                 0.4290493                    0   1.191804
## 111 311                 0.4275126                    0   1.187535
## 283 483                 0.4262489                    0   1.184025
## 191 391                 0.4260784                    0   1.183551
## 88  288                 0.4232768                    0   1.175769
## 9   209                 0.4225614                    0   1.173782
## 70  270                 0.4211631                    0   1.169897
## 239 439                 0.4197660                    0   1.166017
## 54  254                 0.4192535                    0   1.164593
## 94  294                 0.4146887                    0   1.151913
## 295 495                 0.4123342                    0   1.145373
## 139 339                 0.4121527                    0   1.144869
## 222 422                 0.4119083                    0   1.144190
## 165 365                 0.4111984                    0   1.142218
## 170 370                 0.4044878                    0   1.123577
## 214 414                 0.4039748                    0   1.122152
## 231 431                 0.4008132                    0   1.113370
```

6. Compute the cumulative sum (aka running sum) for the predicted response rates in decreasing order. Plot the curve for curve for number of positive responses vs. number of solicitations made.

```r
sortedList$Sum_Probability <- 0
sortedList$Sum_Probability[1] <- sortedList$BinaryLogitProbability[1]

for(i in 2:nrow(sortedList)) {
  sortedList$Sum_Probability[i] <- sortedList$Sum_Probability[i-1] +
    sortedList$BinaryLogitProbability[i]
}

scatter.smooth(sortedList$Sum_Probability, xlab = "n", ylab = "Expected total no of responses",
               main = "Number of positive responses vs Number of solicitations made")
```



7. The CD club has only 40 items of the collector's edition of "Pink Floyd's The Wall". Based on the limited supply rule, which prospects (and how many) on the hold-out list should the CD club send an invitation to?

According to the limited supply rule, as we have only 40 items of the collector's edition of "Pink Floyd's The Wall", we should look the list the people who has the most probaility to purchase on seeing an invitation and limit the invitation to specific number of people. In order to calculate that specific number of people, we need to sort the people based on the probability of buying and filter all those that fall below the cumulative probability of 40.

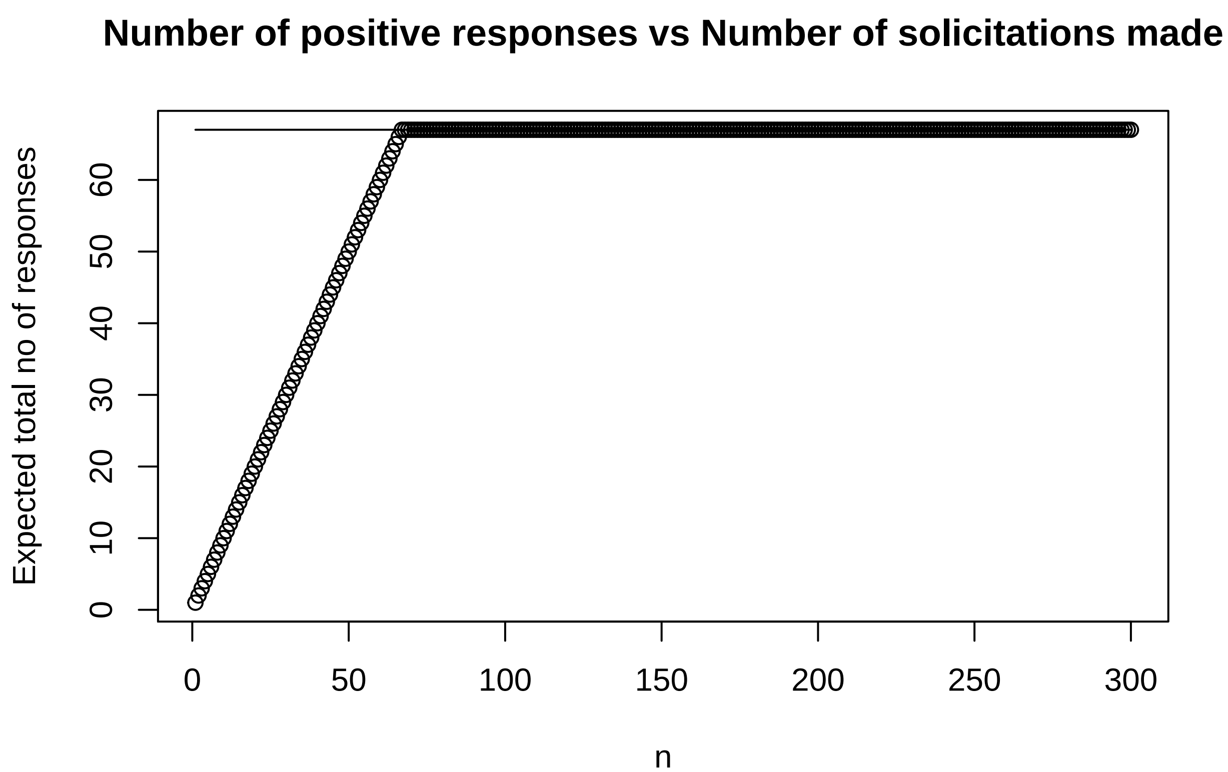
```
#Subsetting the data
limited_subset <- sortedList %>% filter(Sum_Probability < 40)
no_of_ppl <- nrow(limited_subset)
no_of_ppl
```

```
## [1] 64
```

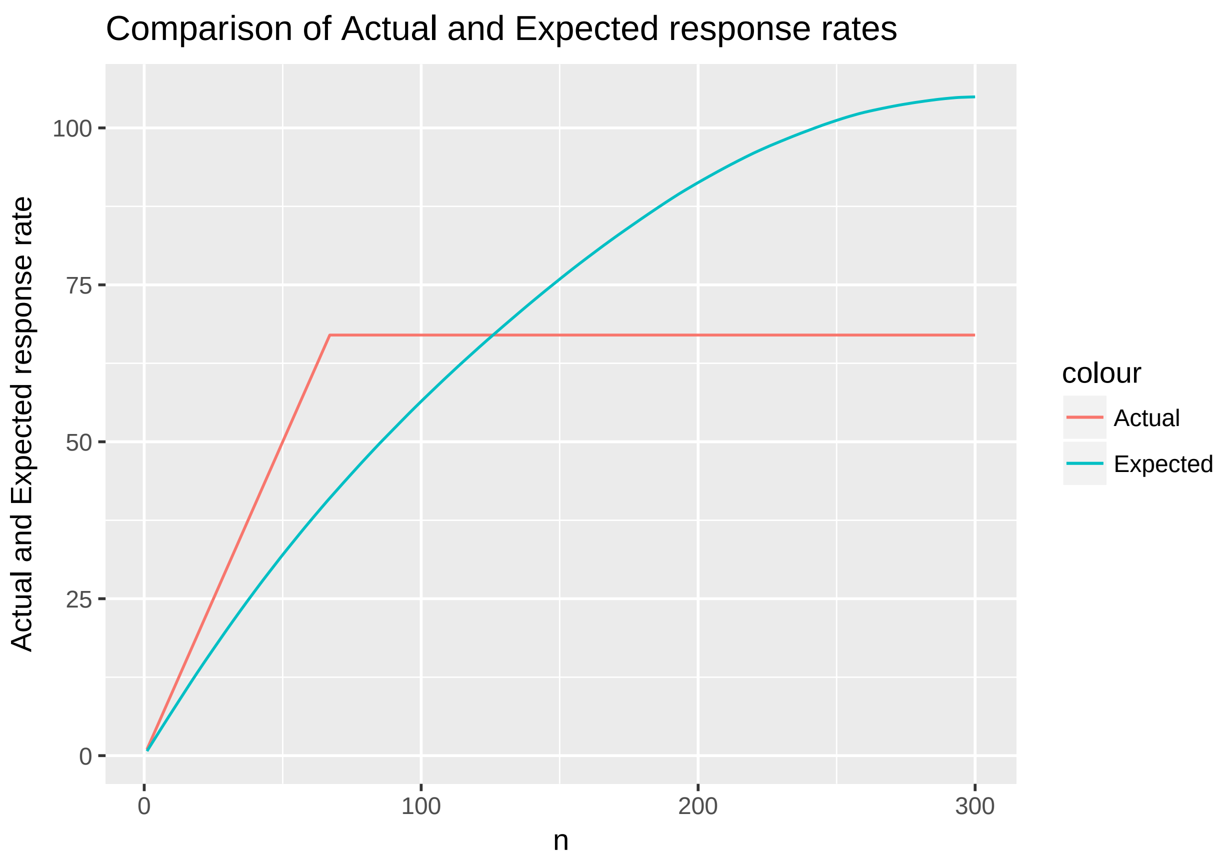
The invitation needs to be sent to 64 people.

8. Compute the cumulative sum (aka running sum) for the actual response rate (recall this is either 0 or 1) in decreasing order of predicted response rate. Plot the curve for curve for number of actual positive responses vs. number of solicitations made. Superimpose on this the curve obtained in step 6 above.

```
sortedList$Sum_Actual <- 0
sortedList$Sum_Actual[1] <- sortedList$BinaryLogitPredict[1]

for(i in 2:nrow(sortedList)) {
  sortedList$Sum_Actual[i] <- sortedList$Sum_Actual[i-1] + sortedList$BinaryLogitPredict[i]
}

scatter.smooth(sortedList$Sum_Actual, xlab = "n", ylab = "Expected total no of responses",
               main = "Number of positive responses vs Number of solicitations made")
```




```
sortedList$n <- c(1:300)

ggplot(sortedList) +
geom_line(aes(x = n, y = Sum_Actual, color = "Actual", group = 1)) +
geom_line(aes(x = n, y = Sum_Probability, color = "Expected", group = 2)) +
  xlab("n") + ylab("Actual and Expected response rate") +
  ggtitle("Comparison of Actual and Expected response rates")
```



The expected model that we see ideally follows a nice curve. For the actual model that we calculated, the data follows a sharp structure without a curve. Thus, this model is over-predicting the values. An overfit model can cause the regression coefficients, p-values and the R squared to be misleading on the results of part 7.