DATA 621 Final Project

Chi Pong, Euclid Zhang, Jie Zou, Joseph Connolly, LeTicia Cancel

4/19/2022

```
#setwd("/Users/dpong/Data 621/Final_Project/Datasets")
# setwd("~/Library/CloudStorage/OneDrive-CityUniversityofNewYork/621/final_churn_modeling")
df <- read.csv("sparkify-medium.csv", stringsAsFactors = FALSE, row.names=1)
```

1. Data Source

In this analysis, we will utilize the Sparkify data set created by Udacity, an educational organization. Sparkify is a fictional company providing music streaming service.

The full data set is too large to run on a regular personal computer so we will focus on a subset of the data which consists of 543,705 user activity records of 448 users from 10/1/2018 to 12/1/2018. Each record contains the following information:

- ts: Time Stamp (in milliseconds) of the user activity, range from 1538352000000 (the beginning of 10/1/2018) to 1543622400000 (the end of 12/1/2018) Greenwich Mean Time (GMT).
- userId: The user's unique ID. NA if for guest activities.
- sessionId: Identifier of a connection session. The Id is not unique and may be used by another user at a different time.
- page: The page corresponding to a user's action. For example, *NextSong* indicates a user starts listening to a new song and *Roll Advert* indicates an advertisement is loaded.
- auth: Indicates the user's status (logged in, logged out, guest, canceled).
- method: method of the user's web request, PUT or GET.
- status: status of a web request. For example, 404 indicates the requested resource is not found.
- level: the user's account level at the time of the activity. 2 levels: paid or free.
- itemInSession: the number of cumulative activities during a web session.
- location: geometric location (city and state) of the user.
- userAgent: user agent of the user, which includes the type of device, operating system and browser version that the user is using.
- lastName: last name of the registered user
- firstName: first name of the registered user
- registration: the time stamp of the time that a user submitted his/her registration
- gender: the gender of the user
- artist: the artist of the song that the user is listening. Different artists may have songs with the same title / name.
- song: the title / name of the song.
- length: the length of the song in seconds

Since this data is fictional, there are several defects of the data we found and need to be aware before approaching our analysis.

1. We only have data from 10/1/2018 to 12/1/2018. Since looking at the events (or activities) before a user registered might be a logical step in doing the analysis or even building a predictive model, we are interested in the that piece of event data. Unfortunately, almost all users in the data set registered before 10/1/2018. So it's understood that there are quite a bit of a limitation as to what we can uncover as insights from this dataset.
2. The user agent is static for each user, which doesn't necessarily reflect the reality of real-world data where people use different devices, browsers, and OSs.
3. The location of the users are static. Again, it doesn't match the reality.
4. By examining on the time stamps of the activities and the length of the songs, we found that there is no partially listened songs.
5. There are "add to Playlist" activities in the record, but there is no "remove from Playlist" activities. The two activities may help the company identify if a user like the service but the latter is missing here.
6. Only users who canceled the service are considered as churned. Free users with no recent activities are not considered as churned.

In this analysis, we will clean up and aggregate the user activities based on the user ID. We will then perform feature engineering to build the features / variables for our model.

2. Data Clean Up and Aggregation

The time of registration for the records of a few users are incorrect (the time of registration is after the user's first log in). Correct the time of registration using the "Submit Registration" page and the session ID.

```
regist_df <- filter(df,df$page=="Submit Registration")

for (i in c(1:nrow(regist_df))) {
  temp_df <- df %>%
                filter(sessionId==regist_df$sessionId[i]) %>%
                filter(!is.na(userId)) %>%
                mutate(delta=abs(ts-regist_df$ts[i])) %>%
                arrange(delta,desc=FALSE)

  df[!is.na(df$userId) & df$userId==temp_df$userId[1],"registration"] <- regist_df$ts[i]
}
```

Filter out the guest records (the ones without a userId)

```
df <- filter(df,!is.na(userId))
```

Simplify the user Agent to represent the type of device that the user is using.

```
df$userAgent[str_detect(df$userAgent,"Macintosh")] <- "Macintosh"
df$userAgent[str_detect(df$userAgent,"iPad")] <- "iPad"
df$userAgent[str_detect(df$userAgent,"iPhone")] <- "iPhone"
df$userAgent[str_detect(df$userAgent,"Windows")] <- "Windows"
df$userAgent[str_detect(df$userAgent,"Linux")] <- "Linux"
```

Convert some categorical variables in to factors.

```
factor_columns <- c("page","auth","method","status","level","gender","userAgent")

df[factor_columns] <- lapply(df[factor_columns], factor)
```

Remove some variables that are not used in our analysis. For example, the method of the web request, the name of the user.

```
df$method <- NULL
df$status <- NULL
df$itemInSession <- NULL
df$location <- NULL
df$lastName <- NULL
df$firstName <- NULL
df$auth <- NULL
```

Create a new variable indicating whether it is a song that the user never listened before.

```
df <- arrange(df, ts,desc=FALSE)

df$user_song <- paste0(df$userId, df$artist, df$song)
temp <- df %>% group_by(user_song) %>% mutate(count=row_number())
df$new_song <- temp$count
temp <- NULL
df$user_song <- NULL
df$new_song[df$new_song > 1] <- 0
df$new_song[is.na(df$song)] <- NA
```

Aggregate the total number of records for each category of user activities (indicated by the *page* variable). For example, we would like to know the total number songs listened by a user, the total number of thumbs-ups by a user.

```
page_df <- df %>% group_by(userId) %>%
  count(page) %>%
  spread(page, n, fill = 0)

#Cancel column is identical to "Cancellation Confirmation" so it is removed
page_df$Cancel <- NULL

page_df[,2:ncol(page_df)] <- sapply(page_df[,2:ncol(page_df)], as.integer)
page_df$Total_Activities <- apply(page_df[,2:ncol(page_df)], 1, sum)

page_df
```

```
## # A tibble: 448 x 20
## # Groups:   userId [448]
##    userId About Add F~1 Add t~2 Cance~3 Downg~4 Error  Help  Home Logout NextS~5
##     <int> <int>   <int>   <int>   <int>   <int> <int> <int> <int>  <int>   <int>
##  1      2     0      14      24       0       7     1     7    33     13     779
##  2      3     0       0       0       0       0     0     0     0      0      24
##  3      4     0       7      10       0       0     2     0    21      7     421
##  4      5     0       1       6       0       0     0     1    13      5     219
##  5      6     1       9      35       0      10     2     2    69     18    1127
```

```
##  6      7     2      16       6       0       0     0     3    36     11     462
##  7      8     0       0       1       0       0     0     1     3      1      25
##  8      9     8      37      56       0      16     2    12    75     20    2191
##  9     10     0       3      14       1       1     1     3    15      4     360
## 10     11     1       6       5       1       1     0     2    14      3     194
## # ... with 438 more rows, 9 more variables: `Roll Advert` <int>,
## #   `Save Settings` <int>, Settings <int>, `Submit Downgrade` <int>,
## #   `Submit Upgrade` <int>, `Thumbs Down` <int>, `Thumbs Up` <int>,
## #   Upgrade <int>, Total_Activities <int>, and abbreviated variable names
## #   1: `Add Friend`, 2: `Add to Playlist`, 3: `Cancellation Confirmation`,
## #   4: Downgrade, 5: NextSong
```

Summarize additional user activities (for example, the total number of unique sessions) and user information (for example, the user's account level of the last activity).

```
user_df <- df %>% filter(!is.na(song)) %>%
  arrange(ts, desc=FALSE) %>%
  group_by(userId) %>%
  summarise(active_sessions=n_distinct(sessionId),
            new_songs_listened=sum(new_song),
            registration=first(registration),
            end_level=last(level),
            gender=first(gender),
            userAgent=first(userAgent))
user_df
```

```
## # A tibble: 448 x 7
##    userId active_sessions new_songs_listened registration end_l~1 gender userA~2
##     <int>           <int>              <dbl>        <dbl> <fct>   <fct>  <fct>
##  1      2              11                742      1.54e12 paid    F      Windows
##  2      3               1                 24      1.53e12 free    M      Windows
##  3      4              16                404      1.54e12 free    M      Windows
##  4      5               4                217      1.54e12 free    M      Windows
##  5      6              18               1035      1.52e12 paid    M      Macint~
##  6      7              15                448      1.54e12 free    M      Windows
##  7      8               2                 25      1.53e12 free    F      Windows
##  8      9              22               1956      1.54e12 free    M      Windows
##  9     10               1                350      1.54e12 paid    M      Macint~
## 10     11               6                192      1.53e12 paid    F      Windows
## # ... with 438 more rows, and abbreviated variable names 1: end_level,
## #   2: userAgent
```

Finding the beginning time stamp and ending time stamp of the observation period for each user. For users who registered before 10/1/2018, the observation starts from 10/1/2018 (1538352000000). Otherwise, the observation starts from the time of registration. For users who cancelled their service, the observation ends by the time stamp of the last activity. Otherwise, the observation ends at 12/1/2018.

```
df <- df %>% arrange(userId, desc=FALSE)

obs_df <- data.frame(userId=unique(df$userId))
obs_df$start <- ifelse(user_df$registration > 1538352000000, user_df$registration, 1538352000000)
obs_df$end <- 1543622400000
temp <- filter(df, page == "Cancellation Confirmation")
obs_df$end[obs_df$userId %in% temp$userId] <- temp$ts
```

```
#Calculate ad rolled per hour when account level = paid / free

roll_ad_df <- df %>% filter(page == "Roll Advert") %>%
  group_by(userId, level) %>%
  count() %>%
  spread(level, n, fill = 0)
```

```
#percentage of the song listening time with account level = paid

paid_time_df <- df %>% filter(!is.na(length)) %>%
  group_by(userId, level) %>%
  summarise(length = sum(length)) %>%
  spread(level, length, fill = 0)

paid_time_df$percent_paid <- paid_time_df$paid / (paid_time_df$free + paid_time_df$paid)
paid_time_df$free <- NULL
paid_time_df$paid <- NULL
```

Merging previously aggregated data into one dataframe.

```
prepared_df <- merge(obs_df, user_df, by=c("userId")) %>%
                arrange(userId)

prepared_df <- merge(prepared_df, page_df, by=c("userId")) %>%
                arrange(userId)

prepared_df <- merge(prepared_df, roll_ad_df, by=c("userId"), all.x=TRUE)

prepared_df <- merge(prepared_df, paid_time_df, by=c("userId"), all.x=TRUE)


prepared_df[is.na(prepared_df)] <- 0

names(prepared_df) <- str_replace_all(names(prepared_df), " ", "_")

#df <- merge(df, prepared_df[c("userId","start","end")], by=c("userId"))
```

3. Feature Engineering

Calculation of defined features that can be used as predictors for identifying users that are to churn.

```
train_df <- dplyr::select(prepared_df,userId,end_level,gender,userAgent)
train_df$churn <- as.factor(prepared_df$Cancellation_Confirmation)
```

```
prepared_df$duration_in_hours <- (prepared_df$end - prepared_df$start)/3600/1000

train_df$tot_act_phour <- prepared_df$Total_Activities/prepared_df$duration_in_hours
train_df$songs_phour <- prepared_df$NextSong/prepared_df$duration_in_hours
train_df$tot_tu_phour <- prepared_df$Thumbs_Up/prepared_df$duration_in_hours
train_df$tot_td_phour <- prepared_df$Thumbs_Down/prepared_df$duration_in_hours
```

```
train_df$frds_added_phour <- prepared_df$Add_Friend/prepared_df$duration_in_hours
train_df$tot_add2PL_phour <- prepared_df$Add_to_Playlist/prepared_df$duration_in_hours
train_df$HP_visits_phour <- prepared_df$Home/prepared_df$duration_in_hours
train_df$tot_errs_phour <- prepared_df$Error/prepared_df$duration_in_hours
train_df$upgrades_phour <- prepared_df$Submit_Upgrade/prepared_df$duration_in_hours
train_df$downgrades_phour <- prepared_df$Submit_Downgrade/prepared_df$duration_in_hours
train_df$setting_phour <- prepared_df$Settings/prepared_df$duration_in_hours
train_df$save_setting_phour <- prepared_df$Save_Settings/prepared_df$duration_in_hours
train_df$song_ratio <- prepared_df$NextSong / prepared_df$Total_Activities
train_df$new_songs_ratio <- prepared_df$new_songs_listened / prepared_df$NextSong
train_df$pos_negative_ratio <- (prepared_df$Thumbs_Up+1)/(prepared_df$Thumbs_Down+1)
train_df$ad_per_song <- prepared_df$Roll_Advert / prepared_df$NextSong

train_df$paid_ad_ph <- prepared_df$paid/prepared_df$duration_in_hours
train_df$free_ad_ph <- prepared_df$free/prepared_df$duration_in_hours

train_df$percent_paid <- prepared_df$percent_paid
```

```
# Calculation of user's average number of events per session
session_avg <- df %>%
                group_by(userId, sessionId) %>%
                summarise(events = n(), .groups = 'drop') %>%
                group_by(userId) %>%
                summarise(avg_events_per_session = mean(events))
```

```
# Calculation of user's average session duration

session_avg_length = df  %>%
                    group_by(userId, sessionId) %>%
                    arrange(ts, .by_group = TRUE) %>%
                    # filter(userId==3) %>%
                    summarise( session_begin_ts = min(ts),
                               session_end_ts = max(ts),
                               .groups = 'drop') %>%
                    group_by(userId) %>%
                    summarise( avg_session_duration = mean(session_end_ts-session_begin_ts))

#Convert from timestamp unit to hour
session_avg_length$avg_session_duration <- session_avg_length$avg_session_duration / 3600000
```

Incorporating all the newly defined business metrics into the main data.frame, i.e. *train_df*

Since the lengths of the observation period are different for some users (new registrations and terminated users), we will normalize the numbers of activities by dividing the total number of activities by the total number of hours in the observation period for each user.

The followings are the created features for our analysis

- end_level: The user's account level at the end of the observation period (paid account or free account)
- gender: The gender of the user
- userAgent: The type of device that the user is using (Windows, Iphone, Ipad, etc.)
- tot_act_phour: The total number of user activities / Total number of hours in the observation period

  - $tot\_act\_phour = \frac{Total\_Activities}{duration\_in\_hours}$

- songs_phour: The total number of songs listened / Total number of hours in the observation period
  - $songs\_phour = \frac{NextSong}{duration\_in\_hours}$
- tot_tu_phour: The total number of thumbs-ups / Total number of hours in the observation period
  - $tot\_tu\_phour = \frac{Thumbs\_Up}{duration\_in\_hours}$
- tot_td_phour: The total number of thumbs-downs / Total number of hours in the observation period
  - $tot\_td\_phour = \frac{Thumbs\_Down}{duration\_in\_hours}$
- frds_added_phour: The total number of friends added / Total number of hours in the observation period
  - $frds\_added\_phour = \frac{Add\_Friend}{duration\_in\_hours}$
- tot_add2PL_phour: The total number of songs added to the play list / Total number of hours in the observation period
  - $tot\_add2PL\_phour = \frac{Add\_to\_Playlist}{duration\_in\_hours}$
- HP_visits_phour: The total number of home page visit / Total number of hours in the observation period
  - $HP\_visits\_phour = \frac{Home}{duration\_in\_hours}$
- tot_errs_phour: The total number of error page encountered / Total number of hours in the observation period
  - $tot\_errs\_phour = \frac{Error}{duration\_in\_hours}$
- upgrades_phour: The total number of account level upgrading submitted / Total number of hours in the observation period
  - $upgrades\_phour = \frac{Submit\_Upgrade}{duration\_in\_hours}$
- downgrades_phour: The total number of account level downgrading submitted / Total number of hours in the observation period
  - $downgrades\_phour = \frac{Submit\_Downgrade}{duration\_in\_hours}$
- setting_phour: The total number of setting updates attempted / Total number of hours in the observation period
  - $settin\_phour = \frac{Settings}{duration\_in\_hours}$
- save_setting_phour: The total number of setting updates submitted / Total number of hours in the observation period
  - $save\_setting\_phour = \frac{Save\_Settings}{duration\_in\_hours}$
- song_ratio: The percentage of the activities that are NextSong (start listening to a song)
  - $song\_ratio = \frac{NextSong}{Total\_Activities}$

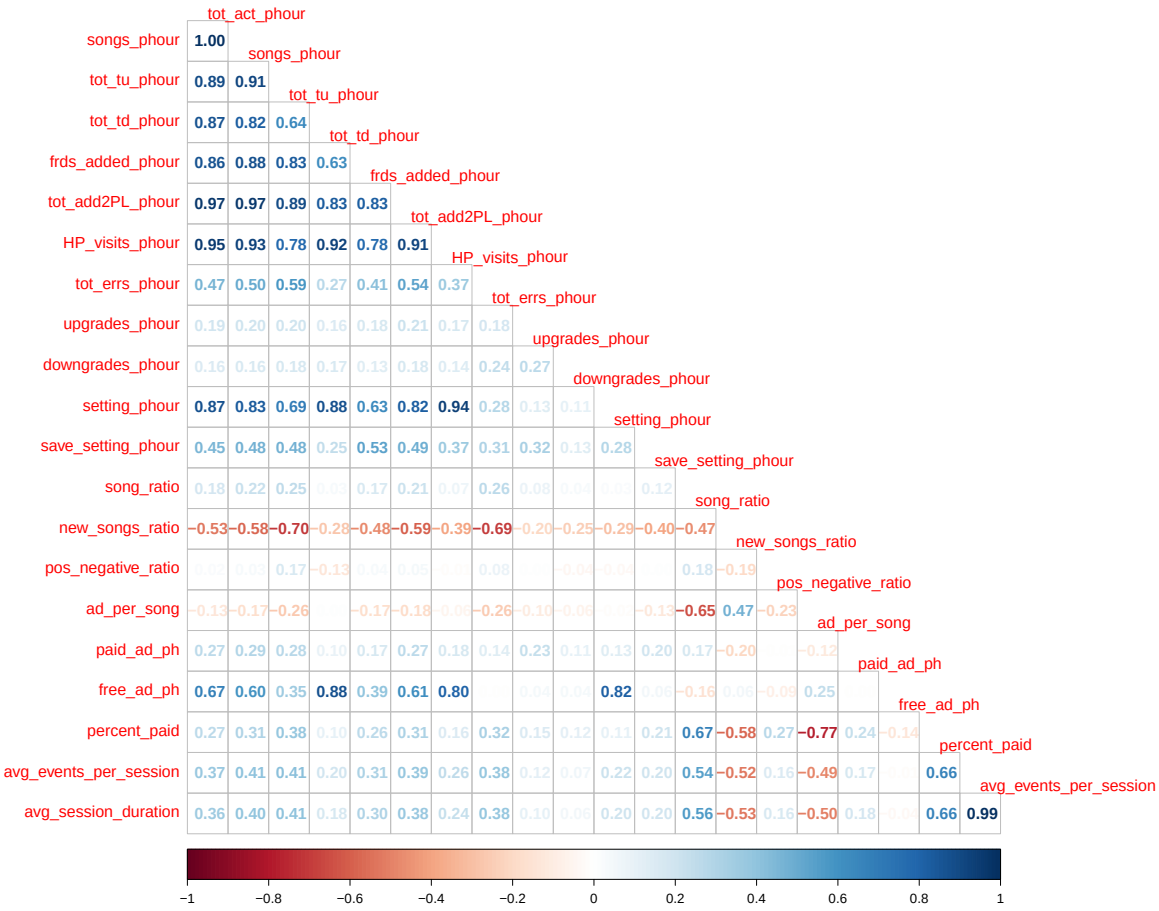
- new_songs_ratio: The percentage of the songs listened that the user has not listened before (which are the non-repeated songs)
  - $new\_songs\_ratio = \frac{new\_songs\_listened}{NextSong}$
- pos_negative_ratio: The ratio of the number of thumbs-ups to the number of thumbs-downs. To handle the issue of dividing by zero, the ratio is modified to (thumbs-ups + 1) / (thumbs-downs + 1).
  - $pos\_negative\_ratio = \frac{Thumbs\_Up+1}{Thumbs\_Down+1}$
- ad_per_song: The average number of advertisement per song listened
  - $ad\_per\_song = \frac{Roll\_Advert}{NextSong}$
- paid_ad_ph: The total number of advertisement listened when the user account level = paid / Total number of hours in the observation period
  - $paid\_ad\_ph = \frac{paid}{duration\_in\_hours}$
- free_ad_ph: The total number of advertisement listened when the user account level = free / Total number of hours in the observation period
  - $free\_ad\_ph = \frac{free}{duration\_in\_hours}$
- percent_paid: The percentage of the song-listening time that the user's account is in the paid level
- avg_events_per_session: The average number of activities per session
- avg_session_duration: The average duration per session in hours

4. Exploratory Data Analysis

This step is where we start to analyze the correlations between all these features.

4.1 Feature Correlations - Collinearity Check for Numeric Variables

We plugged in all the predictors, or independent variables, into this correlation matrix to visualize if there are any variables constitute multicollinearity.



To reduce multicollinearity of our data and reduce the complexity of our model, variables with correlation more than 0.8 are considered highly correlated and some variables are excluded as described below:

tot_act_phour is highly correlated with *songs_phour*, *tot_tu_phour*, *tot_td_phour*, *freds_added_phour*, *tot_add2PL_phour*, *HP_visits_phour*, and *setting_phour*. As song listening is main service here and we would like to know our service's user experience, we would keep *songs_phour* and drop the other variables.

HP_visits_phour is highly correlated with *free_ad_phour*. Likewise, setting_phour is highly positively correlated with HP_visits_phour. setting_phour was already dropped in the previous block. As we would like to know the user's option about the ads, we would keep *free_ad_phour* and drop *HP_visits_phour*

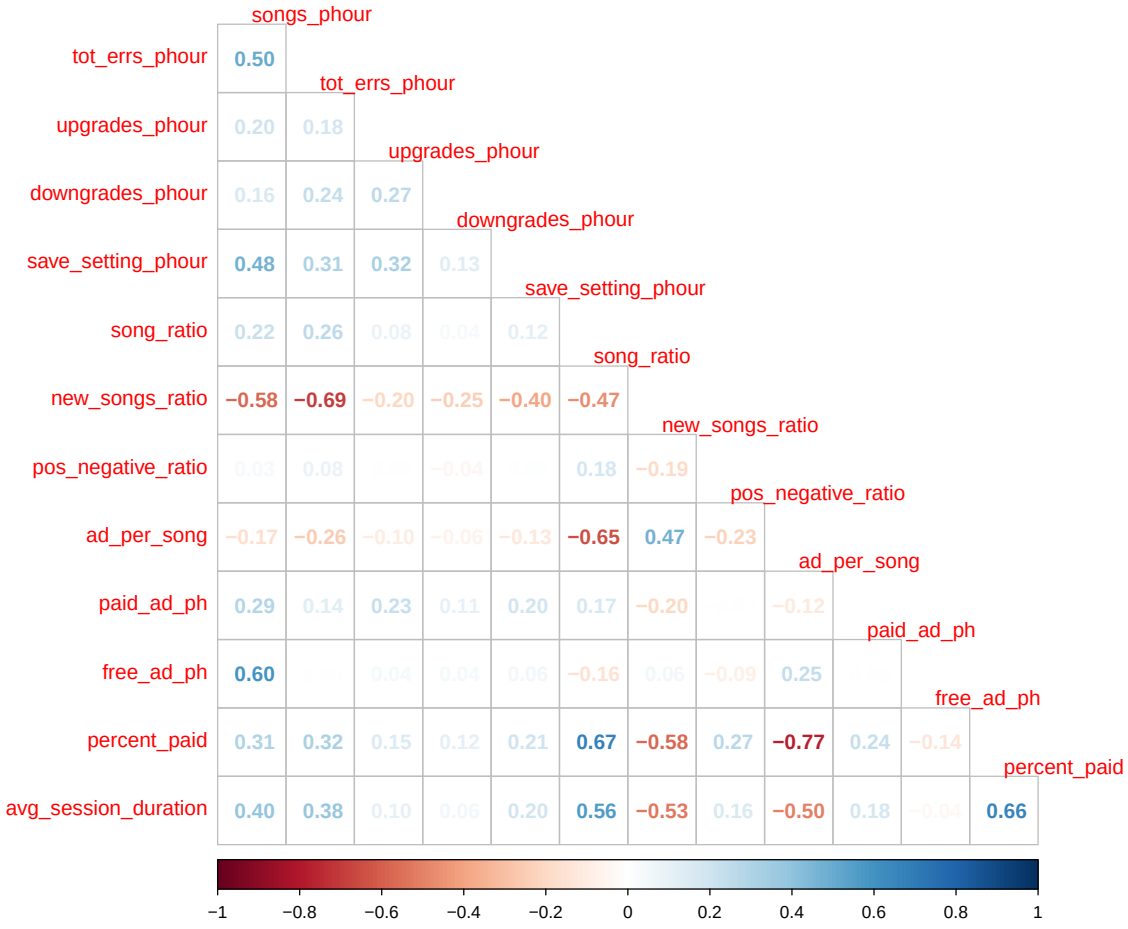
As expected, *avg_events_per_session* and *avg_session_duration* are highly correlated. It makes sense that the longer the session the more events there are. We would keep *avg_session_duration* and drop *avg_events_per_session*

To summarize, here is the list of variables we wanted to remove:

- tot_act_phour
- tot_tu_phour
- tot_td_phour
- freds_added_phour
- tot_add2PL_phour
- HP_visits_phour
- setting_phour
- avg_events_per_session

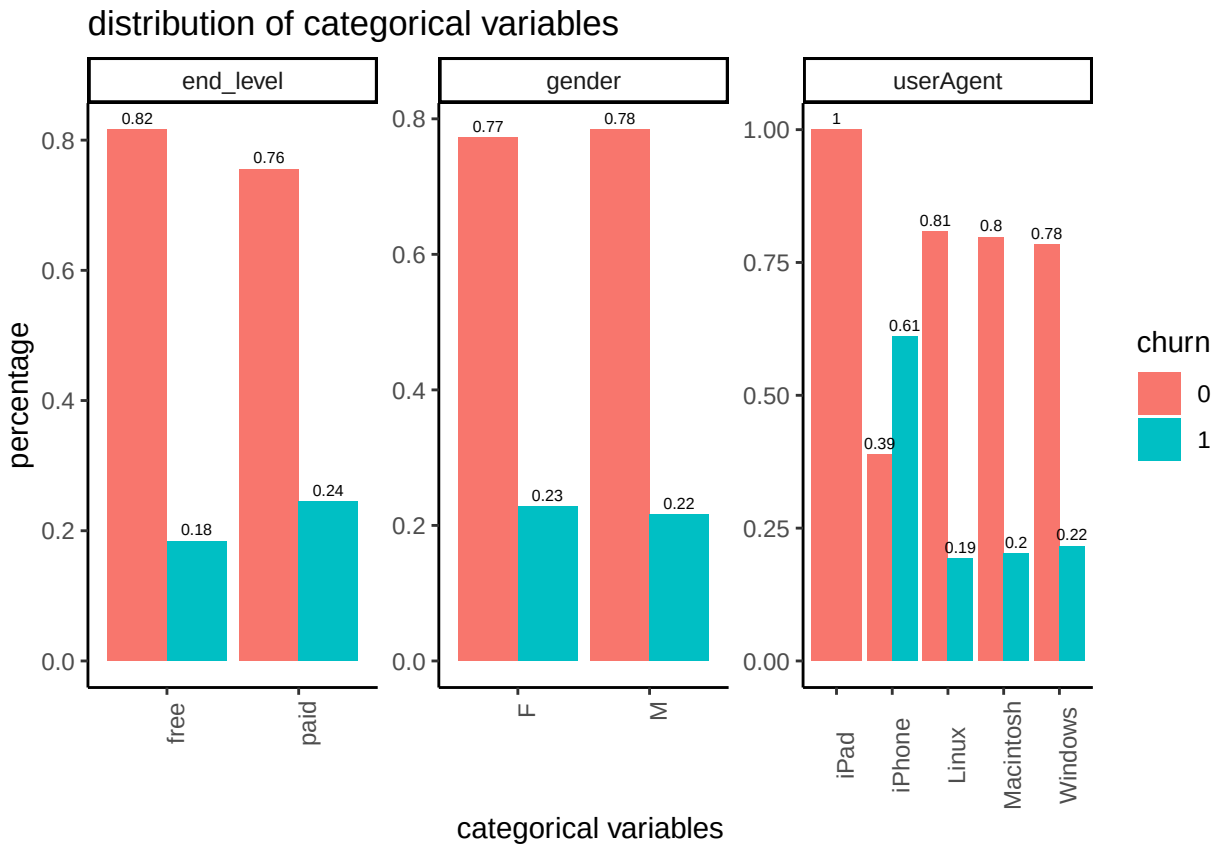
```
# str(train_df)
train_df$tot_act_phour <- NULL
train_df$tot_tu_phour <- NULL
train_df$tot_td_phour <- NULL
train_df$frds_added_phour <- NULL
train_df$tot_add2PL_phour <- NULL
train_df$HP_visits_phour <- NULL
train_df$setting_phour <- NULL
# train_df$diff_act_phour <- NULL
train_df$avg_events_per_session <- NULL
```

We have the following correlations after the highly correlated ones are removed.



4.2 Relationship with the target variable - *churn*

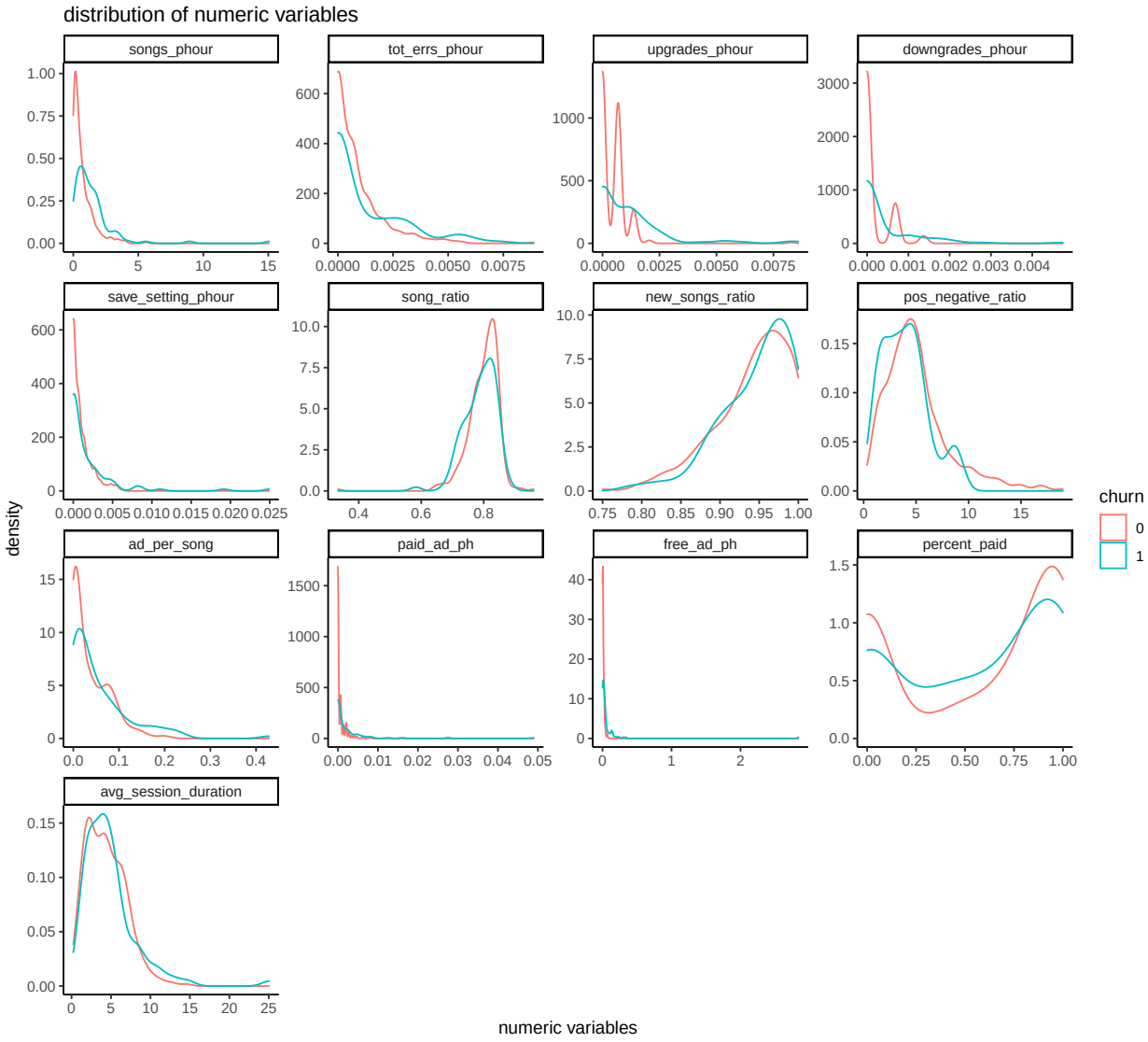
4.2.1 Distribution of categorical variables



From the plots, we find:

users will account level = paid seem to have a higher churn rate. The user's gender seems to have no effect on the likelihood of churning. iPhone users seem to be more likely to churn than other device users. It could be due to the fact the app has problems like errors on iphone (iOS). But it could also be just the sheer number of users on iphone rather than other device types.

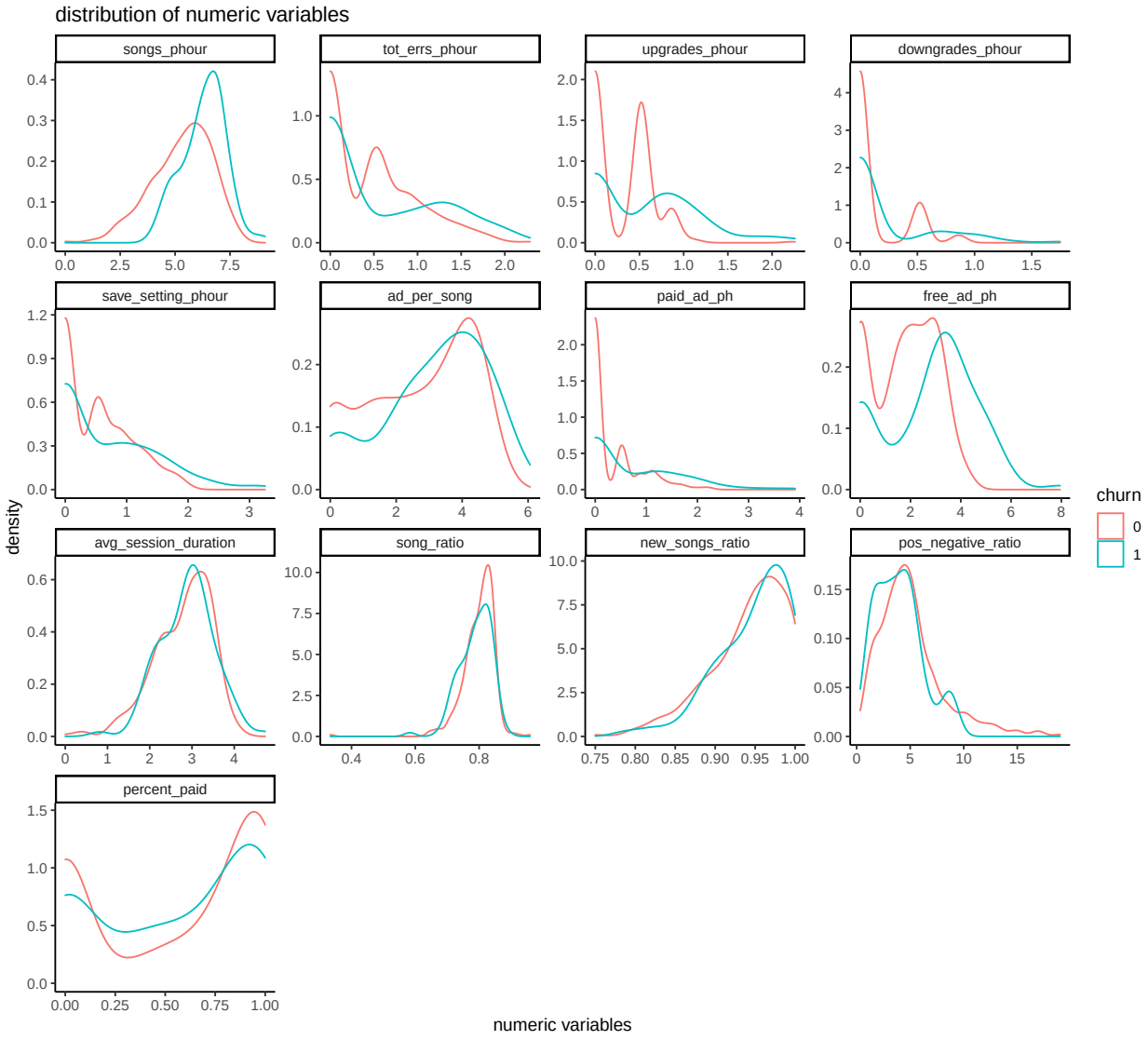
4.2.2 Distribution of numeric variables



In the following plots, some variables are log-transformed and shifted for the better visualization, they are not transformed in the data for our modeling.

```
train_df %>%
  keep(is.numeric) %>%
  select(-c(song_ratio, new_songs_ratio, pos_negative_ratio, percent_paid)) %>%
  map(., function (x) x+0.001) %>%
  map(., log) %>%
  as.data.frame() %>%
  apply(2,function(x) x-min(x))  %>%
  as.data.frame() %>%
  cbind(churn = as.factor(train_df$churn),
        song_ratio = train_df$song_ratio,
        new_songs_ratio = train_df$new_songs_ratio,
        pos_negative_ratio = train_df$pos_negative_ratio,
        percent_paid = train_df$percent_paid) %>%
```

```
  gather("key", "value", -churn, factor_key = T) %>%
  ggplot(aes(value, color = churn)) +
    facet_wrap(~ key, scales = "free") +
    geom_density() +
    theme_classic() +
    labs(x = "numeric variables",
         title = "distribution of numeric variables")
```



For the user activities *songs_phour*, *tot_errs_phour*, *upgrades_phour*, *downgrades_phour*, and *save_setting_phour*, the higher the number of activities, the more likely the user is going to churn. This may imply that the more the user uses the service, the more unsatisfying the user feels. We wanted to call out that *tot_errs_phour* is very illustrative of what is happening with churn. Churned users have a higher average of *tot_errs_phour* than non-churned.

Song Ratio (song_ratio) calculates how often is the user going to go to the next song among all the activities. You see that at ratio has a mean of of distribution near 0.8 for both churned and non-churned users. What I notice is there are way more people from non-churned that has that ratio than the churned. Not churned users have exhaustively tried out next song much more than churned users.

pos_negative_ratio seems to have little effect to the churn rate. This may imply that the dissatisfaction of the users are very likely to be irrelevant to the quality of the songs.

The effect of advertisement listening is not so obvious on the *ad_per_song plot*. However, the *free_ad_ph* and *paid_ad_ph* do show that higher number of ad listened leads to high rate of churn. The dissatisfaction of the users may come from the advertisements.

For *percent_paid*, users who do not change their account level (the ones staying 100% of the time free or 100% of the time paid) seem to have lower churn rate. They are content with service they are using. The ones who switched their account level may still feel unsatisfied about the service.

song_ratio, *new_songs_ratio*, *avg_session_duration* do not show any obvious patterns about churning.

The above findings are just ideas we get from the plots, the actual effect would need to be confirmed by our model analysis.

Lastly, the target variable that is in the training dataset is observed to be imbalanced.

```
summary(train_df$churn)
```

```
##   0   1
## 349  99
```

5. Modeling and Performance Evaluation

In order to build a model with higher unbiasedness, we would use the method of upsampling. The method adds duplicate records of the minor class to the sample so that the size of the minor class is adjusted to be close to the size of the major class.

```
temp <- train_df %>% filter(churn == 1) %>%
      slice(rep(1:n(),
            round(nrow(filter(train_df, churn == 0))/
                    nrow(filter(train_df, churn == 1)),0)-1))
train_df2 <- bind_rows(train_df, temp)
```

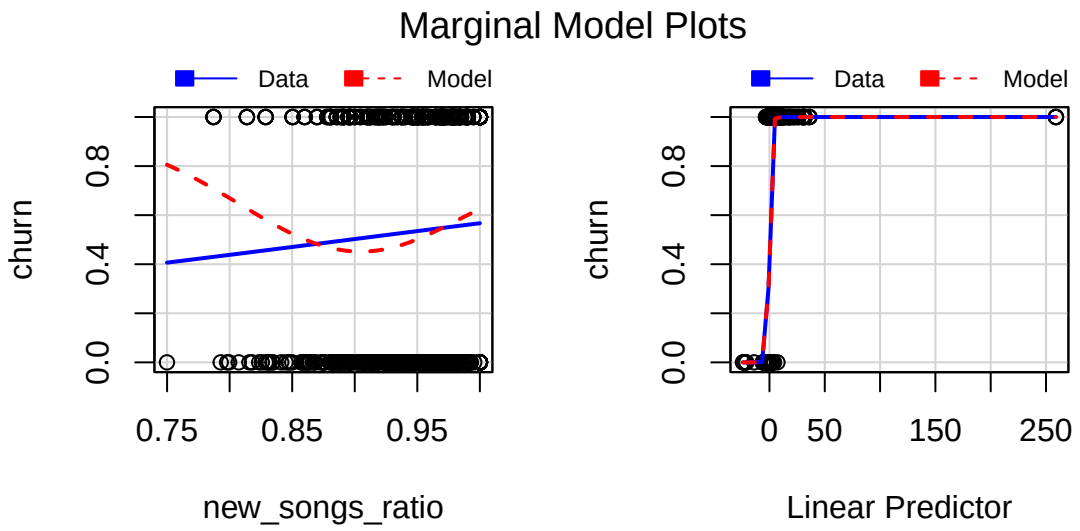
Finally, we can start building our model.

First, let's build our preliminary model with all the defined features.

```
model_logi <- glm(churn~.,family = binomial, train_df2)
```

Checking the marginal model plots, we find that the variable *new_songs_ratio* does not fit well to our model.

```
marginalModelPlots(model_logi,~new_songs_ratio)
```



In order to fit the curvature, we will add a squared term of *new_songs_ratio* to our model

```
model_logi <- glm(churn~.+I(new_songs_ratio^2),family = binomial, train_df2)
```

```
summary(model_logi)
```

```
##
## Call:
## glm(formula = churn ~ . + I(new_songs_ratio^2), family = binomial,
##     data = train_df2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0027  -0.4708   0.0000   0.2581   2.4211
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -6.290e+02  7.097e+02  -0.886  0.37541
## end_levelpaid         9.017e-01  4.897e-01   1.841  0.06556 .
## genderM               8.163e-02  2.642e-01   0.309  0.75736
## userAgentiPhone       1.829e+01  7.050e+02   0.026  0.97931
## userAgentLinux        1.609e+01  7.050e+02   0.023  0.98179
## userAgentMacintosh    1.601e+01  7.050e+02   0.023  0.98188
## userAgentWindows      1.597e+01  7.050e+02   0.023  0.98193
## songs_phour           8.935e+00  1.054e+00   8.480  < 2e-16 ***
## tot_errs_phour       -1.990e+01  1.373e+02  -0.145  0.88472
## upgrades_phour        1.094e+02  2.475e+02   0.442  0.65843
## downgrades_phour     -2.055e+02  3.942e+02  -0.521  0.60220
## save_setting_phour    3.036e+01  1.070e+02   0.284  0.77658
## song_ratio           -4.434e+00  3.872e+00  -1.145  0.25216
## new_songs_ratio       1.163e+03  1.640e+02   7.091 1.34e-12 ***
## pos_negative_ratio   -1.506e-01  5.470e-02  -2.753  0.00591 **
## ad_per_song           9.938e+00  5.395e+00   1.842  0.06544 .
## paid_ad_ph            2.782e+02  9.471e+01   2.938  0.00331 **
## free_ad_ph            4.491e+01  1.376e+01   3.263  0.00110 **
```

```
## percent_paid          2.241e+00  8.144e-01   2.751  0.00594 **
## avg_session_duration  1.811e-03  7.150e-02   0.025  0.97979
## I(new_songs_ratio^2) -5.495e+02  8.246e+01  -6.664 2.66e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1029.82  on 744  degrees of freedom
## Residual deviance:  428.78  on 724  degrees of freedom
## AIC: 470.78
##
## Number of Fisher Scoring iterations: 15
```

From our full model, there are some features that are insignificant. We would utilize the method of backward elimination based on the AIC score.

```
model_logi <- step(model_logi, trace=0)
```

The following is the result of our final model. Most of the coefficients are statistically significant

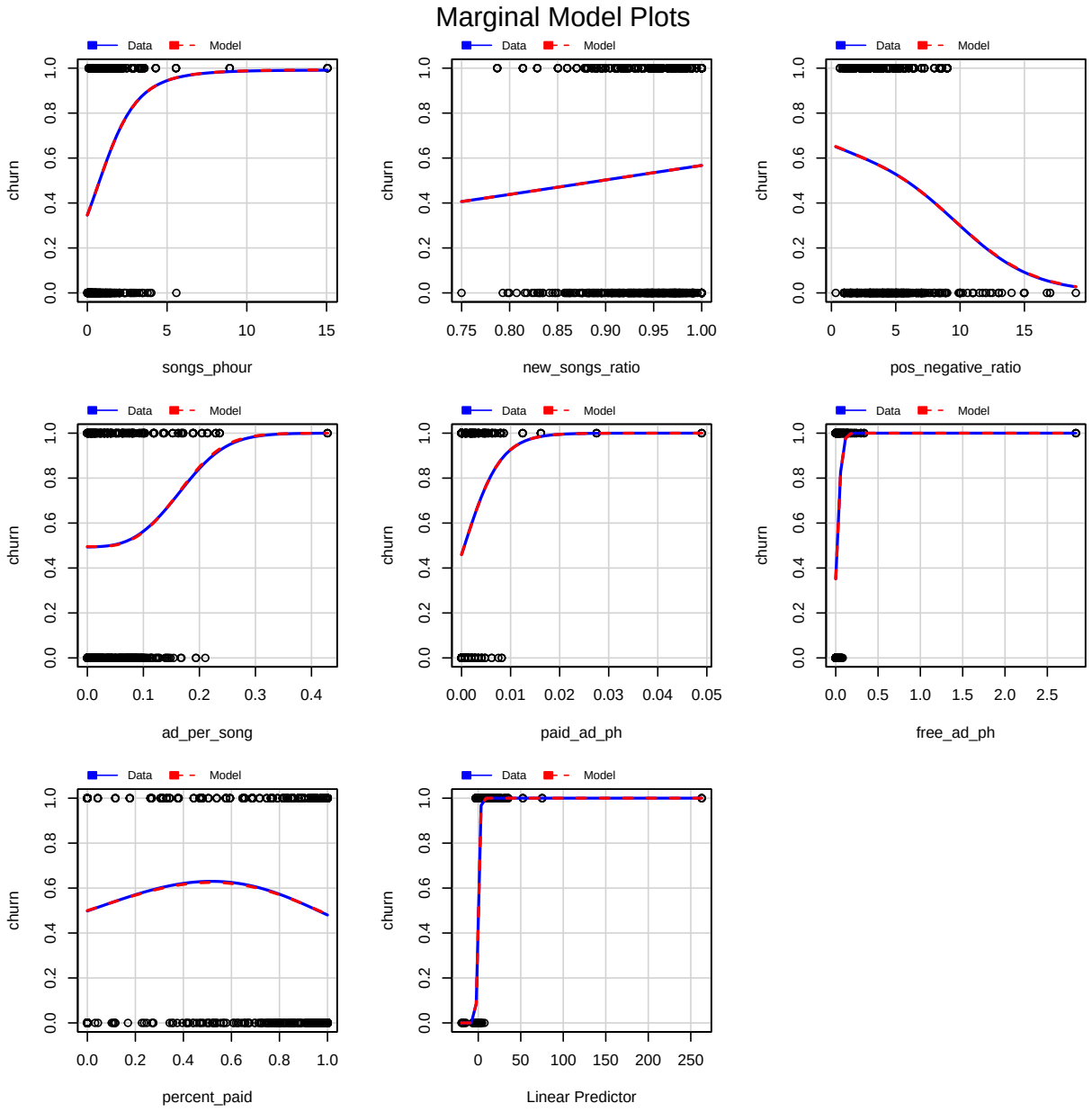
```
summary(model_logi)
```

```
##
## Call:
## glm(formula = churn ~ end_level + userAgent + songs_phour + new_songs_ratio +
##     pos_negative_ratio + ad_per_song + paid_ad_ph + free_ad_ph +
##     percent_paid + I(new_songs_ratio^2), family = binomial, data = train_df2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.8177  -0.4831   0.0000   0.2596   2.4452
##
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -628.87451  692.48295  -0.908 0.363802
## end_levelpaid           1.10537    0.38790   2.850 0.004377 **
## userAgentiPhone        18.36385  688.03763   0.027 0.978707
## userAgentLinux         16.16706  688.03760   0.023 0.981254
## userAgentMacintosh     16.07003  688.03745   0.023 0.981366
## userAgentWindows       16.02738  688.03744   0.023 0.981415
## songs_phour             8.93534    0.97286   9.185  < 2e-16 ***
## new_songs_ratio      1153.47860  156.71905   7.360 1.84e-13 ***
## pos_negative_ratio     -0.14576    0.05143  -2.834 0.004593 **
## ad_per_song            11.74476    5.15449   2.279 0.022694 *
## paid_ad_ph            277.22956   90.61246   3.060 0.002217 **
## free_ad_ph             45.63953   12.18794   3.745 0.000181 ***
## percent_paid            1.96790    0.74120   2.655 0.007930 **
## I(new_songs_ratio^2) -543.96482   78.58793  -6.922 4.46e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 1029.8  on 744  degrees of freedom
## Residual deviance:  430.5  on 731  degrees of freedom
## AIC: 458.5
##
## Number of Fisher Scoring iterations: 15
```

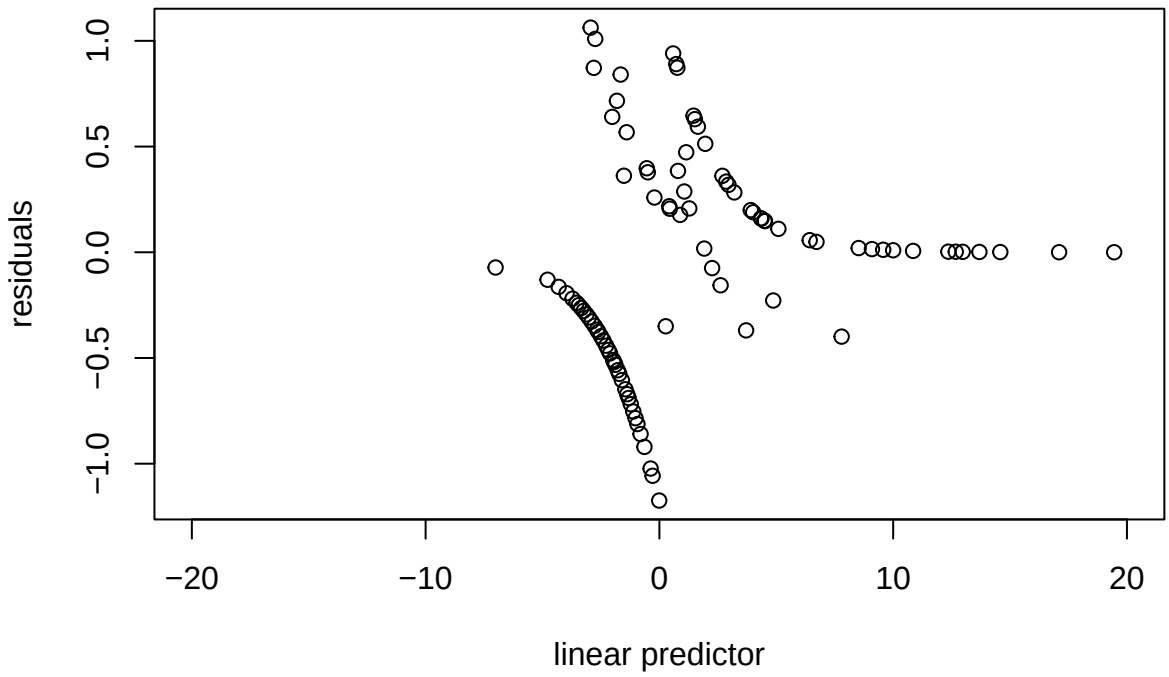
By looking at the marginal model plots, there is no lack of fit of the model.

```
marginalModelPlots(model_logi,~songs_phour+new_songs_ratio+pos_negative_ratio+ad_per_song+
                   paid_ad_ph+free_ad_ph+percent_paid, layout =c(3,3))
```



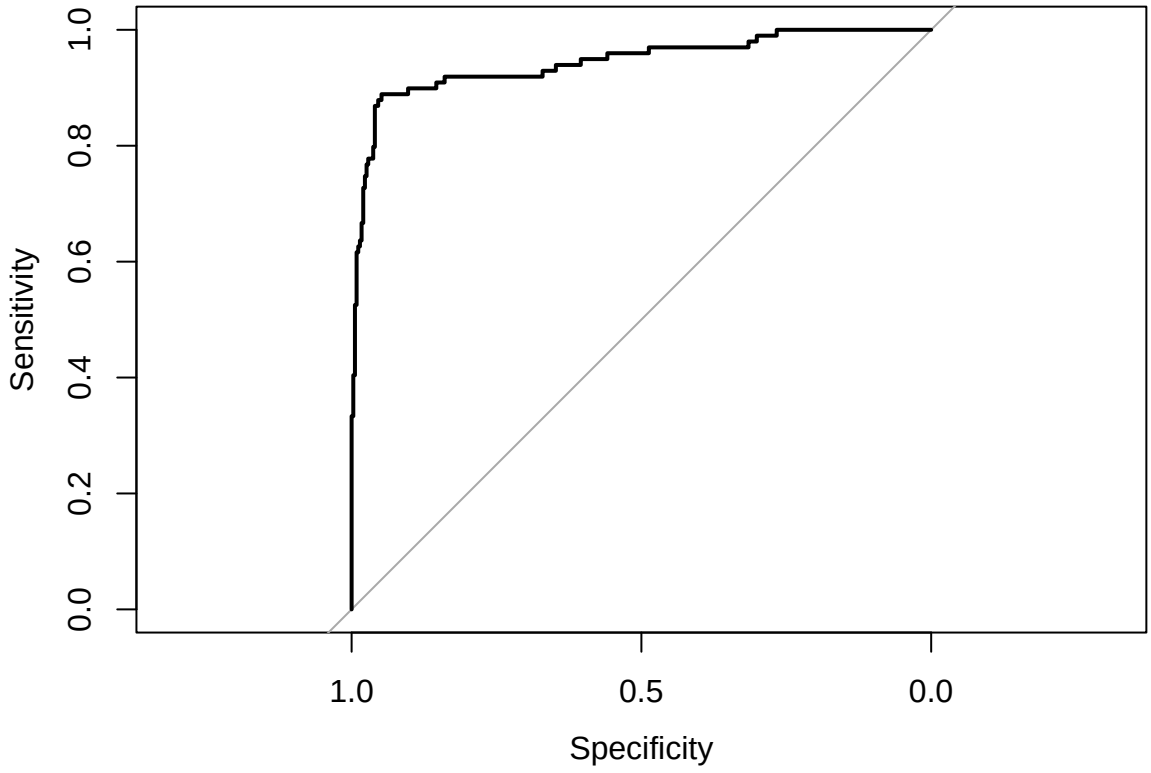
The following deviance residual vs linear predictor plot also confirms that our model is valid. The model is producing accurate predictions at the two ends. The errors around the match point 0 are independent and random.

```
residual_df <- mutate(train_df2, residuals=residuals(model_logi,type="deviance"),
                      linpred=predict(model_logi,type = "link"))
gdf <- group_by(residual_df, cut(linpred, breaks=unique(quantile(linpred,(1:100)/101))))
diagdf <- summarise(gdf, residuals=mean(residuals), linpred=mean(linpred))
plot(residuals ~ linpred, diagdf, xlab="linear predictor",xlim=c(-20,20))
```



Now let's check the performance of our model

```
rocCurve <- roc(train_df2$churn, model_logi$fitted.values)
plot(rocCurve)
```




```
rocCurve$auc
```

```
## Area under the curve: 0.9445
```

We have an AUC of 0.9445, which is a good number. The optimal threshold of classifying the target variable *churn* is

```
optimal_threshold <- coords(rocCurve, "best", ret = "threshold")
optimal_threshold
```

```
##   threshold
## 1 0.5739793
```

Performance evaluation using the up-sampled data

```
predicted_class <- ifelse(model_logi$fitted.values>optimal_threshold[1,1],1,0)
confusion_matrix <- confusionMatrix(as.factor(predicted_class),
                                    train_df2$churn,
                                   mode = "everything",positive = "1")
confusion_matrix
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   0   1
##          0 331  44
##          1  18 352
##
##                Accuracy : 0.9168
##                  95% CI : (0.8946, 0.9356)
##     No Information Rate : 0.5315
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8336
##
##  Mcnemar's Test P-Value : 0.001498
##
##             Sensitivity : 0.8889
##             Specificity : 0.9484
##          Pos Pred Value : 0.9514
##          Neg Pred Value : 0.8827
##               Precision : 0.9514
##                  Recall : 0.8889
##                      F1 : 0.9191
##              Prevalence : 0.5315
##          Detection Rate : 0.4725
##    Detection Prevalence : 0.4966
##       Balanced Accuracy : 0.9187
##
##        'Positive' Class : 1
##
```

Performance evaluation using the pre-up-sampled data

```
predicted_class <- ifelse(predict(model_logi,train_df,type="response")>optimal_threshold[1,1],1,0)
confusion_matrix <- confusionMatrix(as.factor(predicted_class),
                                    train_df$churn,
                                   mode = "everything",positive = "1")
confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 331  11
##          1  18  88
##
##                Accuracy : 0.9353
##                  95% CI : (0.9084, 0.9562)
##     No Information Rate : 0.779
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8166
##
##  Mcnemar's Test P-Value : 0.2652
##
```

```
##             Sensitivity : 0.8889
##             Specificity : 0.9484
##          Pos Pred Value : 0.8302
##          Neg Pred Value : 0.9678
##               Precision : 0.8302
##                  Recall : 0.8889
##                      F1 : 0.8585
##              Prevalence : 0.2210
##          Detection Rate : 0.1964
##    Detection Prevalence : 0.2366
##       Balanced Accuracy : 0.9187
##
##        'Positive' Class : 1
##
```

The evaluation using the pre-upsampled data and upsampled data both have a Balanced Accuracy of over 91% and a F1 score over 0.85

6. Conclusion and Next Steps

Now let's confirm the findings from our model output

```
summary(model_logi)
```

```
##
## Call:
## glm(formula = churn ~ end_level + userAgent + songs_phour + new_songs_ratio +
##     pos_negative_ratio + ad_per_song + paid_ad_ph + free_ad_ph +
##     percent_paid + I(new_songs_ratio^2), family = binomial, data = train_df2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.8177  -0.4831   0.0000   0.2596   2.4452
##
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -628.87451  692.48295  -0.908 0.363802
## end_levelpaid           1.10537    0.38790   2.850 0.004377 **
## userAgentiPhone        18.36385  688.03763   0.027 0.978707
## userAgentLinux         16.16706  688.03760   0.023 0.981254
## userAgentMacintosh     16.07003  688.03745   0.023 0.981366
## userAgentWindows       16.02738  688.03744   0.023 0.981415
## songs_phour             8.93534    0.97286   9.185  < 2e-16 ***
## new_songs_ratio      1153.47860  156.71905   7.360 1.84e-13 ***
## pos_negative_ratio     -0.14576    0.05143  -2.834 0.004593 **
## ad_per_song            11.74476    5.15449   2.279 0.022694 *
## paid_ad_ph            277.22956   90.61246   3.060 0.002217 **
## free_ad_ph             45.63953   12.18794   3.745 0.000181 ***
## percent_paid            1.96790    0.74120   2.655 0.007930 **
## I(new_songs_ratio^2) -543.96482   78.58793  -6.922 4.46e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1029.8  on 744  degrees of freedom
## Residual deviance:  430.5  on 731  degrees of freedom
## AIC: 458.5
##
## Number of Fisher Scoring iterations: 15
```

- The coefficients of *songs_phour* is positive, which implies that the more the user uses the service, the more unsatisfying the user feels. It is consistent with our findings from the distribution plots
- The coefficients of *end_levelpaid* and *percent_paid* both indicate that the users are unsatisfied with the paid service.
- The coefficients of *ad_per_song*, *paid_ad_ph*, and *free_ad_ph* indicate that the more advertisement the user listened, the more likely the user is going to churn.
- The coefficient of *paid_ad_ph* is 6 times the coefficient of *free_ad_ph*. Being a premium user but still need to listen to ads is intolerable to the users.
- The *new_songs_ratio* has two coefficients, but since the range of the variable is 0-1, the combined effect is always positive. This tells us that users who find songs they like to listen repeatedly are less likely to churn.
- The coefficient of *pos_negative_ratio* is negative. This indicates that the quality of the songs does matter, even though it is not showed obviously on the distribution plots. The better the user likes the songs, the lower the rate of churn.
- Additionally, though the coefficients of *userAgent* are not statistically significant, the feature does help improve the model based on the AIC score. The coefficient of iPhone is the largest, which confirms our findings from the distribution plots that iphone users have higher churn rate.

Though our model may help predict the users that may churn so the company can take actions such as offering discounts to ask them stay. This is also a common practice that is mentioned by the references found in the Literature Review section. The tools company has is to be more proactive by leveraging the churn model to garner more signals to devise better segmentation strategies and intervention strategies to right the course of the boat, which literally means the business objectives of lowering churn rates and attaining a higher retention rate of paid customers. This can be achieved by aligning the right offers for these users with high propensity to churn, with whether an email creative with messages to convince the customer to stay, or simply sending them to the outbound sales team to provide the customer with additional financial incentives to downgrade or stay with the plan. In addition, it's imperative to continually to make product improvements to the streaming app service to lower the chance of churn.

As Data Science experts and consultants to this fictitious music streaming app company Sparkify, we have the following recommendations:

1. Improve the user experience of their premium (account level: paid) users, especially removing advertisements that proved to be providing negative customer experience resulting in churn.
2. Lower the cadence of ads to the paid level users.
3. Remove the frictions encountered by iPhone users.
4. Personalization of the offerings of songs. Refine their recommender system to suggest songs or packaged in a way of recommended playlist for each individual user.

Recommendation 3 & 4 are definitely something that can be done with some sort of modeling approach. But due to the fact we don't have the necessary data to help build these Machine Learning (ML) models, we wanted to make sure the company is aware that these are some recommended next steps the company can take.