Eddie Shim
Playlist Skip Predictor Model

Model goal: Predict whether or not a user will skip past a recommended track
Ideas for defining skip:

1. Hard cutoff of *x* seconds on listen duration
    - Drawback: would not classify against shorter songs well if *x* is too high
2. Percentage of song user has listened to where

$$percentage\_listened = \frac{listen\_duration\_seconds}{track\_duration\_seconds}$$

An ideal definition would capture nuances of the data more finely. For the purposes of starting simple, let's continue with option 2. The next question that follows is: at what percentage do we classify skipped versus listened to? Let's take a look at the data.

Track duration:

```
> summary(track_duration)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0   302.8   374.8   524.0   453.7 42004.1
```

Let's take a look at the log transformed distribution to get a clearer visualization. It's almost normal but exhibits high kurtosis and some fat tails.

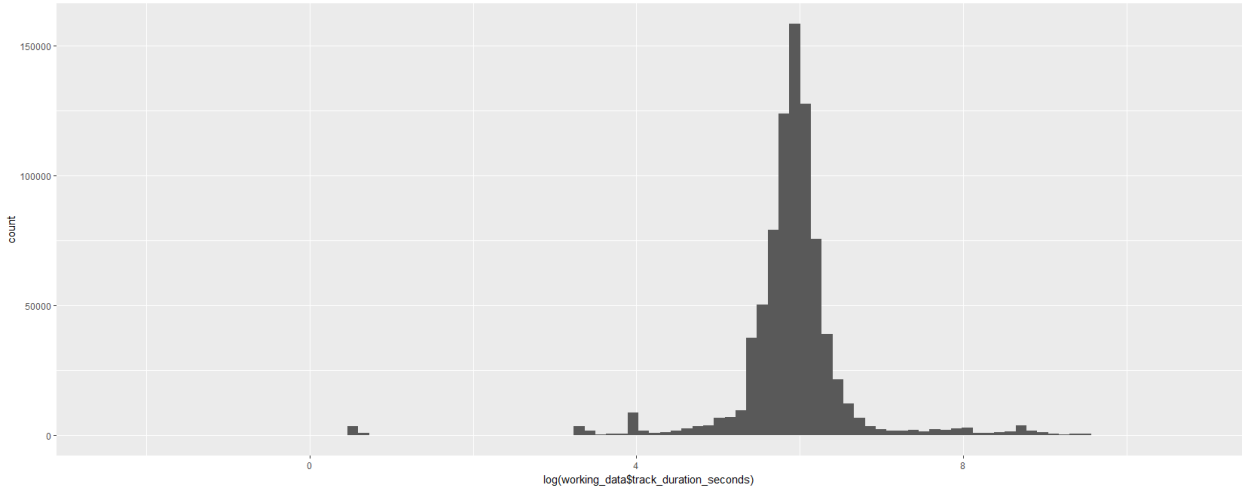


Figure 1

Listening duration:

```
> summary(working_data$listen_duration_seconds)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
  -70.49    42.04   280.62   300.80   396.09 34654.27
```

Let's take a look at the log transformed distribution to get a clearer visualization. It's skewed left with a distinct cutoff point at x = 5 (*e*^5 = 148 seconds).

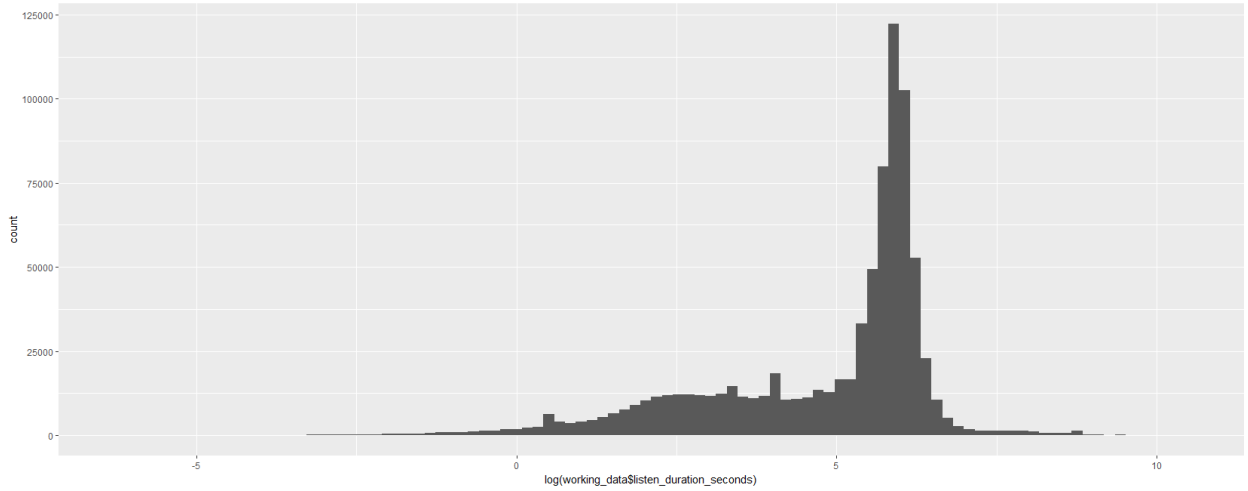


Figure 2

Percentage listened:

```
> summary(percentage_listened)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
-0.1000  0.1323  1.0000     Inf  1.0000     Inf      10
```

Let's check how the data is distributed. Here's what it looks like when we zoom in on 0 to 100% scale in the *x*-axis. Surprisingly a lot of 100% points.

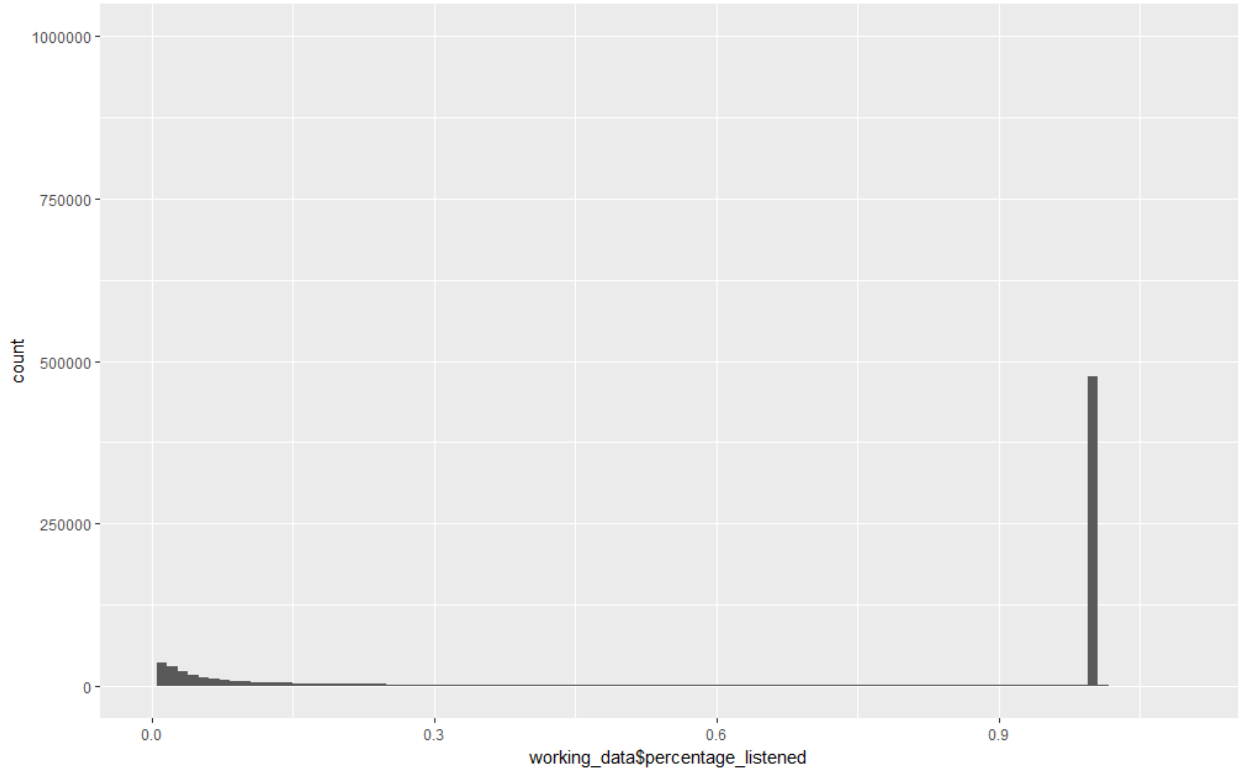


Figure 3

Some metrics to help grasp the scale of the data:

- Total dataset: ~820K rows
- Under 100%: ~350K (42%)
- At 100%: ~460K (55%)
- Over 100%: ~13K (2%)
- Under 30%: ~260K (31%)
- Under 15%: ~215K (26%)
- Under 5%: ~145K (18%)
- NA points: 10 (likely a data recording issue. Trivial amount, throw away these points)
- Inf points: 629 (means `track_duration` = 0. Let's keep these points)

I decided to define 'skipped songs' as songs with less than 5 percentage listened. When looking at the `percentage_listened` graph above, there is a distinct drop in frequency at 5% and at 10%. Intuitively, 5% makes more sense. The median time length of these recommended songs is 374 seconds, and the first quartile falls at 302 seconds; at 5%, that's 18 and 15 seconds respectively. Intuitively, I would make the assumption that 15 seconds of listening seems like a significant enough time to consider as a non-skipped song.
Now we have a binary metric we can define as the dependent variable. Categorize all data under 5 `percentage_listened` as 0 and the rest of the data as 1.

Model Selection:

Amongst binary classification models, I will proceed with a tree-based machine learning model called Extreme Gradient Boost (xgboost) because of its interpretability, robustness to mixed data and outliers, computational efficiency, robustness against overfitting (regularization), and predictive power. Other classification models in consideration would be Support Vector Machine, Random Forest (a bagging model), or logistic regression.

Data Selection:

It is important to choose dependent variable that add predictive power through signal, not random noise. I will throw out `client_version, track_id, listener_id`. These variables have no causation signal towards predicting whether the song is skipped or not. I will also throw out `track_duration, listen_duration, and listener_prev_month_listening_time`, to prevent double counting since all of these variables are already covered by a proxy or transformed variable. I also transform all time stamps into relevant buckets (hours, days, year).

Independent Variables

1. ts_day (categorical)
2. ts_hour (categorical)
3. ts_year (categorical)
4. track_upload_day (categorical)
5. listener_signup_day (categorical)
6. country_code (categorical)
7. listening_context (categorical)
8. recommender_algorithm_name (categorical)
9. track_genre_category (categorical)
10. listener_top_genre_category_listened (categorical)
11. listener_prev_month_avg_daily_track_listened (numerical)
12. listener_prev_month_listening_time_hours (numerical)

Dependent Variable:

1. took_rec (binary)

Model Results:

Confusion Matrix:

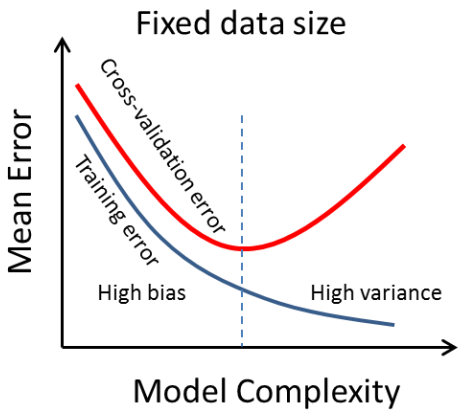
Absolute

Predicted	Observed 0	Observed 1
0	24,733	6,748
1	120,153	676,525
	Total:	828,159

Relative

Predicted	Observed 0	Observed 1
0	3%	1%
1	15%	82%
	Overall accuracy:	85%

A lot of false positives (15%) compared to false negatives (1%). Our algorithm is overly optimistic on predicting who takes recommendations. Overall, we have a prediction accuracy of 85%. In terms of addressing overfitting, there are a few tools I've implemented. I used a *k*-fold cross validation method in order to choose optimal parameters which weigh upon model accuracy versus model parsimony. XGBoost naturally has a regularization term as it builds itself, penalizing the model for overfitting its parameters to the specific dataset. If I had more time, I would plot a logloss versus model complexity graph in order to validate our parameter selection. For example:



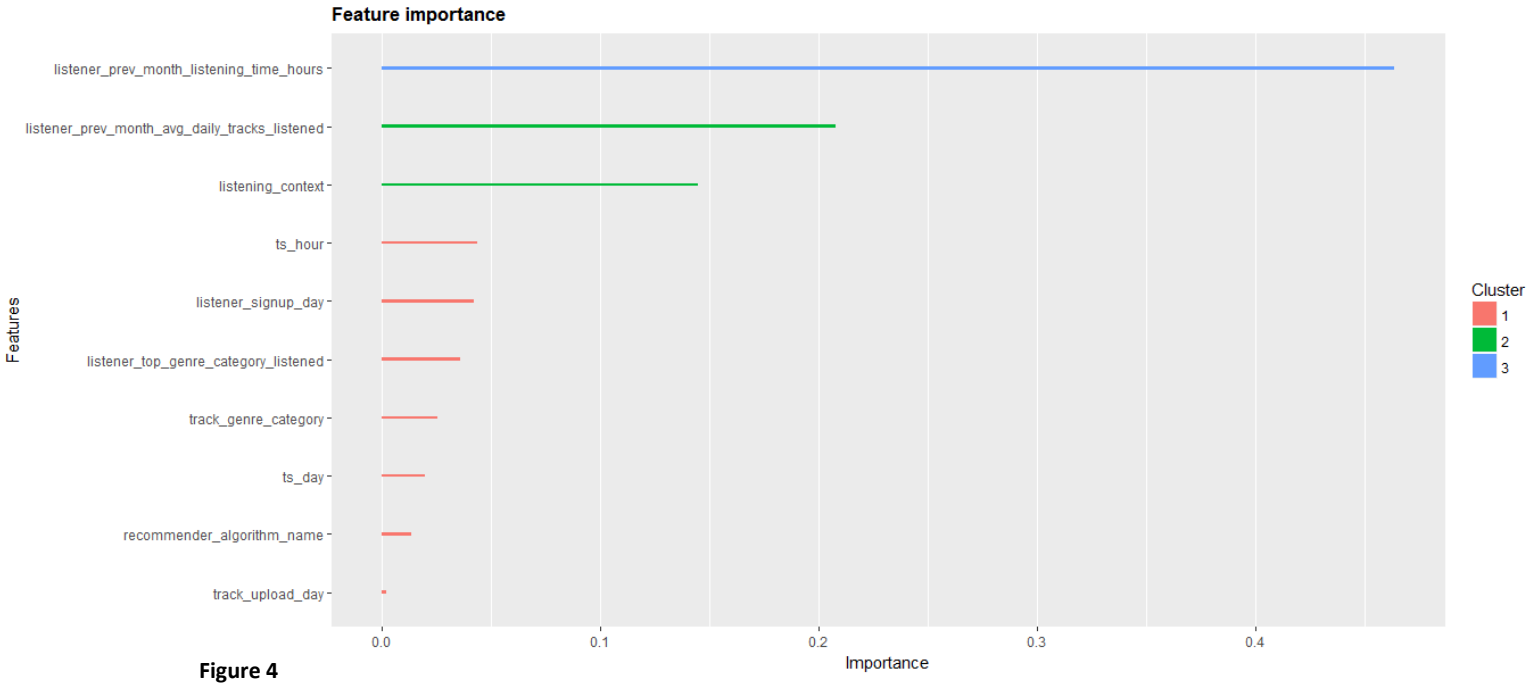


Figure 4

	Feature	Gain
1	listener_prev_month_listening_time_hours	46%
2	listener_prev_month_avg_daily_tracks_listened	21%
3	listening_context	14%
4	ts_hour	4%
5	listener_signup_day	4%
6	listener_top_genre_category_listened	4%
7	track_genre_category	3%
8	ts_day	2%
9	recommender_algorithm_name	1%
10	track_upload_day	0%
11	country_code	0%

Gain can be thought of as a metric that is a proxy for how much signal is extracted from a feature. It is the information gain the model can organize on the data when separating based on a split on the variable (e.g. everyone who listened less than *x* hours in the previous month represents *y*% of the population, and *y* is used to calculate gain). It looks like from our model, the most important six variables containing the most signal are:

1. Listener's previous month listening time hours
2. Listener's previous month average daily track listened
3. Timestamp (hour)
4. Listening context
5. Listener signup date
6. Listener top genre category listened

Below is a snippet of the xgb.dump chart, which prints all the trees considered within the model. We can see the first optimal split is on feature 2, which is listener_prev_month_listening_hours < 7.5. To gain more of an intuitive sense of this chart, let's look at some plots before diving into the analysis.

```
booster[0]
0:[f2<7.5] yes=1                    no=2        missing=1   gain=2302.7     cover=103572
1:[f2<6.5] yes=3                    no=4        missing=3   gain=2783.63    cover=85603.5
3:[f0<1.46947e+009] yes=7           no=8        missing=7   gain=2149.55    cover=25745.2
7:[f9<2777.78] yes=15               no=16       missing=15  gain=487.338    cover=24117.5
15:[f9<2776.68] yes=31              no=32       missing=31  gain=1179.57    cover=16576.8
31:[f6<1.37864e+009] yes=63         no=64       missing=63  gain=123.089    cover=16403.5
63:leaf=0.831824                    cover=4140.5
64:leaf=0.731205                    cover=12263
32:[f0<1.46778e+009] yes=65         no=66       missing=65  gain=64.9855    cover=173.25
```

Figure 5

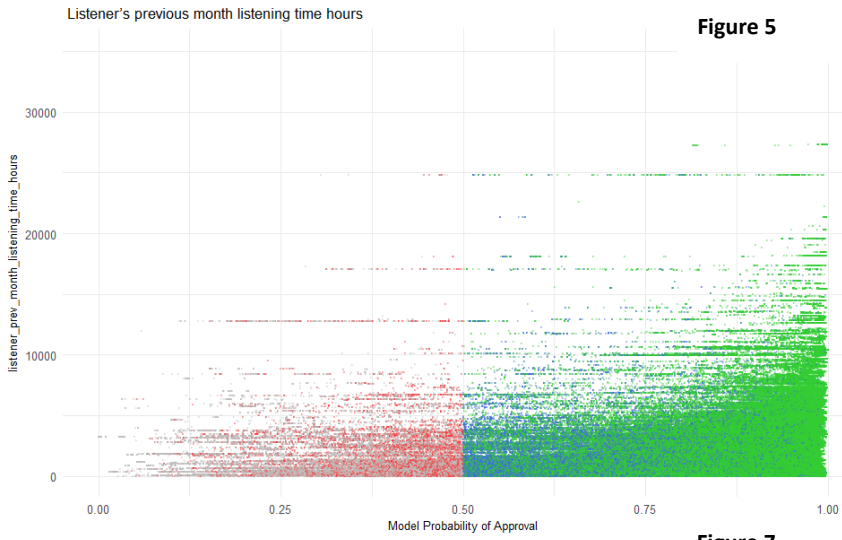


Figure 6

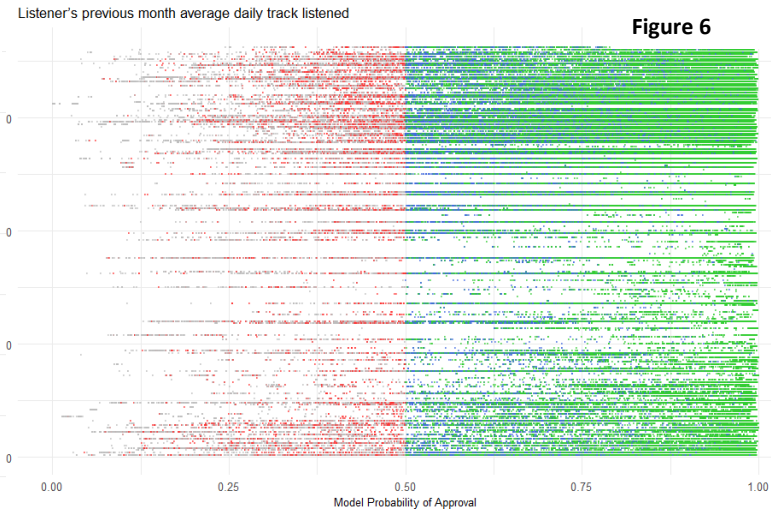


Figure 7

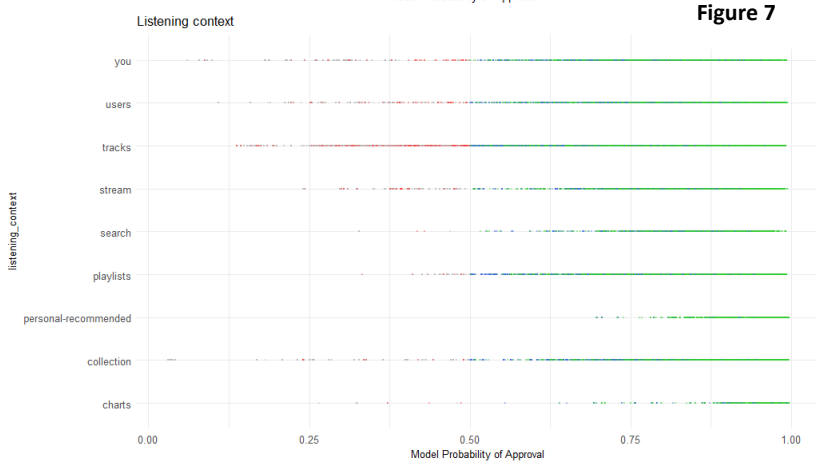


Figure 8

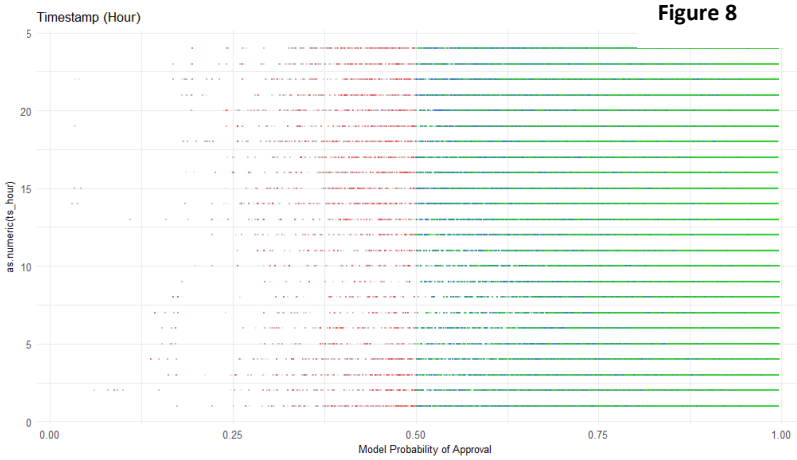


Figure 9

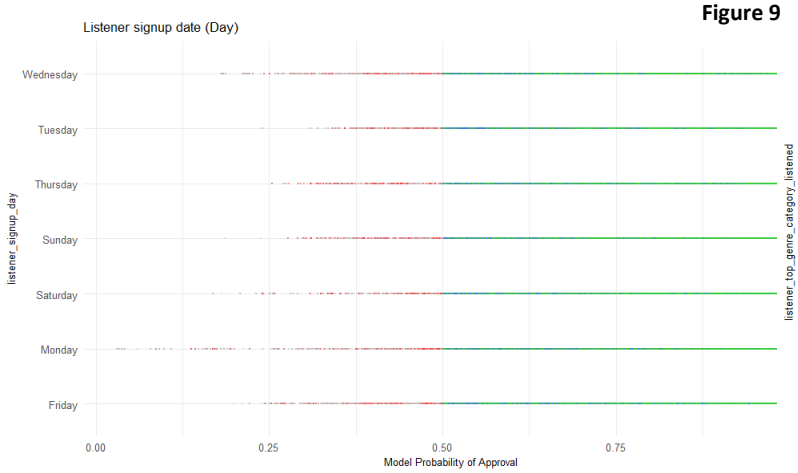
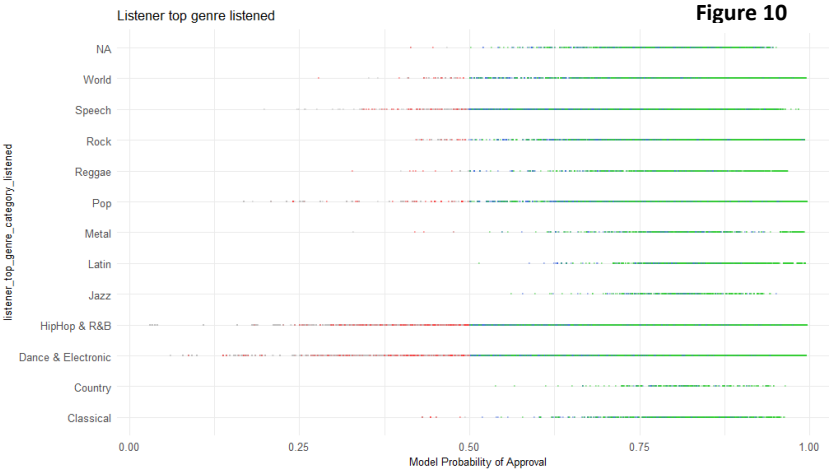
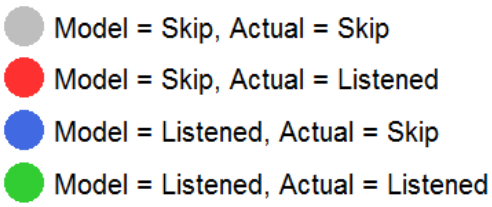


Figure 10





Note: one of the issues with displaying large datasets is the trouble in seeing overlapping points. Let's take a look at the last four plots on a discrete bar graph scale instead of continuous dot plots.

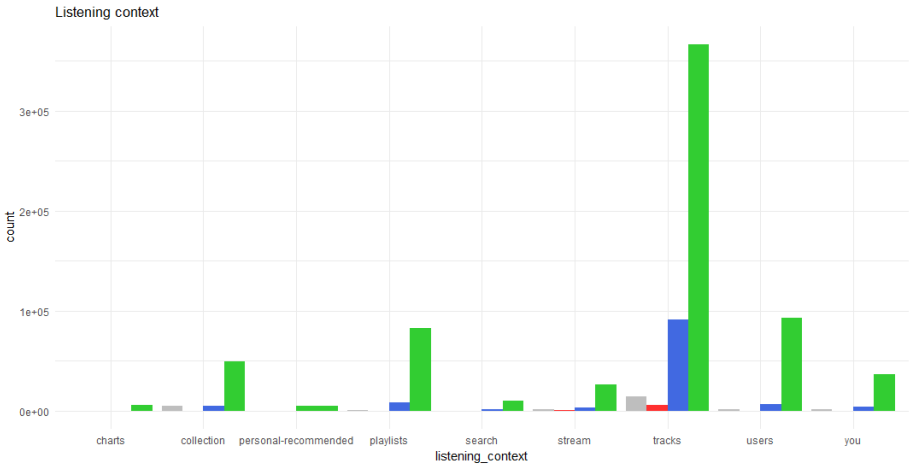


Figure 11

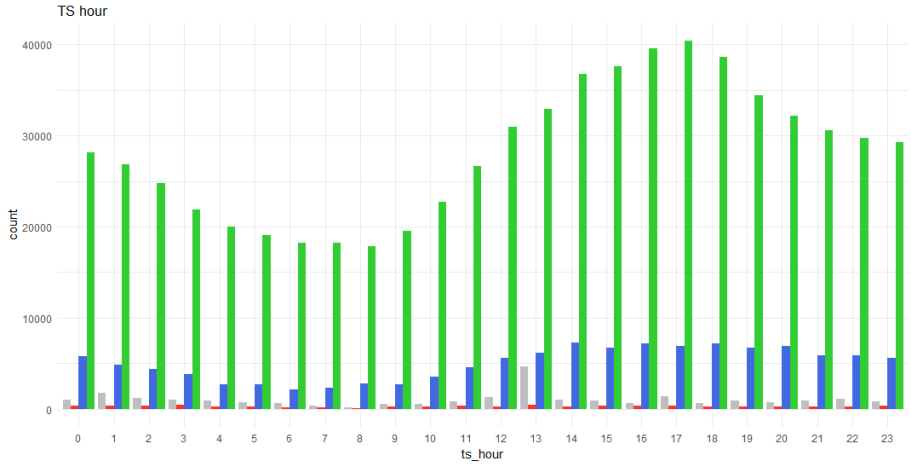


Figure 12

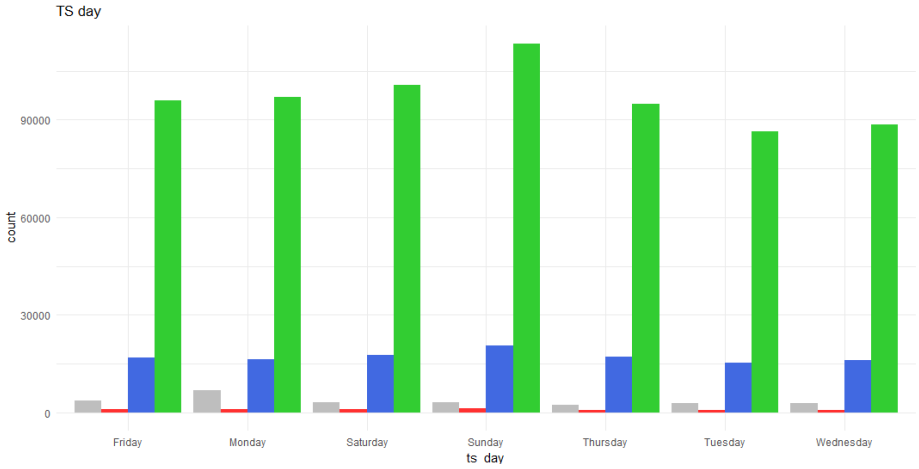


Figure 13

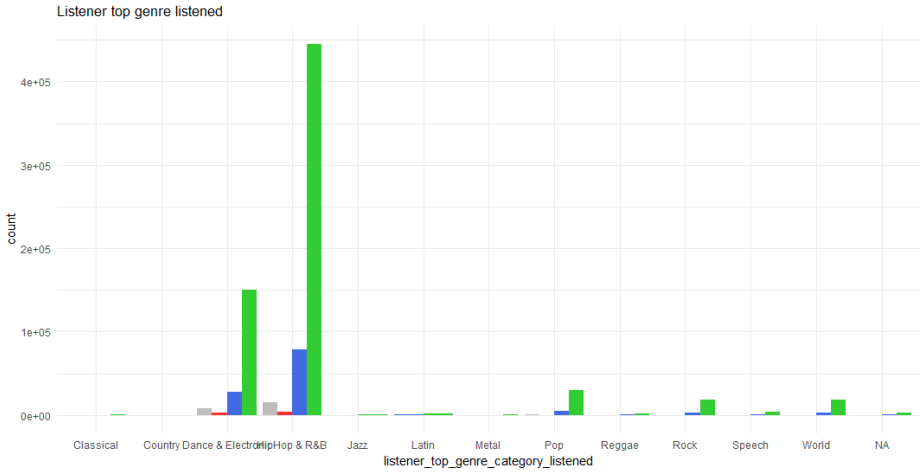


Figure 14

Interpretation of model results and data visualization:

- For the top two most important variables (listener's previous month listened, month average daily track listened), we see that users who use this music app more are more likely to take recommendations. Intuitively, this makes sense; there is probably a positive feedback loop at play. If the user likes the recommendation, they will probably come back to use this app more. However, this could be a 'chicken and egg' problem, the causation might be flipped; people who use this app heavily may just have more exposure to recommended songs and thus may have a higher hit rate
- For listening context, observe many categories have a high skew left shape. Search, charts, and personal recommended almost no red and grey dots whereas tracks, users, and you had a larger spread of hit rates.
- There's some time seasonality when looking at ts_hour. Usage peaks around 5PM and bottoms out around 8AM. 1PM had an unusually high skip rate (high grey bar).
- Sunday is the peak time for days of the week. Monday had an unusually high skip rate (high grey bar).
- EDM and R&B dominate the dataset. Jazz, Classical, Metal had effectively no blue or grey dots, meaning there was high signal for a user to take recommendation.

Bottomline:

- Focus on EDM and R&B users. They dominate the population. Segment these genres further, these are too macro of genres
- Recommender algorithms don't provide much signal. Take a further look into these
- Focus on accelerating that positive feedback loop of heavy users and high recommendation hit rates. 1) focus on getting more people into the heavy user bucket (>7.5 hours use as seen by first tree split in xgb dump table) and 2) focus on retaining heavy users
- Advertise at peak hours and days. There is clear time correlation in usage
- Look into why 1PM has such a high skip rate (grey bar). Could be data error or some fundamental reason (are you releasing more/different recommendations at 1PM?)

How to improve the model / what I would continue to work on:

- Current model is overly optimistic (too many blue dots). Would develop another metric, blue to green dot ratios and dive into segmentation analysis
- Implement other challenger models (SVM, logistic regression, random forest) to see if we can increase predictive power without overfitting. If interesting enough to make this project into a scalable, long term project, reorganize models with more cognizance towards memory and speed, and utilize object orientated programing
- Use more comprehensive data:
    - Increase size of the data or hone in on specific time period of interest (e.g. launch new feature in 2017, limit data to 2016- present)
    - Popularity of recommended song (could use likes/repost on track as proxy)
    - Popularity of recommended artist (could use artist subscription number as proxy)
    - If user likes recommended song
    - If user reposts recommend song
    - If and when advertisements were playing during recommended song