Churn Management

Ben Heim, Marcell Milo-Sidlo, Julia Luo, Pierre Chan, Alicia Soosai

Table of Contents

library(bit64)  
library(data.table)  
library(ggplot2)  
library(broom)  
library(knitr)

# Overview

Cell2Cell, a wireless telecommunications company (with its name altered for confidentiality), is working to reduce customer churn. The objective of this project is to create a model that predicts customer churn at Cell2Cell and to leverage insights from the model to design a targeted incentive strategy aimed at decreasing churn rates.

In the assignment, you will address these key issues:

1. Is customer churn at Cell2Cell predictable from the customer information that Cell2Cell maintains?
2. What factors drive customer churn? Which factors are particularly important?
3. What incentives should Cell2Cell offer to prevent customer churn?
4. What is the economic value of a proposed targeted plan to prevent churn, and how does this value differ across customer segments? Compare the economic value to an incentive with a cost of $100 and another incentive with a cost of $175. Which customers segments should receive the incentive? Does the answer depend on the success probability?

Note that, in what follows, the key steps you need to take are highlighted in *italic*.

# Data

All data are contained in the file Cell2Cell.RData, which is posted on Canvas.

load("Cell2Cell.RData")  
ls()

[1] "cell2cell\_DT"

Please consult the file Cell2Cell-Database-Documentation.xlsx for a description of the data and some summary statistics. Note that *calibration sample* is an alternative term for *training* or *estimation* sample.

*Report the churn rate in the calibration sample and in the validation sample and compare the two.*

mean(cell2cell\_DT[calibrat == 1, churn])

[1] 0.5

mean(cell2cell\_DT[calibrat == 0, churn])

[1] 0.01961542

The mean of the calibration sample is .5 while the mean of the validation sample is .0196. The calibration sample’s mean is significantly higher - over 25x as high.

You can see that the calibration sample was selected using *oversampling*. The purpose of oversampling was to obtain more precise estimates (lower standard errors) when estimating a logistic regression model. The validation sample, on the other hand, was not created using oversampling and represents the *true churn rate* in the data.

As you can see, some variables have missing values, which—as you know by now—is common and of no concern (unless the missing values indicate some *systematic* flaws or bias in how the data were constructed). Most estimation methods in R will automatically delete rows with missing values before estimating the model. However, the predict methods will yield NA values if a row in the data used for prediction contains missing values. Hence, in a situation where you don’t need to keep the full data I recommend to remove any observations with missing values before you conduct the main analysis.

*Perform this data-cleaning step.*

cell2cell\_DT = cell2cell\_DT[complete.cases(cell2cell\_DT),]

# Model estimation

*Estimate a logit model to predict the conditional churn probability.*

You can inspect the regression output using methods you already used, such as summary. Having said this, especially when you have a large number of inputs, it can be convenient to store the regression estimates in a table. A simple way to do this is to install the [broom package](https://cran.r-project.org/web/packages/broom/vignettes/broom.html) that has the purpose of cleaning up messy R output.

Using the tidy function in the broom package it is trivial to capture the regression output in the form of a data.table:

For kable to work, you need to load the knitr library.

calibration\_data = cell2cell\_DT[calibrat == 1]  
  
predictors = setdiff(names(calibration\_data), c("churn", "customer", "calibrat"))  
  
# Dynamically build the formula  
formula = as.formula(paste("churn ~", paste(predictors, collapse = " + ")))  
  
# Fit the logistic regression model  
fit = glm(formula, data = calibration\_data, family = binomial(link = "logit"))  
  
results\_DT = as.data.table(tidy(fit))  
kable(results\_DT, digits = 5)

| term | estimate | std.error | statistic | p.value |
| --- | --- | --- | --- | --- |
| (Intercept) | 0.14970 | 0.09527 | 1.57127 | 0.11612 |
| revenue | 0.00196 | 0.00080 | 2.46016 | 0.01389 |
| mou | -0.00028 | 0.00005 | -5.65709 | 0.00000 |
| recchrge | -0.00312 | 0.00089 | -3.51332 | 0.00044 |
| directas | -0.00120 | 0.00594 | -0.20135 | 0.84042 |
| overage | 0.00076 | 0.00028 | 2.71118 | 0.00670 |
| roam | 0.00709 | 0.00206 | 3.43641 | 0.00059 |
| changem | -0.00049 | 0.00005 | -9.19411 | 0.00000 |
| changer | 0.00230 | 0.00037 | 6.24656 | 0.00000 |
| dropvce | 0.01134 | 0.00725 | 1.56304 | 0.11804 |
| blckvce | 0.00640 | 0.00716 | 0.89452 | 0.37104 |
| unansvce | 0.00092 | 0.00045 | 2.05780 | 0.03961 |
| custcare | -0.00595 | 0.00255 | -2.33129 | 0.01974 |
| threeway | -0.03029 | 0.01125 | -2.69107 | 0.00712 |
| mourec | 0.00013 | 0.00013 | 1.01771 | 0.30882 |
| outcalls | 0.00112 | 0.00059 | 1.89446 | 0.05816 |
| incalls | -0.00311 | 0.00106 | -2.93705 | 0.00331 |
| peakvce | -0.00067 | 0.00022 | -3.05791 | 0.00223 |
| opeakvce | -0.00021 | 0.00027 | -0.78284 | 0.43372 |
| dropblk | -0.00311 | 0.00704 | -0.44249 | 0.65814 |
| callfwdv | -0.00264 | 0.02315 | -0.11414 | 0.90913 |
| callwait | 0.00208 | 0.00314 | 0.66379 | 0.50682 |
| months | -0.02128 | 0.00200 | -10.65184 | 0.00000 |
| uniqsubs | 0.18438 | 0.01999 | 9.22522 | 0.00000 |
| actvsubs | -0.20572 | 0.02791 | -7.37167 | 0.00000 |
| phones | 0.04866 | 0.01817 | 2.67837 | 0.00740 |
| models | 0.01380 | 0.02787 | 0.49501 | 0.62060 |
| eqpdays | 0.00144 | 0.00007 | 19.30918 | 0.00000 |
| age1 | -0.00330 | 0.00087 | -3.78708 | 0.00015 |
| age2 | -0.00117 | 0.00068 | -1.71810 | 0.08578 |
| children | 0.09455 | 0.02815 | 3.35918 | 0.00078 |
| credita | -0.17807 | 0.03550 | -5.01584 | 0.00000 |
| creditaa | -0.36265 | 0.03458 | -10.48750 | 0.00000 |
| prizmrur | 0.06649 | 0.04956 | 1.34154 | 0.17975 |
| prizmub | -0.03963 | 0.02441 | -1.62389 | 0.10440 |
| prizmtwn | 0.04622 | 0.03145 | 1.46985 | 0.14160 |
| refurb | 0.23404 | 0.03196 | 7.32343 | 0.00000 |
| webcap | -0.15613 | 0.03756 | -4.15660 | 0.00003 |
| truck | 0.02689 | 0.03600 | 0.74698 | 0.45508 |
| rv | 0.01186 | 0.04801 | 0.24700 | 0.80491 |
| occprof | -0.01987 | 0.03250 | -0.61131 | 0.54100 |
| occcler | 0.03949 | 0.07491 | 0.52720 | 0.59805 |
| occcrft | -0.02013 | 0.06290 | -0.32009 | 0.74890 |
| occstud | 0.11997 | 0.12187 | 0.98441 | 0.32492 |
| occhmkr | 0.25587 | 0.19008 | 1.34611 | 0.17827 |
| occret | -0.03993 | 0.09055 | -0.44096 | 0.65924 |
| occself | -0.07057 | 0.08059 | -0.87566 | 0.38122 |
| ownrent | 0.00255 | 0.04272 | 0.05978 | 0.95233 |
| marryun | 0.10881 | 0.03403 | 3.19771 | 0.00139 |
| marryyes | 0.05570 | 0.03249 | 1.71446 | 0.08644 |
| mailord | 0.00077 | 0.08565 | 0.00897 | 0.99284 |
| mailres | -0.12971 | 0.08604 | -1.50754 | 0.13167 |
| mailflag | -0.04818 | 0.08445 | -0.57055 | 0.56830 |
| travel | -0.00053 | 0.04732 | -0.01124 | 0.99103 |
| pcown | 0.03418 | 0.03096 | 1.10391 | 0.26963 |
| creditcd | 0.04202 | 0.04371 | 0.96123 | 0.33644 |
| retcalls | 0.01203 | 0.18367 | 0.06552 | 0.94776 |
| retaccpt | -0.12786 | 0.10764 | -1.18781 | 0.23491 |
| newcelly | -0.07053 | 0.02727 | -2.58607 | 0.00971 |
| newcelln | -0.00508 | 0.03153 | -0.16127 | 0.87188 |
| refer | -0.05003 | 0.04214 | -1.18703 | 0.23521 |
| incmiss | -0.09151 | 0.06006 | -1.52357 | 0.12761 |
| income | -0.01324 | 0.00603 | -2.19481 | 0.02818 |
| mcycle | 0.12231 | 0.08898 | 1.37445 | 0.16930 |
| setprcm | -0.09632 | 0.04051 | -2.37748 | 0.01743 |
| setprc | 0.00062 | 0.00028 | 2.19418 | 0.02822 |
| retcall | 0.79370 | 0.19458 | 4.07899 | 0.00005 |

# Prediction: Accounting for oversampling

The idea of oversampling is as follows. If the response rate in the data is small, there is a strong imbalance between observations with a response of and a response of . As a consequence, estimating the model is difficult and the estimates will be imprecise, i.e. they will have large standard errors.

The solution: Create a training sample with one half of observations randomly chosen from the original data with response , and the other half randomly chosen from the original data with response . Now estimation is easier and the standard errors will be smaller.

However, when applied to logistic regression, oversampling will result in an inconsistent estimate of the intercept (constant) term, although all other estimates will be consistent. Hence, if we do not de-bias (adjust) the intercept, the predicted probabilities will be too large, reflecting the artificial response rate of in the over-sampled training data.

In order to de-bias the scale of the predicted response (in this example: churn) in the validation sample we need to supply an *offset variable* to the logistic regression model. An offset is a known number that is added to the right-hand side of the regression when estimating the model, and adding the offset will correspondingly change the estimate of the intercept. The offset takes the form:

Here, is the average response rate in the training sample and is the average response rate in the validation sample. Note that the offset is positive (given that ), so that including the offset term when estimating the model accounts for the fact that the training sample has a higher share of relative to the validation sample.

Conversely, when we predict the response rate in the validation sample, we set the offset variable to 0.

Why does this work? — Conceptually, logistic regression is a regression model for the log-odds of the response (outcome) probability,

When we add the offset variable to the right hand side of the regression model the estimation algorithm will “incorporate” the offset in the intercept, . The effect of setting the offset to 0 (when applying the model to the validation sample) is equivalent to subtracting the offset from the intercept. Subtracting the offset amounts to:

1. Subtracting , the log-odds of the artificial response rate in the training sample, and
2. Adding , the log-odds in the validation sample that reflects the true log-odds in the data.

This process de-biases the predicted response, i.e. restores the correct response level in the validation sample.

Note: Never use over-sampling to create the validation sample, otherwise the offset variable approach will not work.

*Create an offset\_var variable and add it to the data set. Then re-estimate the logistic regression. To tell glm that you want to use offset\_var, you need to use a formula of the form:*

y ~ offset(offset\_var) +

*Where* you place offset() on the right-hand side of the formula is irrelevant.

*Before predicting the response rate in the validation sample set the offset to 0. Then, when you invoke the predict function, supply the data with the offset set to 0 using the newdata option.*

*Compare the average predicted response with the average observed response rate in the validation sample.*

mean(calibration\_data$churn)

[1] 0.4981382

data\_Y1 = cell2cell\_DT[churn == 1]  
data\_Y0 = cell2cell\_DT[churn == 0]  
  
#TODO: this is making the training data include part of the validation sample.   
set.seed(11111) # Set seed for reproducibility  
sample\_size = min(nrow(data\_Y1), nrow(data\_Y0))  
training\_data = rbind(  
 data\_Y1[sample(.N, sample\_size)],  
 data\_Y0[sample(.N, sample\_size)]  
)  
  
training\_data = calibration\_data  
  
validation\_data = cell2cell\_DT[calibrat == 0]  
  
  
p\_t = mean(training\_data$churn) # Oversampled rate  
p\_v = mean(validation\_data$churn) # True rate in the validation sample  
  
  
offset\_value = (log(p\_t) - log(1 - p\_t)) - (log(p\_v) - log(1 - p\_v))  
training\_data[, offset\_var := offset\_value]  
  
  
predictors = setdiff(names(training\_data), c("churn", "customer", "calibrat", "offset\_var"))  
  
  
formula = as.formula(paste("churn ~ offset(offset\_var) +", paste(predictors, collapse = " + ")))  
  
  
fit = glm(formula, data = training\_data, family = binomial(link = "logit"))  
  
  
results\_DT = as.data.table(tidy(fit))  
kable(results\_DT, digits = 5)

| term | estimate | std.error | statistic | p.value |
| --- | --- | --- | --- | --- |
| (Intercept) | -3.77119 | 0.09527 | -39.58269 | 0.00000 |
| revenue | 0.00196 | 0.00080 | 2.46016 | 0.01389 |
| mou | -0.00028 | 0.00005 | -5.65709 | 0.00000 |
| recchrge | -0.00312 | 0.00089 | -3.51332 | 0.00044 |
| directas | -0.00120 | 0.00594 | -0.20135 | 0.84042 |
| overage | 0.00076 | 0.00028 | 2.71118 | 0.00670 |
| roam | 0.00709 | 0.00206 | 3.43641 | 0.00059 |
| changem | -0.00049 | 0.00005 | -9.19411 | 0.00000 |
| changer | 0.00230 | 0.00037 | 6.24656 | 0.00000 |
| dropvce | 0.01134 | 0.00725 | 1.56304 | 0.11804 |
| blckvce | 0.00640 | 0.00716 | 0.89452 | 0.37104 |
| unansvce | 0.00092 | 0.00045 | 2.05780 | 0.03961 |
| custcare | -0.00595 | 0.00255 | -2.33129 | 0.01974 |
| threeway | -0.03029 | 0.01125 | -2.69107 | 0.00712 |
| mourec | 0.00013 | 0.00013 | 1.01771 | 0.30882 |
| outcalls | 0.00112 | 0.00059 | 1.89446 | 0.05816 |
| incalls | -0.00311 | 0.00106 | -2.93705 | 0.00331 |
| peakvce | -0.00067 | 0.00022 | -3.05791 | 0.00223 |
| opeakvce | -0.00021 | 0.00027 | -0.78284 | 0.43372 |
| dropblk | -0.00311 | 0.00704 | -0.44249 | 0.65814 |
| callfwdv | -0.00264 | 0.02315 | -0.11414 | 0.90913 |
| callwait | 0.00208 | 0.00314 | 0.66379 | 0.50682 |
| months | -0.02128 | 0.00200 | -10.65184 | 0.00000 |
| uniqsubs | 0.18438 | 0.01999 | 9.22522 | 0.00000 |
| actvsubs | -0.20572 | 0.02791 | -7.37167 | 0.00000 |
| phones | 0.04866 | 0.01817 | 2.67837 | 0.00740 |
| models | 0.01380 | 0.02787 | 0.49501 | 0.62060 |
| eqpdays | 0.00144 | 0.00007 | 19.30918 | 0.00000 |
| age1 | -0.00330 | 0.00087 | -3.78708 | 0.00015 |
| age2 | -0.00117 | 0.00068 | -1.71810 | 0.08578 |
| children | 0.09455 | 0.02815 | 3.35918 | 0.00078 |
| credita | -0.17807 | 0.03550 | -5.01584 | 0.00000 |
| creditaa | -0.36265 | 0.03458 | -10.48750 | 0.00000 |
| prizmrur | 0.06649 | 0.04956 | 1.34154 | 0.17975 |
| prizmub | -0.03963 | 0.02441 | -1.62389 | 0.10440 |
| prizmtwn | 0.04622 | 0.03145 | 1.46985 | 0.14160 |
| refurb | 0.23404 | 0.03196 | 7.32343 | 0.00000 |
| webcap | -0.15613 | 0.03756 | -4.15660 | 0.00003 |
| truck | 0.02689 | 0.03600 | 0.74698 | 0.45508 |
| rv | 0.01186 | 0.04801 | 0.24700 | 0.80491 |
| occprof | -0.01987 | 0.03250 | -0.61131 | 0.54100 |
| occcler | 0.03949 | 0.07491 | 0.52720 | 0.59805 |
| occcrft | -0.02013 | 0.06290 | -0.32009 | 0.74890 |
| occstud | 0.11997 | 0.12187 | 0.98441 | 0.32492 |
| occhmkr | 0.25587 | 0.19008 | 1.34611 | 0.17827 |
| occret | -0.03993 | 0.09055 | -0.44096 | 0.65924 |
| occself | -0.07057 | 0.08059 | -0.87566 | 0.38122 |
| ownrent | 0.00255 | 0.04272 | 0.05978 | 0.95233 |
| marryun | 0.10881 | 0.03403 | 3.19771 | 0.00139 |
| marryyes | 0.05570 | 0.03249 | 1.71446 | 0.08644 |
| mailord | 0.00077 | 0.08565 | 0.00897 | 0.99284 |
| mailres | -0.12971 | 0.08604 | -1.50754 | 0.13167 |
| mailflag | -0.04818 | 0.08445 | -0.57055 | 0.56830 |
| travel | -0.00053 | 0.04732 | -0.01124 | 0.99103 |
| pcown | 0.03418 | 0.03096 | 1.10391 | 0.26963 |
| creditcd | 0.04202 | 0.04371 | 0.96123 | 0.33644 |
| retcalls | 0.01203 | 0.18367 | 0.06552 | 0.94776 |
| retaccpt | -0.12786 | 0.10764 | -1.18781 | 0.23491 |
| newcelly | -0.07053 | 0.02727 | -2.58607 | 0.00971 |
| newcelln | -0.00508 | 0.03153 | -0.16127 | 0.87188 |
| refer | -0.05003 | 0.04214 | -1.18703 | 0.23521 |
| incmiss | -0.09151 | 0.06006 | -1.52357 | 0.12761 |
| income | -0.01324 | 0.00603 | -2.19481 | 0.02818 |
| mcycle | 0.12231 | 0.08898 | 1.37445 | 0.16930 |
| setprcm | -0.09632 | 0.04051 | -2.37748 | 0.01743 |
| setprc | 0.00062 | 0.00028 | 2.19418 | 0.02822 |
| retcall | 0.79370 | 0.19458 | 4.07899 | 0.00005 |

validation\_data[, offset\_var := 0]  
  
  
predicted\_response = predict(fit, newdata = validation\_data, type = "response")  
  
  
mean\_predicted\_response = mean(predicted\_response)  
mean\_observed\_response = mean(validation\_data$churn)  
  
  
list(mean\_predicted = mean\_predicted\_response, mean\_observed = mean\_observed\_response)

$mean\_predicted  
[1] 0.01943244  
  
$mean\_observed  
[1] 0.01929663

# Predictive power: Lift

We evaluate the predictive fit of the logistic regression model using a lift table and lift chart. To develop reusable code, we develop a function that returns a lift table. The function (call it liftTable) will need to take the following inputs:

* Predicted outcome or score
* Observed outcome
* Number of segments to be created based on the score

liftTable will return a data.table that contains:

* An index (score\_group) for each segment that was created based on the score
* The average score value (predicted outcome) in the score\_group
* The average observed outcome in the score\_group
* The lift factor

To code the liftTable command, I recommend to use the cut\_number function in the ggplot2 package. cut\_number takes a variable x and creates n groups with an approximately equal number of observations in each group. Observations are assigned to the groups based on their ranking along the variable x. The format is:

cut\_number(x, n = <no. of groups>)

To illustrate, we draw 10,000 random numbers from a uniform distribution on . cut\_number assigns each number to one of five (because we set n = 5) groups.

set.seed(123)  
DT = data.table(x = runif(10000, min = 0, max = 5))  
DT[, group := cut\_number(x, n = 5)]  
DT[, group\_no := as.integer(group)]

head(DT)

x group group\_no  
 <num> <fctr> <int>  
1: 1.4378876 (1,2.01] 2  
2: 3.9415257 (2.98,3.98] 4  
3: 2.0448846 (2.01,2.98] 3  
4: 4.4150870 (3.98,5] 5  
5: 4.7023364 (3.98,5] 5  
6: 0.2277825 [0.000327,1] 1

table(DT$group)

[0.000327,1] (1,2.01] (2.01,2.98] (2.98,3.98] (3.98,5]   
 2000 2000 2000 2000 2000

As expected, because x is uniformly distributed on , the five groups created by cut\_number correspond almost exactly to a interval (), and each of these intervals contains exactly 20 percent of all observations based on the rank of the x values. The group variable that we created is a factor that we converted to an integer score.

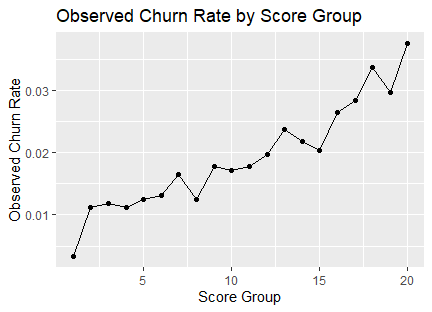
*Calculate a lift table for 20 segments. Inspect the lift table. Then provide two charts. First, plot the score\_group segments on the x-axis versus the observed churn rate on the y-axis. Second, plot the segments versus the lift factor, and add a horizontal line at . How to do this in ggplot2 is explained in the ggplot2 guide (look for the yintercept option).*

liftTable <- function(predicted, observed, n\_segments = 20) {  
 DT <- data.table(predicted = predicted, observed = observed)  
 DT[, score\_group := cut\_number(predicted, n = n\_segments)]  
 DT[, score\_group\_no := as.integer(score\_group)]  
   
 lift\_DT <- DT[, .(  
 avg\_score = mean(predicted),  
 avg\_observed = mean(observed),  
 lift\_factor = mean(observed) / mean(DT$observed)  
 ), by = score\_group\_no]  
   
 lift\_DT  
}

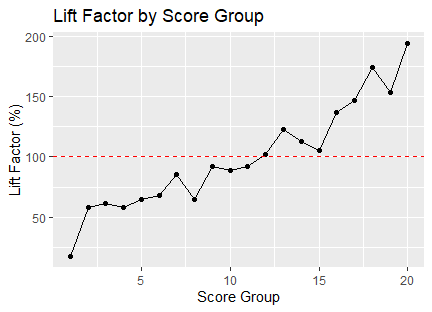
lift\_DT <- liftTable(predicted = predicted\_response, observed = validation\_data$churn, n\_segments = 20)  
kable(lift\_DT, digits = 3)

| score\_group\_no | avg\_score | avg\_observed | lift\_factor |
| --- | --- | --- | --- |
| 1 | 0.007 | 0.003 | 0.171 |
| 2 | 0.010 | 0.011 | 0.580 |
| 3 | 0.011 | 0.012 | 0.614 |
| 9 | 0.017 | 0.018 | 0.921 |
| 10 | 0.017 | 0.017 | 0.888 |
| 5 | 0.013 | 0.013 | 0.649 |
| 20 | 0.048 | 0.038 | 1.945 |
| 7 | 0.015 | 0.016 | 0.853 |
| 17 | 0.025 | 0.028 | 1.468 |
| 16 | 0.024 | 0.026 | 1.366 |
| 19 | 0.032 | 0.030 | 1.536 |
| 4 | 0.012 | 0.011 | 0.580 |
| 14 | 0.021 | 0.022 | 1.127 |
| 6 | 0.014 | 0.013 | 0.682 |
| 12 | 0.019 | 0.020 | 1.023 |
| 8 | 0.016 | 0.013 | 0.649 |
| 13 | 0.020 | 0.024 | 1.229 |
| 11 | 0.018 | 0.018 | 0.922 |
| 15 | 0.022 | 0.020 | 1.058 |
| 18 | 0.028 | 0.034 | 1.740 |

ggplot(lift\_DT, aes(x = score\_group\_no, y = avg\_observed)) +  
 geom\_line() +  
 geom\_point() +  
 labs(  
 x = "Score Group",  
 y = "Observed Churn Rate",  
 title = "Observed Churn Rate by Score Group"  
 )



ggplot(lift\_DT, aes(x = score\_group\_no, y = lift\_factor \* 100)) +  
 geom\_line() +  
 geom\_point() +  
 geom\_hline(yintercept = 100, linetype = "dashed", color = "red") +  
 labs(  
 x = "Score Group",  
 y = "Lift Factor (%)",  
 title = "Lift Factor by Score Group"  
 )



# Why do customers churn? — Effect sizes

We would like to understand *why* customers churn, which can help us to propose incentives to prevent customer churn

*To this end, construct a table that contains comparable effect sizes (changes in the churn probability) for all independent variables, as we discussed in class.*

Here are a few more details on the steps needed to create this table:

1. Because logistic regression coefficients are not directly interpretable, we estimate a linear probability model of customer churn. In a linear probability model we regress the output on all the customer features. The estimated coefficients can be interpreted as differences in for a one-unit difference in one of the features, . Note: **The *offset variable* should not be included in the linear probability model as it is specific to logistic regression.**
2. Note that our analysis is based on a *comparison* of the effect sizes of the different variables. However, because the variables have different scales, the effect sizes are not directly comparable. For example, revenue (mean monthly revenue) and mou (mean monthly minutes use) have different means and standard deviations, and hence the effects of increasing revenue and mou by one unit on the churn probabilities are not comparable without taking the scale differences into account.
3. To solve this problem we **standardize** the independent variables in the data. To standardize, we divide the values of each independent variable by its standard deviation, except if the variable is a 0/1 dummy. Once standardized, all variables except the dummies will have a standard deviation of 1, and a one unit difference corresponds to a one standard deviation difference in the original, non-standardized variable. Here’s a function, standardize\_columns, that takes a column x as input and returns the standardized values of the column:

standardize\_columns <- function(x) {  
   
 # Check if the column is a dummy variable  
 elements = unique(x)  
 if (length(elements) == 2L & all(elements %in% c(0L,1L))) {  
 is\_dummy = TRUE  
 } else {  
 is\_dummy = FALSE  
 }  
   
 # If not a dummy, divide values in x by its standard deviation  
 if (is\_dummy == FALSE) x = x/sd(x, na.rm = TRUE)  
   
 return(x)  
}

The first part of the function checks that the input x has exactly two elements and that these elements are the integers 0 and 1. Note that in R, numbers are represented as floating point numbers by default. However, adding L after the numbers tells R to represent the number as an integer.

class(1)

[1] "numeric"

class(1L)

[1] "integer"

In order to standardize all independent variables in the training data, you can use:

DT\_lin\_prob = cell2cell\_DT[calibrat == 1]  
  
# Create a vector that contains the names of all inputs (covariates)  
all\_columns = names(DT\_lin\_prob)  
input\_columns = all\_columns[-c(1:3, length(all\_columns))]  
  
# Standardize all input columns  
DT\_lin\_prob[, (input\_columns) := lapply(.SD, standardize\_columns), .SDcols = input\_columns]

1. In order to create a table that captures the linear probability model estimates, use the tidy function. Add a column, e.g. effect\_size, that scales the estimates by the factor

* This scales the effect sizes to the correct magnitude of the churn probabilities in the validation sample and puts the effects on a 0-100% scale. Sort the variables according to the magnitude of the effect sizes, and print the results table using kable.

#Q4  
# Fit Linear Probability Model  
predictors = setdiff(names(DT\_lin\_prob), c("churn", "customer", "calibrat"))  
formula = as.formula(paste("churn ~", paste(predictors, collapse = " + ")))  
  
# Fit the logistic regression model  
fit = glm(formula, data = DT\_lin\_prob, family = binomial(link = "logit"))  
  
# 3. Calculate effect sizes  
# Extract coefficients using tidy() from the broom package  
results\_DT = as.data.table(tidy(fit))  
  
# Calculate effect size using scaling formula  
p\_t = mean(training\_data$churn) # Churn rate in the training data  
p\_v = mean(validation\_data$churn) # Churn rate in the validation data  
results\_DT[, effect\_size := 100 \* estimate \* (p\_v / p\_t)]  
  
# Sort results by effect size and display the table  
results\_DT = results\_DT[order(abs(effect\_size), decreasing = TRUE)]  
kable(results\_DT, digits = 5)

| term | estimate | std.error | statistic | p.value | effect\_size |
| --- | --- | --- | --- | --- | --- |
| retcall | 0.79370 | 0.19458 | 4.07899 | 0.00005 | 3.07459 |
| eqpdays | 0.36792 | 0.01905 | 19.30918 | 0.00000 | 1.42522 |
| creditaa | -0.36265 | 0.03458 | -10.48750 | 0.00000 | -1.40481 |
| occhmkr | 0.25587 | 0.19008 | 1.34611 | 0.17827 | 0.99117 |
| uniqsubs | 0.24313 | 0.02635 | 9.22522 | 0.00000 | 0.94182 |
| refurb | 0.23404 | 0.03196 | 7.32343 | 0.00000 | 0.90662 |
| months | -0.20352 | 0.01911 | -10.65184 | 0.00000 | -0.78838 |
| credita | -0.17807 | 0.03550 | -5.01584 | 0.00000 | -0.68982 |
| webcap | -0.15613 | 0.03756 | -4.15660 | 0.00003 | -0.60480 |
| (Intercept) | 0.14970 | 0.09527 | 1.57127 | 0.11612 | 0.57990 |
| mou | -0.14703 | 0.02599 | -5.65709 | 0.00000 | -0.56955 |
| actvsubs | -0.13941 | 0.01891 | -7.37167 | 0.00000 | -0.54003 |
| mailres | -0.12971 | 0.08604 | -1.50754 | 0.13167 | -0.50248 |
| changem | -0.12706 | 0.01382 | -9.19411 | 0.00000 | -0.49222 |
| mcycle | 0.12231 | 0.08898 | 1.37445 | 0.16930 | 0.47378 |
| occstud | 0.11997 | 0.12187 | 0.98441 | 0.32492 | 0.46475 |
| marryun | 0.10881 | 0.03403 | 3.19771 | 0.00139 | 0.42149 |
| dropvce | 0.10047 | 0.06428 | 1.56304 | 0.11804 | 0.38921 |
| setprcm | -0.09632 | 0.04051 | -2.37748 | 0.01743 | -0.37312 |
| children | 0.09455 | 0.02815 | 3.35918 | 0.00078 | 0.36627 |
| incmiss | -0.09151 | 0.06006 | -1.52357 | 0.12761 | -0.35449 |
| changer | 0.09107 | 0.01458 | 6.24656 | 0.00000 | 0.35279 |
| revenue | 0.08609 | 0.03499 | 2.46016 | 0.01389 | 0.33347 |
| overage | 0.07403 | 0.02731 | 2.71118 | 0.00670 | 0.28679 |
| recchrge | -0.07379 | 0.02100 | -3.51332 | 0.00044 | -0.28586 |
| age1 | -0.07287 | 0.01924 | -3.78708 | 0.00015 | -0.28227 |
| occself | -0.07057 | 0.08059 | -0.87566 | 0.38122 | -0.27338 |
| newcelly | -0.07053 | 0.02727 | -2.58607 | 0.00971 | -0.27323 |
| peakvce | -0.06862 | 0.02244 | -3.05791 | 0.00223 | -0.26581 |
| blckvce | 0.06732 | 0.07526 | 0.89452 | 0.37104 | 0.26077 |
| prizmrur | 0.06649 | 0.04956 | 1.34154 | 0.17975 | 0.25758 |
| phones | 0.06422 | 0.02398 | 2.67837 | 0.00740 | 0.24878 |
| roam | 0.05842 | 0.01700 | 3.43641 | 0.00059 | 0.22631 |
| marryyes | 0.05570 | 0.03249 | 1.71446 | 0.08644 | 0.21576 |
| incalls | -0.05145 | 0.01752 | -2.93705 | 0.00331 | -0.19929 |
| mailflag | -0.04818 | 0.08445 | -0.57055 | 0.56830 | -0.18664 |
| dropblk | -0.04723 | 0.10673 | -0.44249 | 0.65814 | -0.18294 |
| prizmtwn | 0.04622 | 0.03145 | 1.46985 | 0.14160 | 0.17905 |
| creditcd | 0.04202 | 0.04371 | 0.96123 | 0.33644 | 0.16277 |
| income | -0.04128 | 0.01881 | -2.19481 | 0.02818 | -0.15992 |
| occret | -0.03993 | 0.09055 | -0.44096 | 0.65924 | -0.15467 |
| prizmub | -0.03963 | 0.02441 | -1.62389 | 0.10440 | -0.15353 |
| occcler | 0.03949 | 0.07491 | 0.52720 | 0.59805 | 0.15298 |
| outcalls | 0.03869 | 0.02042 | 1.89446 | 0.05816 | 0.14987 |
| unansvce | 0.03532 | 0.01716 | 2.05780 | 0.03961 | 0.13682 |
| setprc | 0.03507 | 0.01598 | 2.19418 | 0.02822 | 0.13586 |
| pcown | 0.03418 | 0.03096 | 1.10391 | 0.26963 | 0.13239 |
| threeway | -0.03264 | 0.01213 | -2.69107 | 0.00712 | -0.12645 |
| custcare | -0.03100 | 0.01330 | -2.33129 | 0.01974 | -0.12008 |
| age2 | -0.02782 | 0.01619 | -1.71810 | 0.08578 | -0.10778 |
| truck | 0.02689 | 0.03600 | 0.74698 | 0.45508 | 0.10417 |
| mourec | 0.02209 | 0.02171 | 1.01771 | 0.30882 | 0.08557 |
| occcrft | -0.02013 | 0.06290 | -0.32009 | 0.74890 | -0.07799 |
| occprof | -0.01987 | 0.03250 | -0.61131 | 0.54100 | -0.07697 |
| retaccpt | -0.01934 | 0.01628 | -1.18781 | 0.23491 | -0.07490 |
| opeakvce | -0.01910 | 0.02440 | -0.78284 | 0.43372 | -0.07400 |
| refer | -0.01244 | 0.01048 | -1.18703 | 0.23521 | -0.04821 |
| models | 0.01232 | 0.02489 | 0.49501 | 0.62060 | 0.04774 |
| rv | 0.01186 | 0.04801 | 0.24700 | 0.80491 | 0.04594 |
| callwait | 0.01121 | 0.01689 | 0.66379 | 0.50682 | 0.04342 |
| newcelln | -0.00508 | 0.03153 | -0.16127 | 0.87188 | -0.01970 |
| retcalls | 0.00263 | 0.04015 | 0.06552 | 0.94776 | 0.01019 |
| ownrent | 0.00255 | 0.04272 | 0.05978 | 0.95233 | 0.00989 |
| directas | -0.00246 | 0.01223 | -0.20135 | 0.84042 | -0.00954 |
| callfwdv | -0.00118 | 0.01033 | -0.11414 | 0.90913 | -0.00457 |
| mailord | 0.00077 | 0.08565 | 0.00897 | 0.99284 | 0.00298 |
| travel | -0.00053 | 0.04732 | -0.01124 | 0.99103 | -0.00206 |

1. Inspect the results. Identify some factors that are strongly associated with churn. If actionable, propose an incentive that can be targeted to the customers to prevent churn.
2. Retcall has the highest association with churn with an effect\_size value of ~3.06, which is not surprising given the nature of the variable being a call to the retention team themselves. Thus, for customers with high retcall values, especially high-value customers, it is critical to first identify the trigger(s) for contacting the retention team. Following this, we can offer personalized outreach with incentives like offers, loyalty rewards, and feedback loops perhaps, to mitigate churn. Another method to proactively manage churn rates is to employ predictive analytics to identify at-risk customers before they reach out to the retention team, and proactively offer solutions tailored to their anticipated needs.
3. Eqpdays also exhibits a substantial impact on churn, with an effect\_size value of ~1.42. This indicates that as equipment ages, it may become less efficient or more prone to issues, leading to increased customer dissatisfaction. To curb churn resulting from this, the eqpday variable could be utilized to identify customers with older equipment and target them with offers for upgrades or replacements. This can be framed as a loyalty perk or a preventive measure to ensure they continue to have a positive experience. Incentives in the form of discounted upgrades would also be feasible, for instance, by providing special financing options (for a limited time, perhaps) or customers due for an upgrade after a certain period. Communication in the form of maintenance tips or automated alerts based on the value of eqpdays could also prove useful in product life extension and reducing dissatisfaction from equipment issues.
4. The variable creditaa, reflecting a credit AA rating with an effect size of -1.39958 indicates that having a AA credit rating is a strong protective factor against customer churn. Customers with high credit ratings are likely more financially stable and may perceive less risk in continuing service or making long-term commitments. AA credit rating may also correlate with higher levels of customer satisfaction and trust in a good/service, suggesting these customers are more content or engaged with their current arrangements. Targeted incentives for this factor could include, firstly, offering better terms for upgrades or renewals to customers with higher credit ratings, reflecting the lower risk they pose in terms of payment delinquency. For customers interested in expensive services or products, it may be beneficial to provide flexible financing options that leverage their high credit rating to offer better interest rates or payment terms. In terms of expanding customer base (given the negative association of creditaa with churn), developing marketing campaigns that target customers with high credit ratings, emphasizing the stability and trustworthiness of the brand which aligns with their financial behavior, could help create a reliable and more stable customer target.

# Economics of churn management

Next, we would like to predict the value of a proposed churn management program in order to assess the maximum amount that we would spend to prevent a customer from churning for one year.

*Perform this prediction*, under the following assumptions:

1. We consider a planning horizon of 4 years (the current year and three additional years), and an annual discount rate of 8 percent.
2. Predict the churn management value for 20 groups, but keep in mind that it is good practice to make sure the code works for an arbitrary number of groups in case we wish to modify that in the future. Predict the program value for these 20 customer segments based on the predicted churn rate. Note that we create these segments based on the validation sample data. We predict current and future customer profits at the year-level. Hence, we also need to convert both the monthly churn rate and the revenue data to the year-level.
3. Assume that the churn management program has a success probability gamma () and compare the results for and .

Hint: It is easy to make little mistakes in the lifetime value predictions. Hence, be very clear about what your code is supposed to achieve, and check that every step is correct.

# Goal: calculate LTV without churn program and LTV with churn program  
# Step 1: Create groups based on predicted\_churn ranking.  
# Step 2: Create function calculates LTV without churn program (however, allow the function to take in a churn adjustment that we can plug gamma into) -> Sum for each year retention\_rate^t \* customer\_profit divided by (1+.08)^t  
# Step 3: Calculate this for all groups and store the values  
# Step 4: Calculate this for all groups with churn adjustment of .25 and store the values  
# Step 5: Calculate this for all groups with churn adjustment of .5 and store the values  
  
# Key things to note: churn is monthly and so is revenue. Account for this fact.  
  
  
create\_n\_groups <- function(predicted, observed, n\_segments) {  
 DT <- data.table(predicted\_churn = predicted, observed\_churn = observed$churn, observed\_revenue = observed$revenue)  
 DT[, score\_group := cut\_number(predicted, n = n\_segments)]  
 DT[, score\_group\_no := as.integer(score\_group)]  
   
 group\_DT <- DT[, .(  
 avg\_score = mean(predicted\_churn),  
 avg\_observed = 1 - (1 - mean(observed\_churn))^12,  
 avg\_annual\_revenue = 12 \* mean(observed\_revenue)  
 ), by = score\_group\_no]  
   
 return(group\_DT);  
}  
  
#Create groups based on predicted churn with data on predicted churn, observed churn, and annual revenue. Churn is adjusted for yearly churn.  
grouped\_data = create\_n\_groups(predicted = predicted\_response, observed = validation\_data, n\_segments=20)  
  
  
calculate\_LTV <- function(observed\_churn, churn\_program\_success, annual\_revenue, years, discount\_rate) {  
 # Key assumption: predicting average per person revenue while accounting for churn  
 total\_revenue = 0  
 adjusted\_churn\_rate = observed\_churn \* (1 - churn\_program\_success)  
  
 for(t in 1:years) {  
 retention\_rate = (1 - adjusted\_churn\_rate)^t  
 discounted\_cash\_flow = (annual\_revenue \* retention\_rate) / (1 + discount\_rate)^t  
 total\_revenue = total\_revenue + discounted\_cash\_flow  
 }  
  
 return(total\_revenue)  
}  
  
# Assuming years = 4, discount\_rate = 8%  
years <- 4  
discount\_rate <- 0.08  
success\_probabilities <- c(0.25, 0.5)  
  
# Calculate LTV for each group and each gamma  
grouped\_data[, LTV\_no\_program := calculate\_LTV(avg\_observed, 0, avg\_annual\_revenue, years, discount\_rate)]  
for (gamma in success\_probabilities) {  
 column\_name <- paste("LTV\_with\_gamma", gamma, sep = "\_")  
 grouped\_data[, (column\_name) := calculate\_LTV(avg\_observed, gamma, avg\_annual\_revenue, years, discount\_rate)]  
}

grouped\_data = setorder(grouped\_data, score\_group\_no)  
grouped\_data

# Summarize your main results

*Please organize your main results along the four questions posed in the overview.*