more_exploration

February 20, 2022

Game Plan: 1. Show that the choice of outcome variable does not matter very much because of the extremely high correlation between potential outcomes. - Create a nice looking correlation plot for the outcomes - Probably try to create the sns.pairplot() although that function was very very slow 2. Point out that the Spotify playlists are MUCH more successful than the typical user playlist - Can show that by plotting bar charts next to each other, group by whether owner == "spotify" 3. Have to make a decision on what outcome variable to choose (or most likely a combination of outcome variables!) 4. Look at univariate cuts of the outcome variable(s) with the predictors, just to try to give a sense of the relationship between the variables prior to modeling 5. Build a baseline model that we can use for prediction and then we can compare this to a more advanced model – will allow for simpler interpretations 6. Build an ML model that we can then go compare efficacy with the baseline model.

Note: 1. We will likely want to run separate analyses broken out by owner == 'spotify' vs owner != 'spotify' a. See if the importance of the other predictors is comparable for the two groups 2. If a user streams 2 or more songs consecutively, does that count as 2 streams or only 1 since the stream were consecutive?

Possible Options (no particular order) 1. Look into multivariate methods to accommodate the various potential outcome variables (or could just pick one/create a PCA) 2. Should probably think through how to use the 'skippers' data. Could this be at all helpful in dealing with potential biases in how Spotify may promote playlists?

Assumptions: 1. This is a random sample of playlists. If this a biased sample, then any generalizations that we make from the data is likely to be meaningfully inaccurate. 2. Spotify treats each non-Spotify playlist equally in terms of promotion. For examples, if the Spotify algorithms were promoting some genres above others at the time this data was collected, then we are unlikely to get a good read on how genre affects listenership.
3. (a) Spotify treats its own playlists differently than the non-Spotify playlists. If this assumption is correct, then it is likely that Spotify playlists are not particularly comparable to non-Spotify playlists.
(b) Spotify treats its own playlists equally with each other. Thus, an analysis with only Spotify playlists should be okay.
4. Each playlist included in the dataset has existed for at least two months. This ensures that the monthly average users in the previous month variable is not biased by how long the playlist has existed. 5. The Spotify algorithms do not amplify small variations in success. If playlist A was slightly more successful than Playlist B two months ago under 'fair' algorithmic treatment, then the algorithms will not amplify playlist A over playlist B, and thus widen the gulf between the success of the two playlists. In other words, there is a fair marketplace for the playlists to compete, where success does not necessarily beget success simply due to the algorithms. 6. For the categorical variables, genre_1-genre_3 and mood_1-mood_3, when the value is '-' this is not

a missing value, but is instead imparting the information that the given playlist does not easily fit into the predefined genre and mood types.

1 Introduction

The data under consideration for these analyses consists of 403,366 distinct playlists, with 314,899 distinct playlist owners. Of the 314,899 unique playlist owners, 261,040 (83%) have exactly one playlist in the data. Of the owners with more than one playlist, Spotify itself has the most, with 399. The data is composed of only playlists from US owners, and thus extrapolating any of these analyses to other countries is likely unwarranted or should be done with great caution. Each playlist is categorized by its top three genres and top three moods. There are 26 genres and 27 moods under consideration.

There are a number of potential measures of playlist success included in this dataset. Specifically, we have (1) The number of streams from the playlist today, (2) the number of streams greater than 30 seconds today, (3) the number of active users today, where an active user is defined as having a stream $>$ 30 seconds, (4) the number of active users in the past week, (5) the number of active users in the past month, (6) the number of users who had a stream this playlist for any length of time in the past month, (7) the number of active users in the previous month, (8) the total number of $>$ 30 second streams in the past month, (9) the number of users who were active this month and the previous month, and (10) the number of $>$ 30 second streams by the playlist owner in the past month. The data also includes the number of users who skipped more than 90% of their total streams today who also used this playlist, which could be used as a reverse encoded outcome variable.

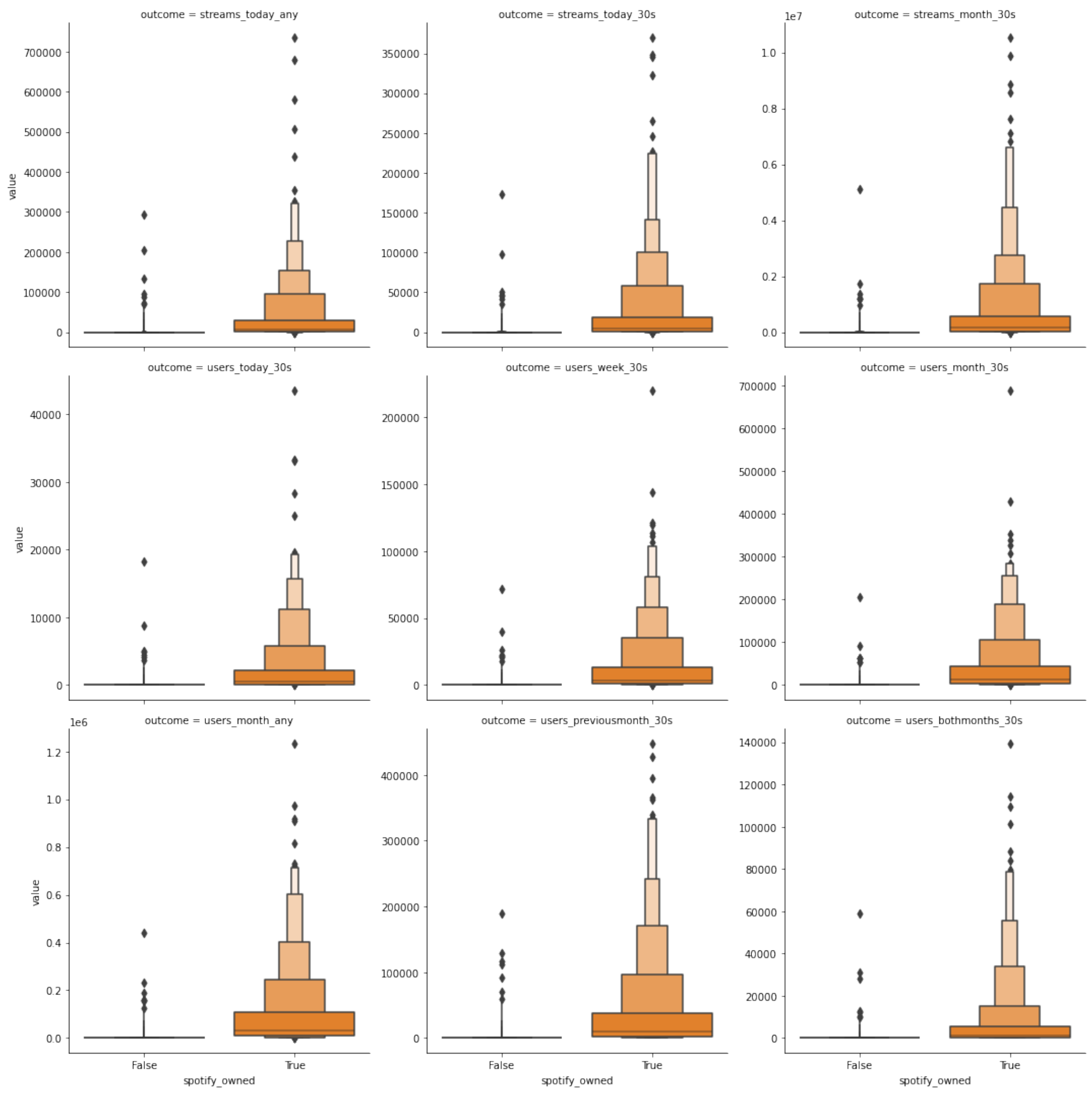
Some of the potential predictors of stream success include: (1) the number of tracks in the playlist, (2) the number of tracks that were added to the playlist today, (3) The number of unique artists in the playlist, (4) the number of unique albums in the playlist, (5-7) the first, second and third most common genre found in the playlist, (8-10) the first, second and third most common mood found in the playlist, and (11) the tokens associated with the playlist.

There are two Spotify-owned playlists that constitute extreme outliers across each of the potential outcome variables. The first is a pop, Dance & House, Indie Rock playlist with 100 tracks and has tokens ‘top’, ‘tracks’, ‘currently’, ‘spotify’. The second is a pop, R&B, Dance & House with 51 tracks and has tokens ‘top’, and ‘hits’. These two playlists have more than three times as many streams today as their nearest competitor and more than four times as many $>$ 30 second streams in the past month as the nearest competitor. For the purposes of plotting, these playlists will be removed.

1.1 Comparing Spotify-Owned and Non-Spotify-Owned Playlists

Even after removing the two most successful Spotify-owned playlists, there is still a wide gulf between the Spotify-owned and non-Spotify-owned playlists. To illustrate this, consider the boxen plot below, which shows the distribution of each of the potential outcome variables stratified by whether the playlist is Spotify-owned. We observe that the Spotify-owned playlists are much more successful than the vast majority of non-Spotify-owned playlists. However, there are a few user-created playlists that can rival an average Spotify-created playlist.

```
<seaborn.axisgrid.FacetGrid at 0x1adb0718ee0>
```

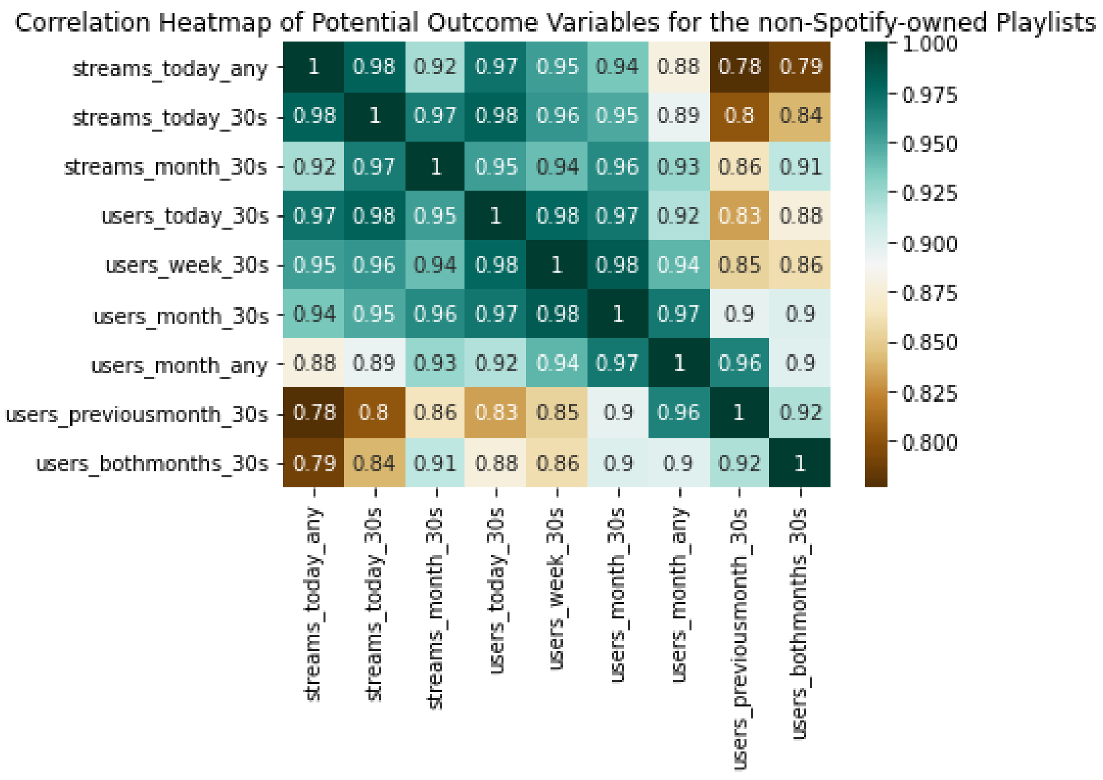


1.2 Exploration of Potential Outcomes

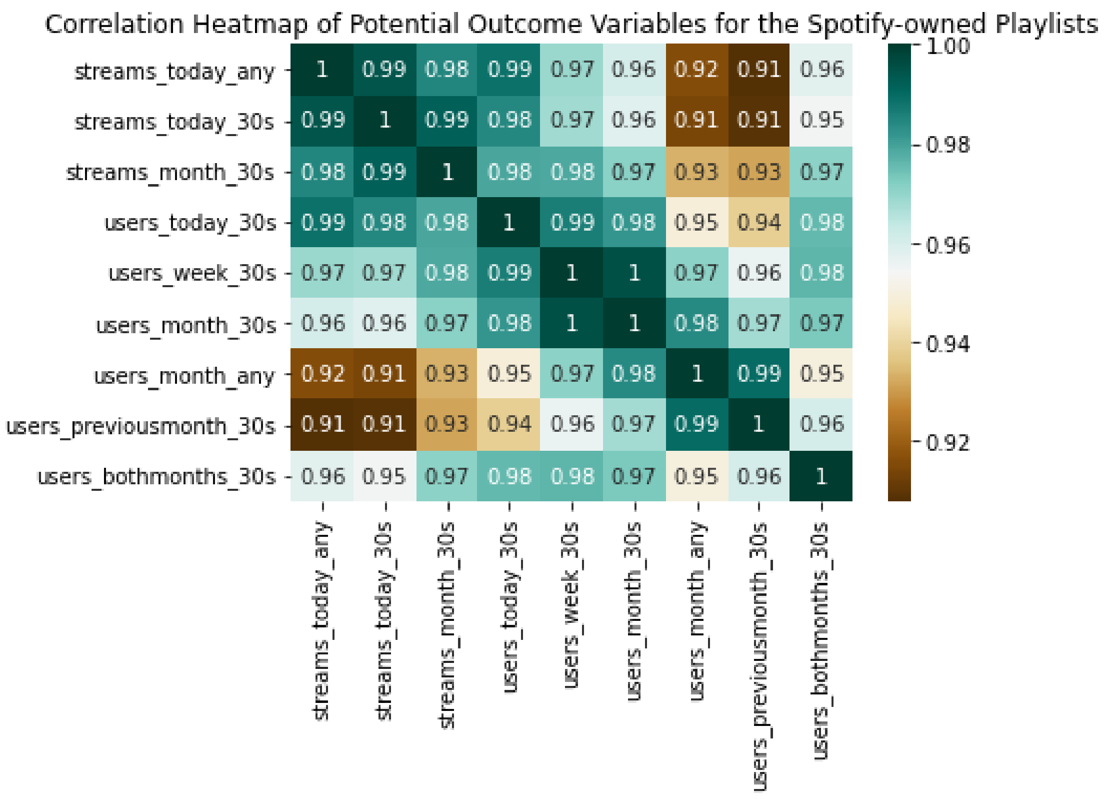
Because there is such an enormous difference between Spotify-created and user-created playlists, we will continue our data exploration stratifying by whether the playlist is Spotify-created. In results not shown, the correlation between the number of >30 second streams by the playlist owner is **much** more weakly correlated with the other potential outcome variables than, and thus will not be included in the following analysis.

To begin understanding the relationship between the potential outcomes of interest, we present a heatmap of the Pearson correlation between each of the outcomes. We create one such heatmap for the non-Spotify-owned playlists and another for the Spotify-owned playlists. Amongst the non-Spotify-owned playlists, we see that the minimum correlation between any of the outcomes is 0.78, thus signifying a great deal of correlation between our potential outcomes. While all potential outcomes are highly correlated, the mostly weakly correlated outcomes are the outcomes related to playlists' longer-term success (i.e. the monthly average users in the given month and the previous month) with the more recent measures of success (i.e. the number of total and >30 second streams today and the number of active users today and in the past week) Amongst the Spotify-owned playlists, the potential outcomes are even more highly correlated, with the smallest correlation being 0.91. The Spotify-owned playlists exhibit a similar general pattern to those of the non-Spotify-owned in that the weakest correlation between the outcomes is between the more long-term measures of success and the measures of success in the more recent past. However, the degree of correlation is still immense between monthly active users over the past two months and the number of streams that occurred today, thus indicating that Spotify playlists tend to have considerable 'staying power'. Of course, if more successful playlists in the past are algorithmically pushed to users, then this could become a self-fulfilling prophecy rather than a true indication of how 'intrinsically good' the palylist is.

```
Text(0.5, 1.0, 'Correlation Heatmap of Potential Outcome Variables for the non-
Spotify-owned Playlists')
```



```
Text(0.5, 1.0, 'Correlation Heatmap of Potential Outcome Variables for the
Spotify-owned Playlists')
```



1.3 Factor Analysis of Potential Outcomes

We will continue our exploration of the relationship between the potential outcomes by performing a factor analysis.

```
Index(['streams_today_any', 'streams_today_30s', 'streams_month_30s',
       'users_today_30s', 'users_week_30s', 'users_month_30s',
       'users_month_any', 'users_previousmonth_30s', 'users_bothmonths_30s'],
      dtype='object')

array([[0.87728143, 0.87016502, 0.76687257, 0.84414808, 0.78935438,
        0.73865839, 0.59523192, 0.43980247, 0.57498533],
       [0.44313307, 0.47061719, 0.5922918 , 0.52229923, 0.589682  ,
        0.66099455, 0.79856375, 0.87838131, 0.71536399]])
```

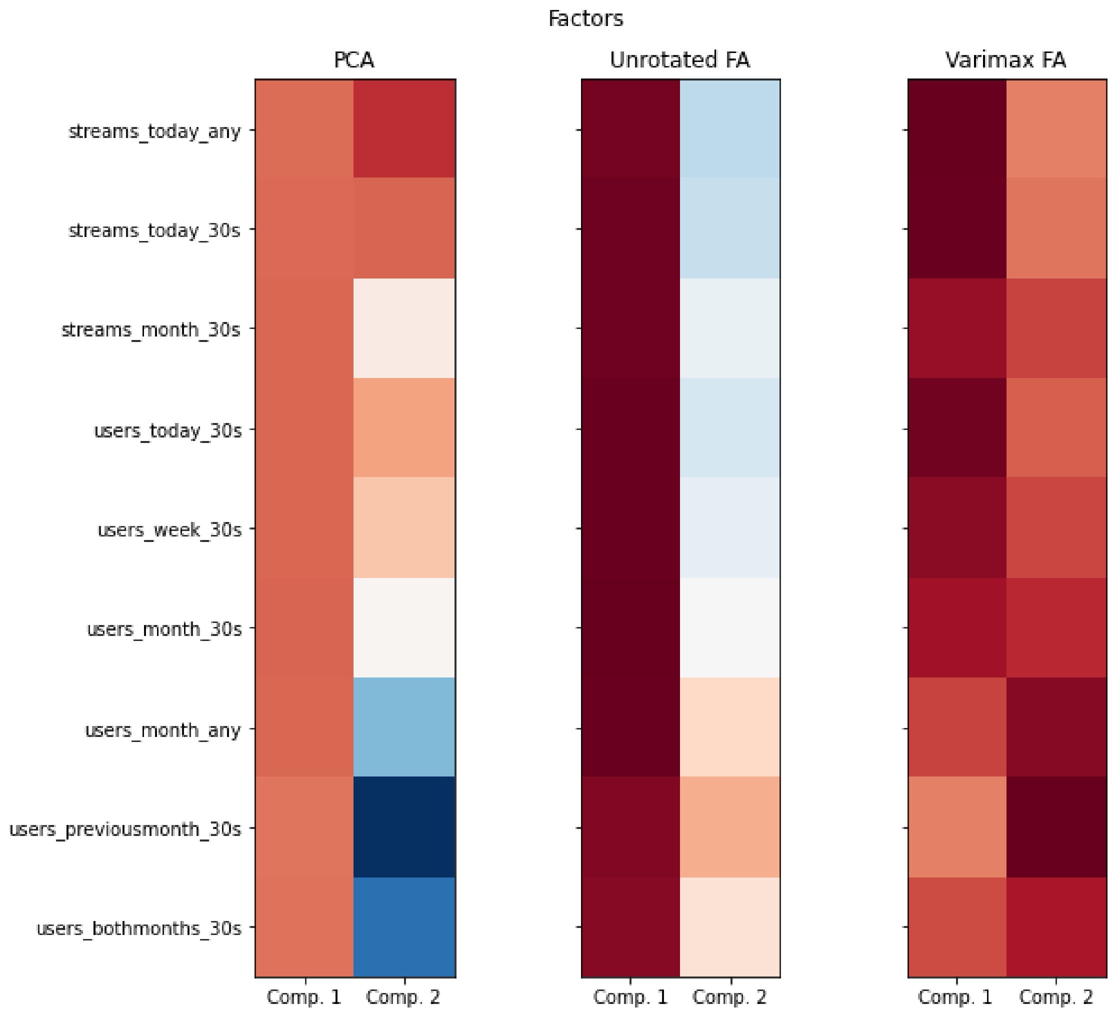
```
array([[0.87728143, 0.87016502, 0.76687257, 0.84414808, 0.78935438,
        0.73865839, 0.59523192, 0.43980247, 0.57498533],
       [0.44313307, 0.47061719, 0.5922918 , 0.52229923, 0.589682  ,
        0.66099455, 0.79856375, 0.87838131, 0.71536399]])
```

```
 PCA :

[[ 0.32924324  0.43471298]
 [ 0.33560123  0.34213424]
 [ 0.33857786  0.04681532]
 [ 0.34001732  0.23790346]
 [ 0.33949301  0.16453315]
 [ 0.34326352  0.01213333]
 [ 0.33617767 -0.26111241]
 [ 0.3164575  -0.58940584]
 [ 0.32012158 -0.43861344]]
```

```
C:\Users\ADMINI~1\AppData\Local\Temp/ipykernel_1912/4147204229.py:25:
MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed
two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
C:\Users\Administrator\OneDrive\anaconda\lib\site-
packages\sklearn\decomposition\_factor_analysis.py:253: ConvergenceWarning:
FactorAnalysis did not converge. You might want to increase the number of
iterations.
  warnings.warn('FactorAnalysis did not converge.' +
C:\Users\ADMINI~1\AppData\Local\Temp/ipykernel_1912/4147204229.py:25:
MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed
two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```

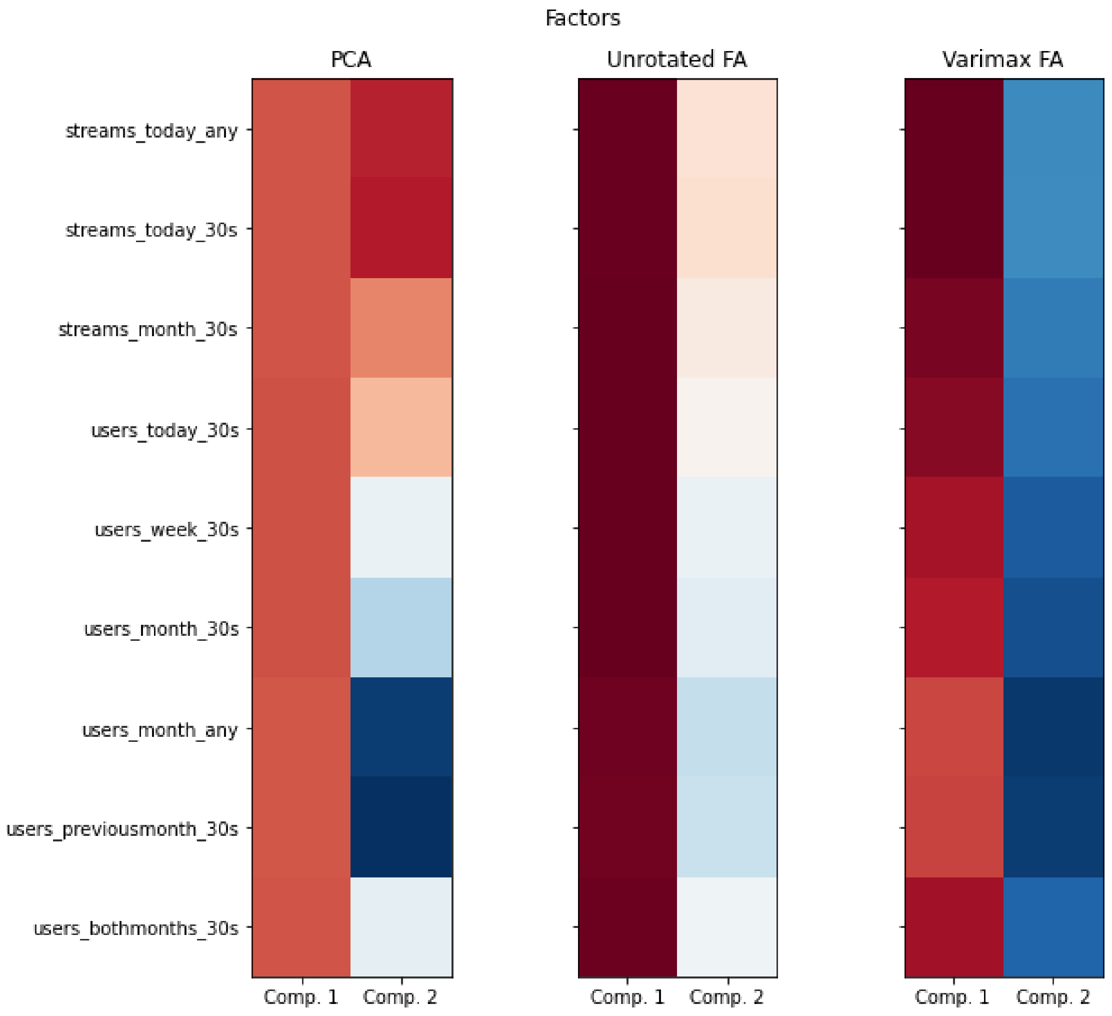
```
 Unrotated FA :

[[ 0.94851763 -0.25749548]
 [ 0.96161405 -0.23230612]
 [ 0.96623397 -0.0727671 ]
 [ 0.97684947 -0.17648679]
 [ 0.98119863 -0.08974654]
 [ 0.99122249 -0.00282654]
 [ 0.97664408  0.19537514]
 [ 0.91450999  0.3586801 ]
 [ 0.90593703  0.14707775]]


 Varimax FA :

[[0.87728143 0.44313307]
 [0.87016502 0.47061719]
 [0.76687257 0.5922918 ]
 [0.84414808 0.52229923]
 [0.78935438 0.589682  ]
 [0.73865839 0.66099455]
 [0.59523192 0.79856375]
 [0.43980247 0.87838131]
 [0.57498533 0.71536399]]
```

```
C:\Users\Administrator\OneDrive\anaconda\lib\site-
packages\sklearn\decomposition\_factor_analysis.py:253: ConvergenceWarning:
FactorAnalysis did not converge. You might want to increase the number of
iterations.
  warnings.warn('FactorAnalysis did not converge.' +
C:\Users\ADMINI~1\AppData\Local\Temp/ipykernel_1912/4147204229.py:25:
MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed
two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```



```
 PCA :

[[ 0.33229849  0.40845353]
 [ 0.33215236  0.41852091]
 [ 0.33466415  0.25973456]
 [ 0.33629116  0.17191397]
 [ 0.33694957 -0.03375294]
 [ 0.33658998 -0.15683509]
 [ 0.32946288 -0.5021951 ]
 [ 0.32769201 -0.52979451]
 [ 0.33377389 -0.04869153]]


 Unrotated FA :

[[ 0.9856163   0.14749079]
 [ 0.98527459  0.15764781]
 [ 0.98928245  0.08554343]
 [ 0.99390694  0.03763906]
 [ 0.99381397 -0.06493129]
 [ 0.99147195 -0.11568914]
 [ 0.96470904 -0.24012516]
 [ 0.95519136 -0.22284167]
 [ 0.97821387 -0.04261088]]


 Varimax FA :

[[ 0.83754815 -0.54009825]
 [ 0.84399878 -0.53224515]
 [ 0.79939399 -0.58903878]
 [ 0.77123286 -0.62806654]
 [ 0.70343017 -0.70503069]
 [ 0.66815321 -0.7416009 ]
 [ 0.56588353 -0.81737351]
 [ 0.57014944 -0.79810937]
 [ 0.70645462 -0.67796749]]
C:\Users\ADMINI~1\AppData\Local\Temp/ipykernel_1912/4147204229.py:25:
MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed
two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
C:\Users\Administrator\OneDrive\anaconda\lib\site-
packages\sklearn\decomposition\_factor_analysis.py:253: ConvergenceWarning:
FactorAnalysis did not converge. You might want to increase the number of
```

```
iterations.
  warnings.warn('FactorAnalysis did not converge.' +
C:\Users\ADMINI~1\AppData\Local\Temp/ipykernel_1912/4147204229.py:25:
MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed
two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
C:\Users\Administrator\OneDrive\anaconda\lib\site-
packages\sklearn\decomposition\_factor_analysis.py:253: ConvergenceWarning:
FactorAnalysis did not converge. You might want to increase the number of
iterations.
  warnings.warn('FactorAnalysis did not converge.' +
C:\Users\ADMINI~1\AppData\Local\Temp/ipykernel_1912/4147204229.py:25:
MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed
two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```



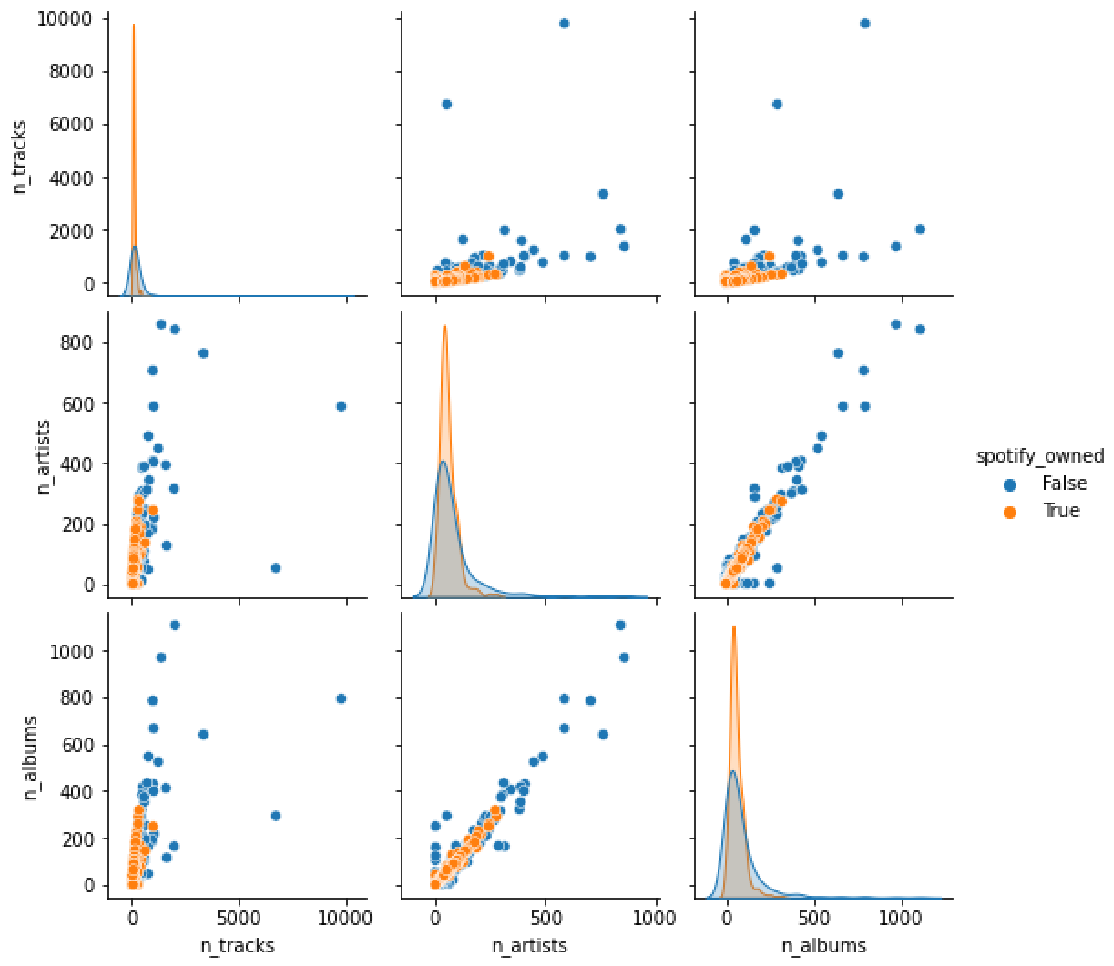
1.4 Exploration of Predictors of Interest

```
   spotify_owned  n_tracks  n_local_tracks  n_artists  n_albums  \
0          False        52               0          4         7
1          False       131               0        112       113
2          False        43               0         35        36
3          False        27               1         27        26
4          False        52               0         47        51

        genre_1      genre_2          genre_3    mood_1      mood_2    mood_3
0  Dance & House      New Age  Country & Folk  Peaceful    Romantic    Somber
1            Pop   Indie Rock     Alternative   Excited   Yearning   Defiant
2          Latin            -               -    Lively      Upbeat  Romantic
3  Dance & House  Electronica             Pop   Excited  Aggressive   Defiant
4     Indie Rock  Alternative     Electronica   Excited     Defiant  Yearning
```

```
            n_tracks  n_local_tracks      n_artists       n_albums
count  403366.000000   403366.000000  403366.000000  403366.000000
mean      201.483432        3.084035      83.852050      88.224250
std       584.077765       40.330266     128.152488     133.193118
min         1.000000        0.000000       1.000000       1.000000
25%        38.000000        0.000000      18.000000      19.000000
50%        84.000000        0.000000      46.000000      48.000000
75%       192.000000        0.000000     100.000000     106.000000
max     79984.000000     9117.000000    5226.000000    6397.000000
```

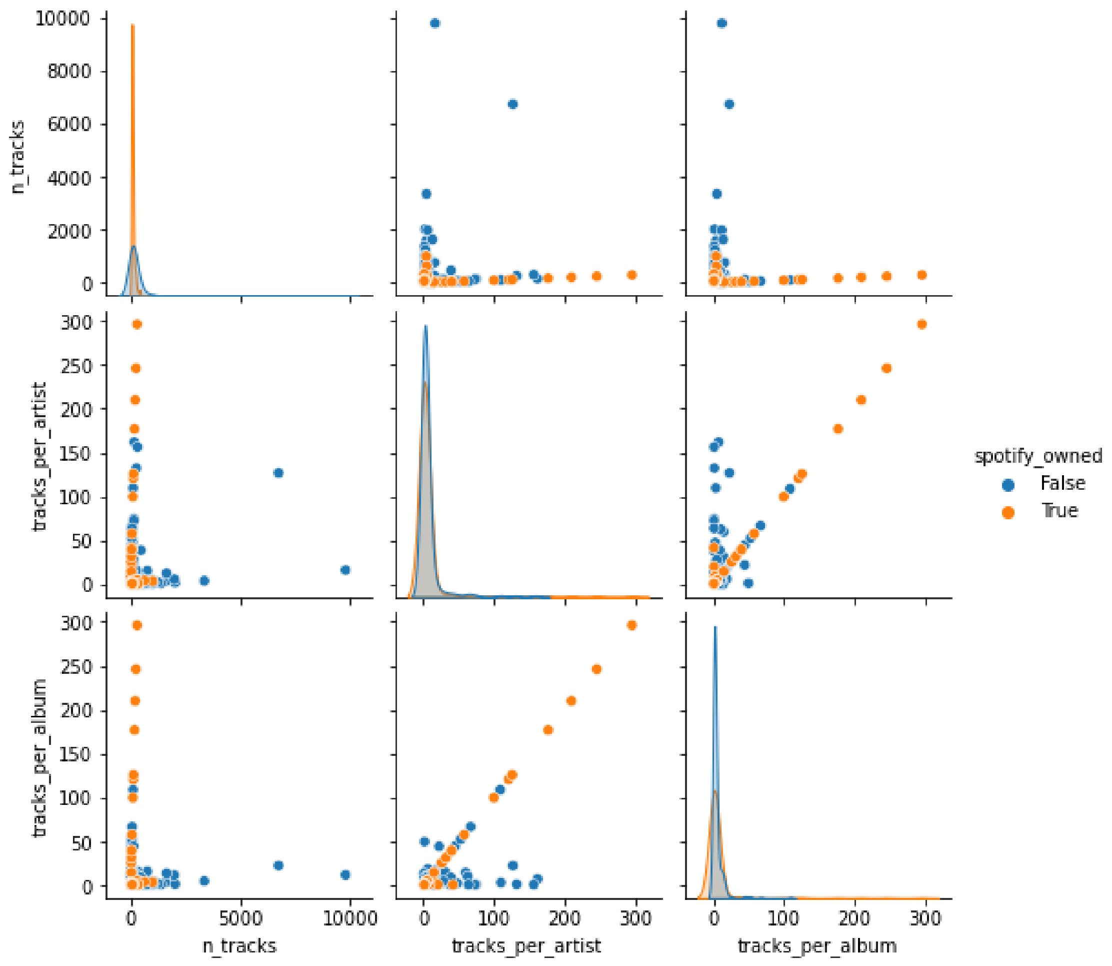
Note that for the plots below, we drew a random sample of user-created playlists for these plots in order for them not to visually swamp the Spotify-created playlists. We can see from the plot below that, unsurprisingly, as a playlist's number of tracks increases so do the number of artists and albums. We will thus define tracks per artist and tracks per album as scaled measures of the number of artists and albums that account for the overall size of the playlist. Also, note that the distribution of number of tracks, artists and albums does not vary between Spotify-created and user-created to the degree that the outcome variables do. However, we still observe some difference between user-created and Spotify-created playlists, namely that user-created playlists tend to have moer tracks, artists and albums, with some user-created playlists having many more of each.

```
<seaborn.axisgrid.PairGrid at 0x1adf493c730>
```



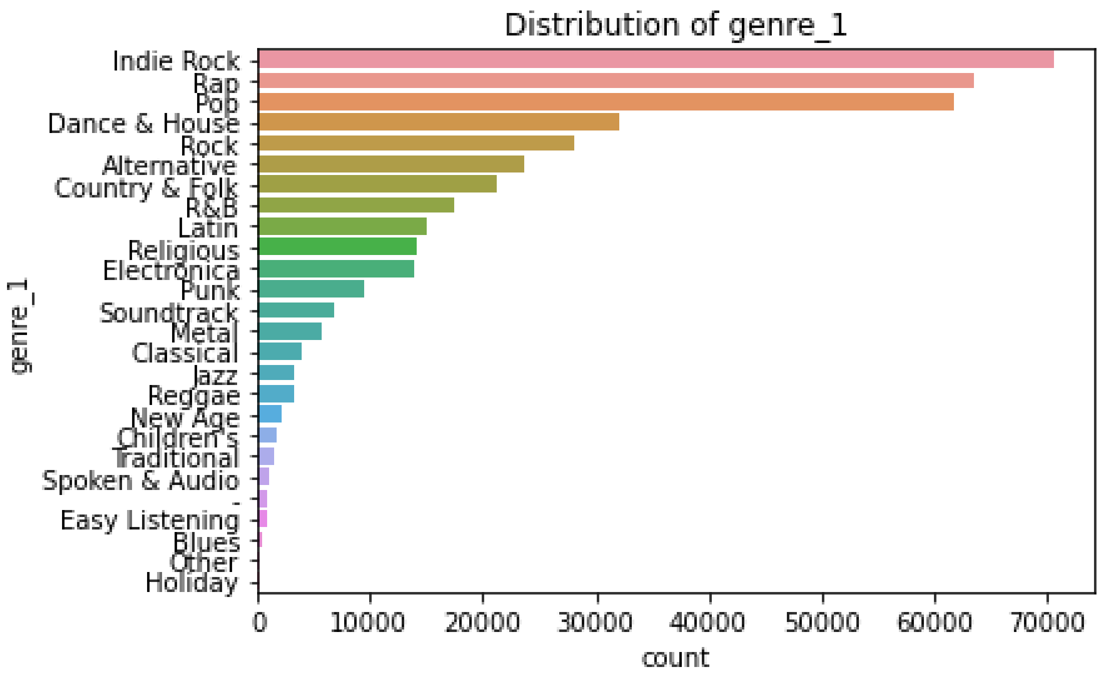
Let's consider the correlation of the new scaled measures of artists and albums as well as whether we observe differences between the user-created and Spotify-created playlists. Notice that the distribution of tracks per artist and tracks per album appear quite similar for the user-created and Spotify-created playlists. Spotify-created playlist appear to exhibit a little bit more artist and album diversity than the user-created playlists, but the difference is miniscule compared to the outcome variables.

```
<seaborn.axisgrid.PairGrid at 0x1adf49e35e0>
```

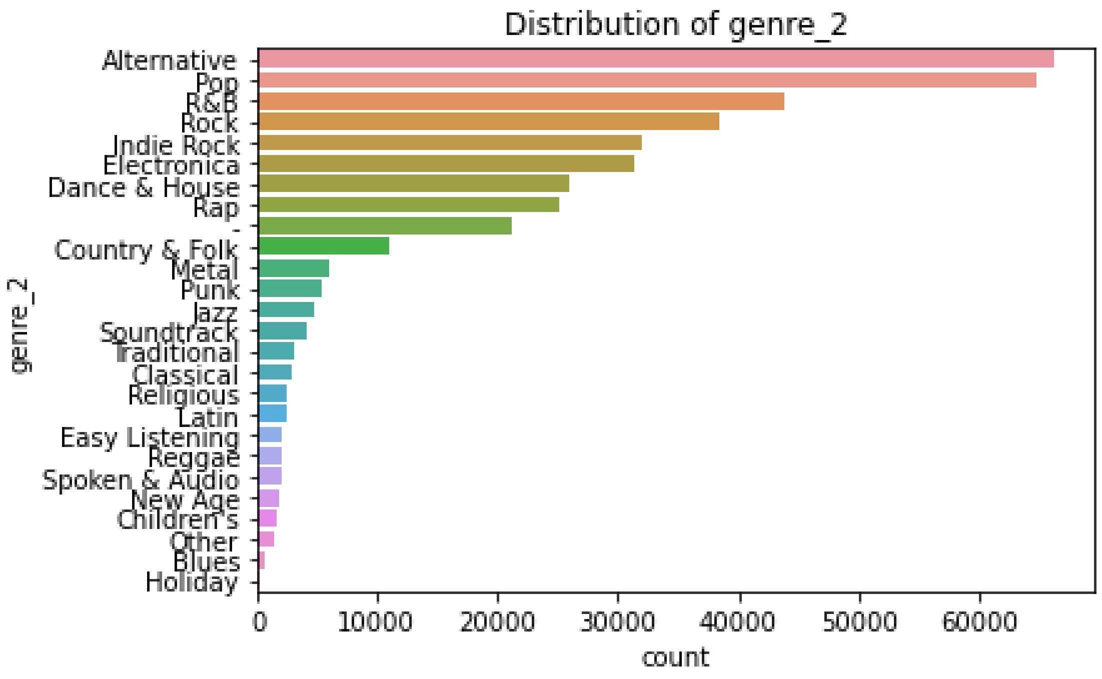


We now turn our attention to describing the categorical predictors: genre and mood.

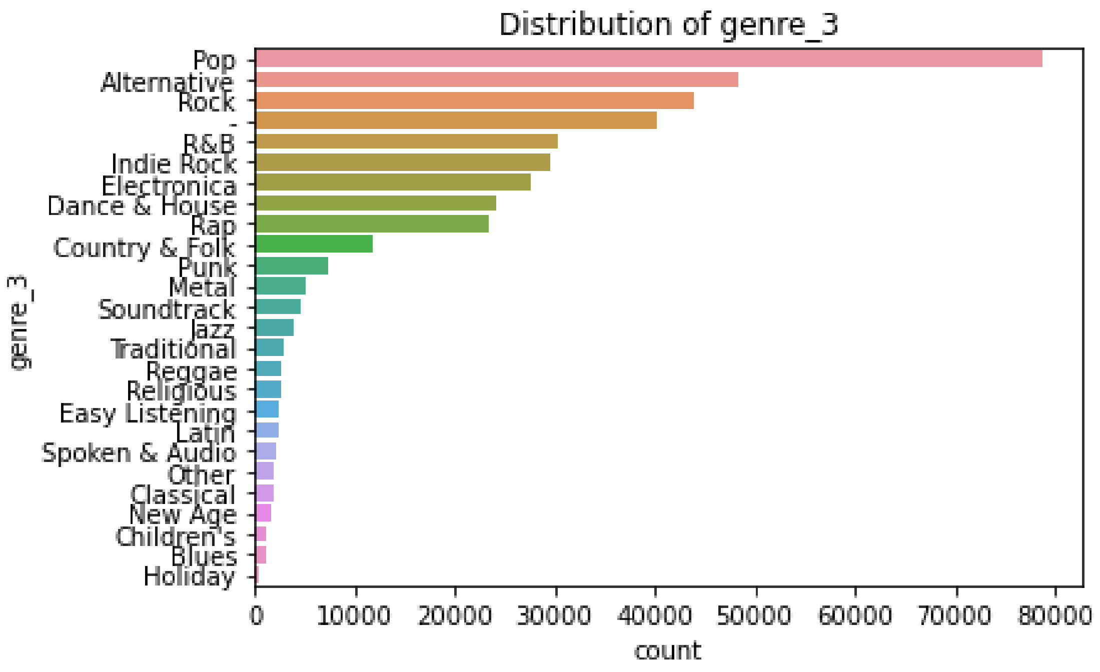
```
<AxesSubplot:title={'center':'Distribution of genre_1'}, xlabel='count',
ylabel='genre_1'>
```



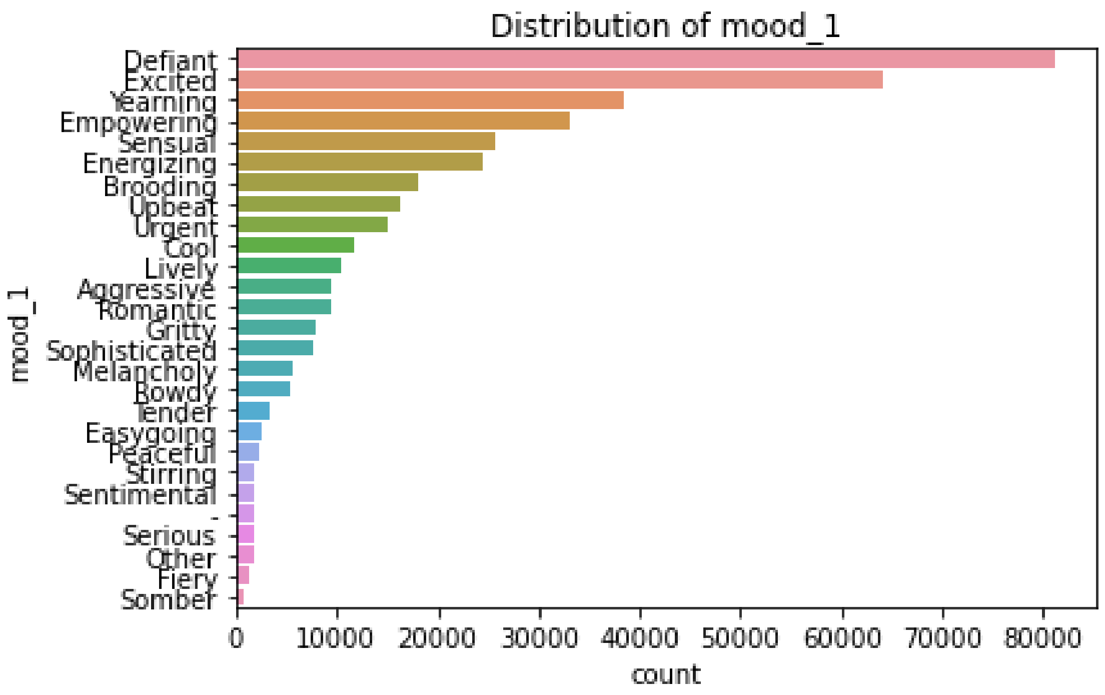
```
<AxesSubplot:title={'center':'Distribution of genre_2'}, xlabel='count',
ylabel='genre_2'>
```



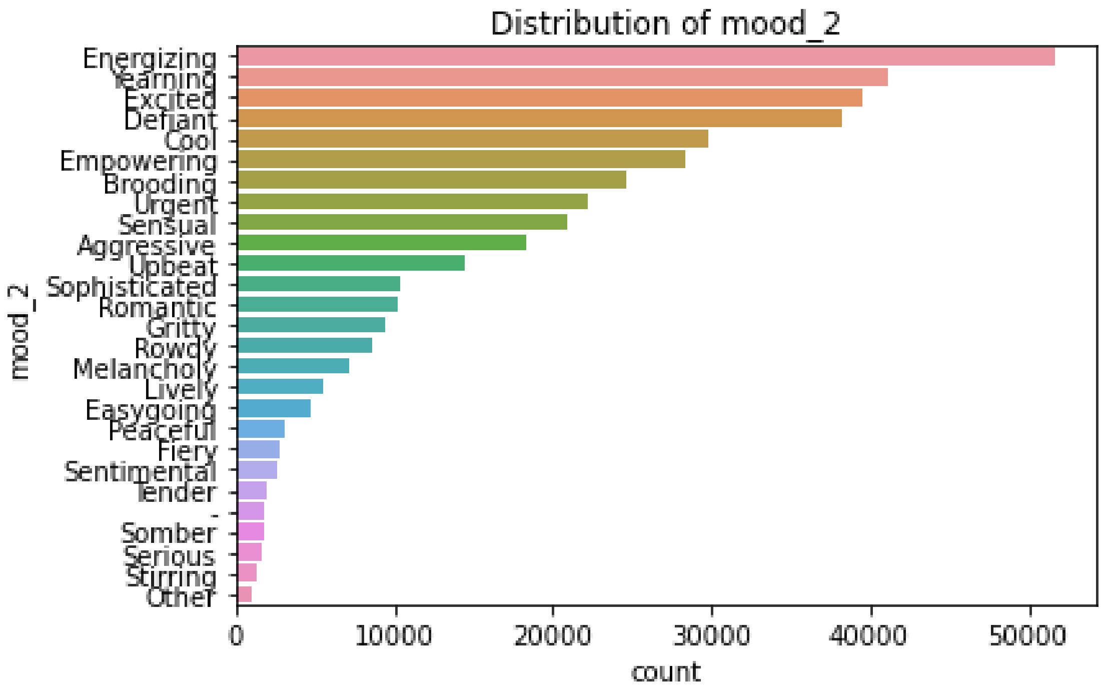
```
<AxesSubplot:title={'center':'Distribution of genre_3'}, xlabel='count',
ylabel='genre_3'>
```



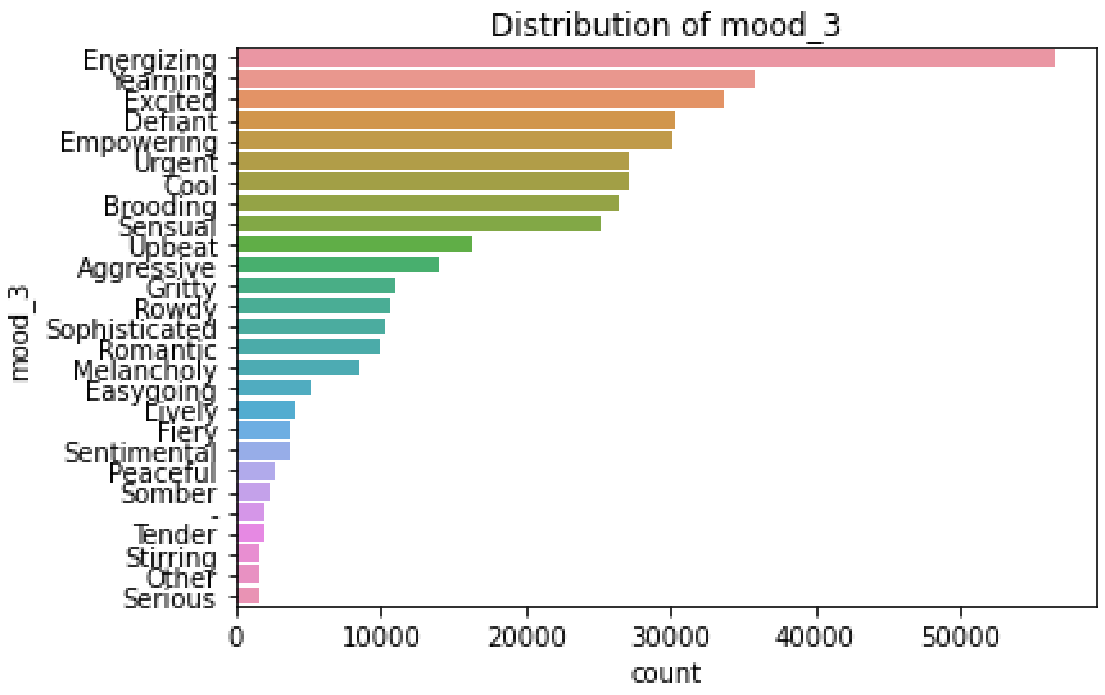
```
<AxesSubplot:title={'center':'Distribution of mood_1'}, xlabel='count',
ylabel='mood_1'>
```



```
<AxesSubplot:title={'center':'Distribution of mood_2'}, xlabel='count',
ylabel='mood_2'>
```



```
<AxesSubplot:title={'center':'Distribution of mood_3'}, xlabel='count',
ylabel='mood_3'>
```



We now need to get a sklearn pipeline built for modeling the various outcomes. We will try to predict the outcomes 'users_today_30s' and 'users_bothmonths_30s'. We choose 'users_today_30s' because amongst the user-created playlists, it very highly correlated with streams_today_any, streams_today_30s, streams_month_30s, users_week_30s, and users_month_30s (with the minimum correlation being 0.95). It is also less correlated with users_previousmonth_30s (0.83) and users_bothmonths_30s (0.88). We choose 'users_bothmonths_30s' because (1) it is less corelated than many of the other outcomes and (2) it is a good measure of a playlist's longer-term staying power.

```
C:\Users\Administrator\OneDrive\anaconda\lib\site-
packages\sklearn\model_selection\_split.py:666: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(("The least populated class in y has only %d"
```

```
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.008500 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363029, number of used
features: 11
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of
```

```
testing was 0.002152 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363029, number of used
features: 11
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.008984 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363029, number of used
features: 11
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.009627 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363029, number of used
features: 11
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.009610 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363029, number of used
features: 11
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.009999 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363029, number of used
features: 11
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.010194 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363030, number of used
features: 11
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.009452 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363030, number of used
features: 11
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.009437 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363030, number of used
features: 11
```

```
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.002662 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 363030, number of used
features: 11
[LightGBM] [Info] Start training from score 4.091067
[LightGBM] [Info] Start training from score 4.515138
[LightGBM] [Info] Start training from score 4.006371
[LightGBM] [Info] Start training from score 4.543609
[LightGBM] [Info] Start training from score 4.572855
[LightGBM] [Info] Start training from score 4.297445
[LightGBM] [Info] Start training from score 4.576591
[LightGBM] [Info] Start training from score 4.494876
[LightGBM] [Info] Start training from score 4.553051
[LightGBM] [Info] Start training from score 4.531648
```

```
C:\Users\Administrator\OneDrive\anaconda\lib\site-
packages\lightgbm\engine.py:620: UserWarning: 'verbose_eval' argument is
deprecated and will be removed in a future release of LightGBM. Pass
'log_evaluation()' callback via 'callbacks' argument instead.
  _log_warning("'verbose_eval' argument is deprecated and will be removed in a
future release of LightGBM. "
```

```
[100]   cv_agg's huber: 3.62114 + 1.62474
[200]   cv_agg's huber: 3.59739 + 1.62517
[300]   cv_agg's huber: 3.58196 + 1.62516
[400]   cv_agg's huber: 3.56924 + 1.62537
[500]   cv_agg's huber: 3.55954 + 1.62392
[600]   cv_agg's huber: 3.55125 + 1.62503
[700]   cv_agg's huber: 3.54561 + 1.62591
[800]   cv_agg's huber: 3.54243 + 1.62725
[900]   cv_agg's huber: 3.53945 + 1.628
[1000]  cv_agg's huber: 3.53771 + 1.62952
[1100]  cv_agg's huber: 3.53538 + 1.63099
[1200]  cv_agg's huber: 3.53353 + 1.63175
[1300]  cv_agg's huber: 3.53245 + 1.63217
[1400]  cv_agg's huber: 3.53163 + 1.63321
[1500]  cv_agg's huber: 3.53113 + 1.63465
```

```
C:\Users\Administrator\OneDrive\anaconda\lib\site-
packages\lightgbm\engine.py:239: UserWarning: 'verbose_eval' argument is
deprecated and will be removed in a future release of LightGBM. Pass
'log_evaluation()' callback via 'callbacks' argument instead.
  _log_warning("'verbose_eval' argument is deprecated and will be removed in a
future release of LightGBM. "
```

```
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.010071 seconds.
```

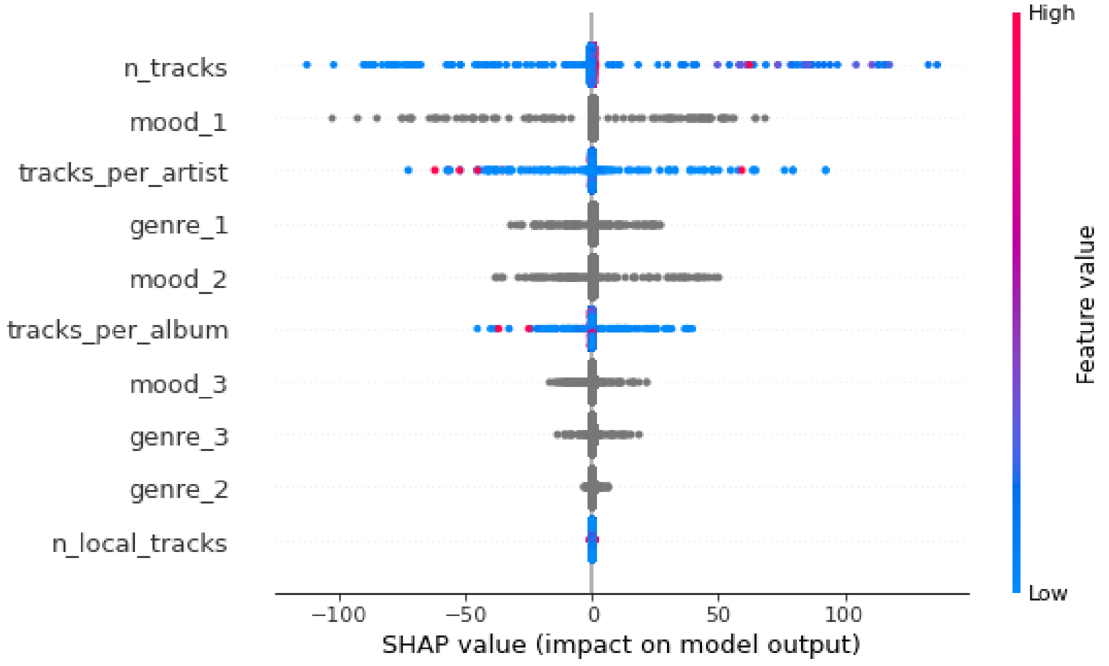
```
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1181
[LightGBM] [Info] Number of data points in the train set: 403366, number of used
features: 11
[LightGBM] [Info] Start training from score 4.418265
```

```
<IPython.core.display.HTML object>
```

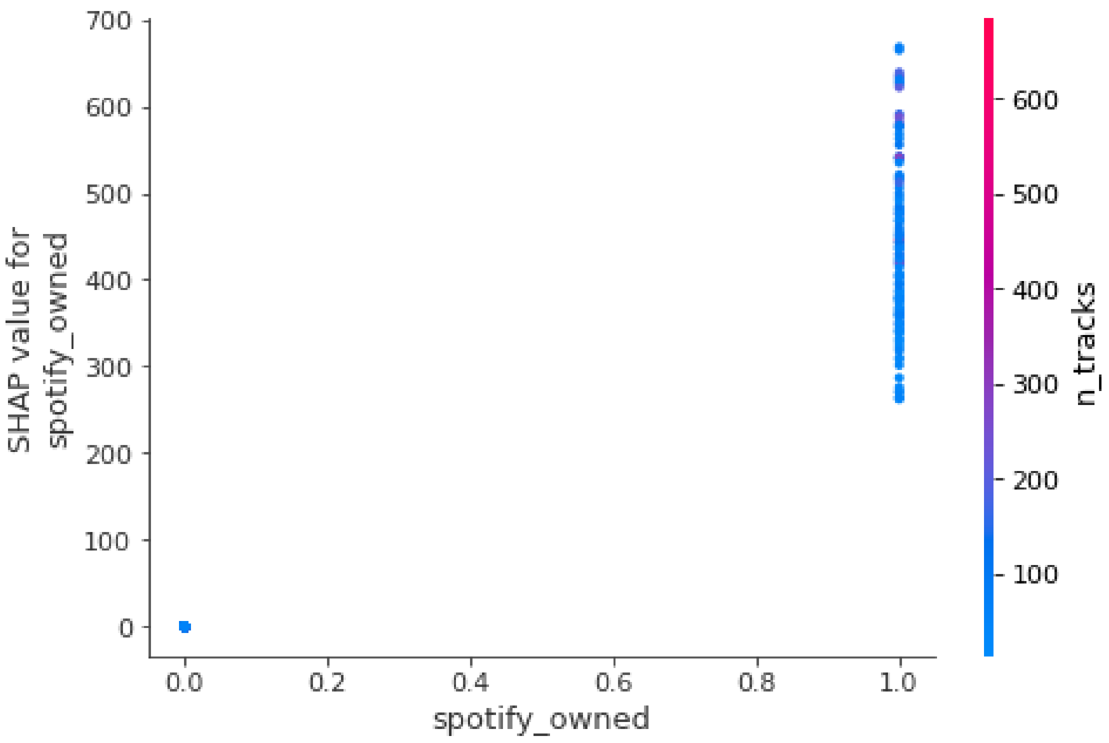
```
 <shap.plots._force.AdditiveForceVisualizer at 0x1aed4bf7970>
```

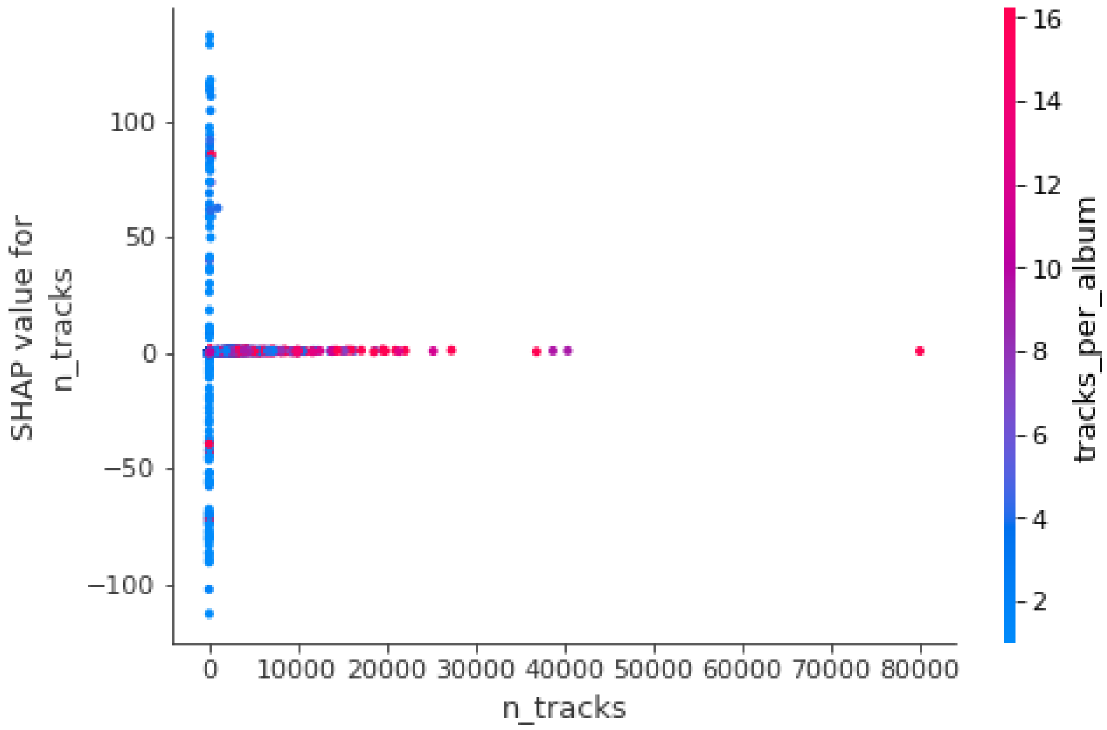
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\ADMINI~1\AppData\Local\Temp/ipykernel_1912/2769655060.py in <module>
----> 1 shap.bar_plot(shap_values, X_reduced)

~\OneDrive\anaconda\lib\site-packages\shap\plots\_bar.py in␣
 ↪bar_legacy(shap_values, features, feature_names, max_display, show)
    372         y_pos, shap_values[feature_inds],
    373         0.7, align='center',
--> 374         color=[colors.red_rgb if shap_values[feature_inds[i]] > 0 else␣
 ↪colors.blue_rgb for i in range(len(y_pos))]
    375     )
    376     pl.yticks(y_pos, fontsize=13)

~\OneDrive\anaconda\lib\site-packages\shap\plots\_bar.py in <listcomp>(.0)
    372         y_pos, shap_values[feature_inds],
    373         0.7, align='center',
--> 374         color=[colors.red_rgb if shap_values[feature_inds[i]] > 0 else␣
 ↪colors.blue_rgb for i in range(len(y_pos))]
    375     )
    376     pl.yticks(y_pos, fontsize=13)

ValueError: The truth value of an array with more than one element is ambiguous ␣
 ↪Use a.any() or a.all()
```

Note that I have deleted the spotify_owned from these plots because its effect makes all other variables effect hard to see.

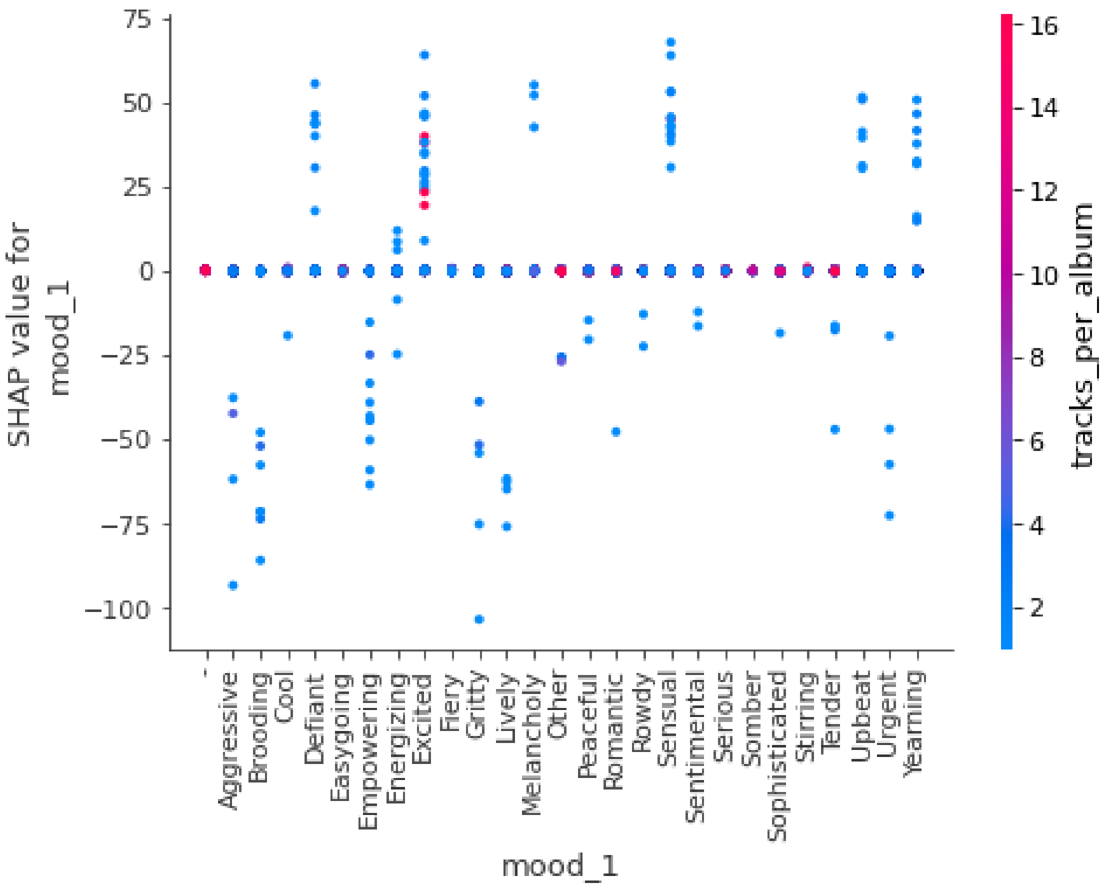

n_tracks
mood_1
tracks_per_artist
genre_1
mood_2
tracks_per_album
mood_3
genre_3
genre_2
n_local_tracks
−100
−50
0
50
100
SHAP value (impact on model output)
High
Feature value
Low

11

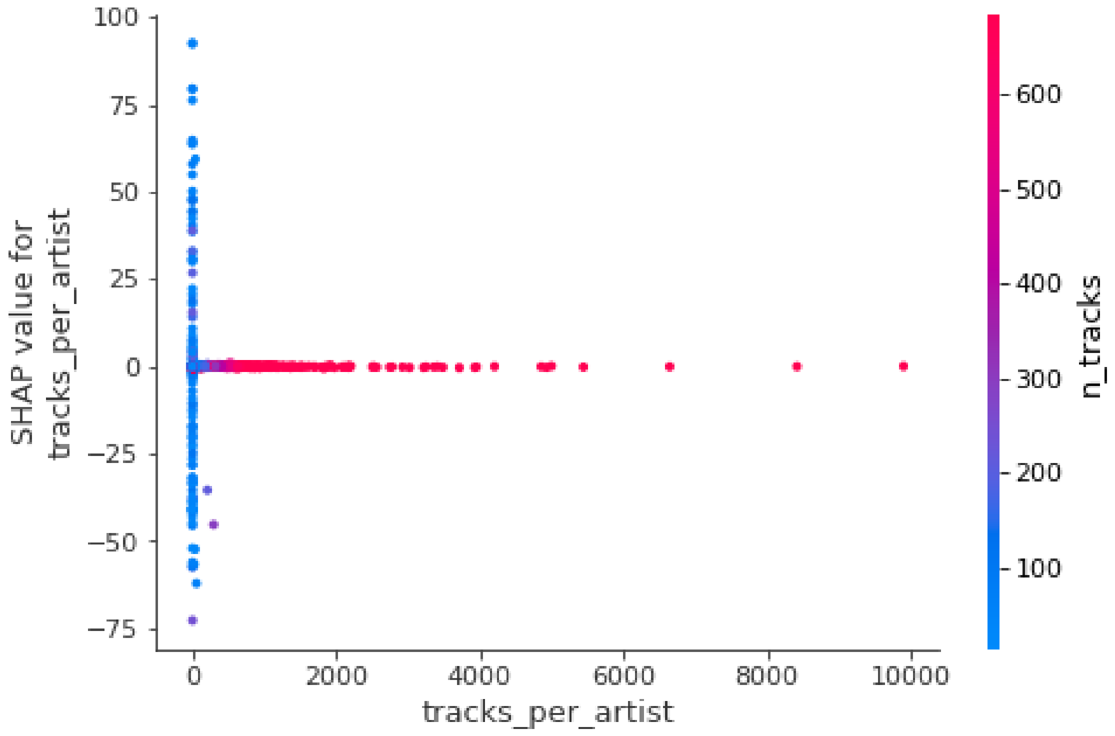

700
600
500
400
300
200
100
0
SHAP value for spotify_owned
0.0
0.2
0.4
0.6
0.8
1.0
spotify_owned
600
500
400
300
200
100
n_tracks



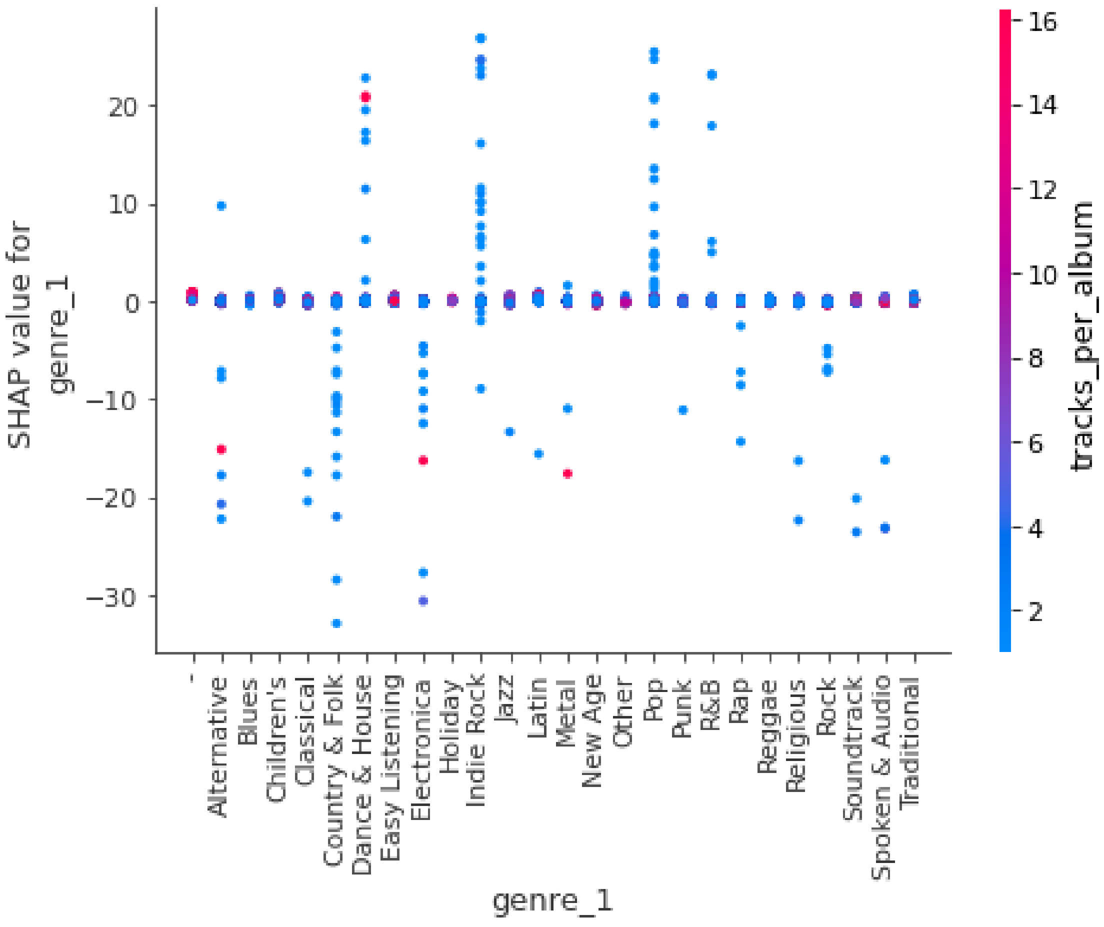
```
Passing the fontdict parameter of _set_ticklabels() positionally is deprecated
since Matplotlib 3.3; the parameter will become keyword-only two minor releases
later.
```



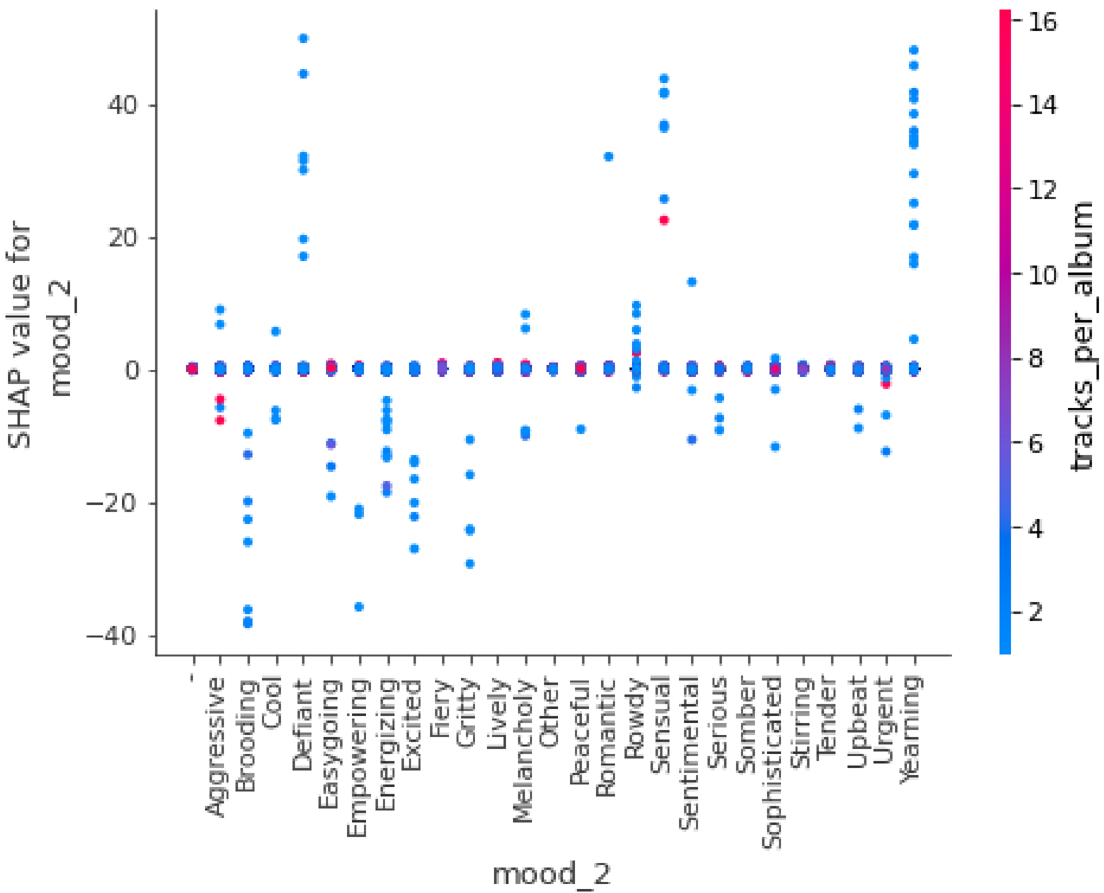
SHAP value for mood_1
75
50
25
0
−25
−50
−75
−100
'
Aggressive
Brooding
Cool
Defiant
Easygoing
Empowering
Energizing
Excited
Fiery
Gritty
Lively
Melancholy
Other
Peaceful
Romantic
Rowdy
Sensual
Sentimental
Serious
Somber
Sophisticated
Stirring
Tender
Upbeat
Urgent
Yearning
mood_1
16
14
12
10
8
6
4
2
tracks_per_album

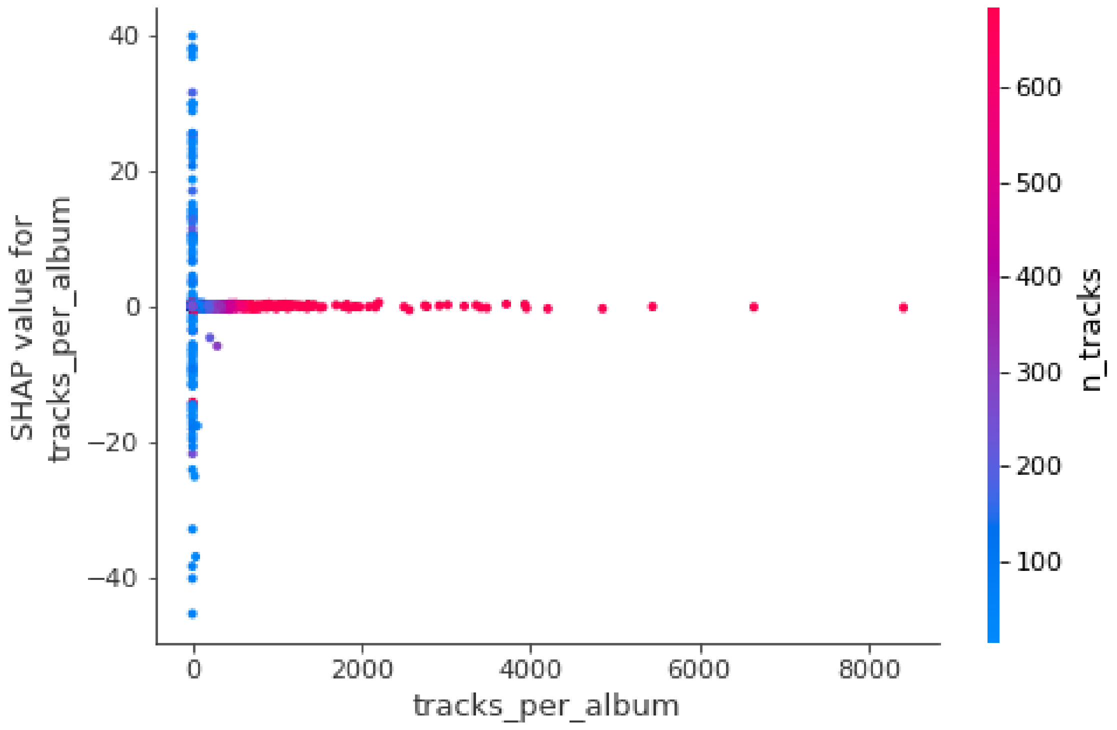


```
Passing the fontdict parameter of _set_ticklabels() positionally is deprecated
since Matplotlib 3.3; the parameter will become keyword-only two minor releases
later.
```

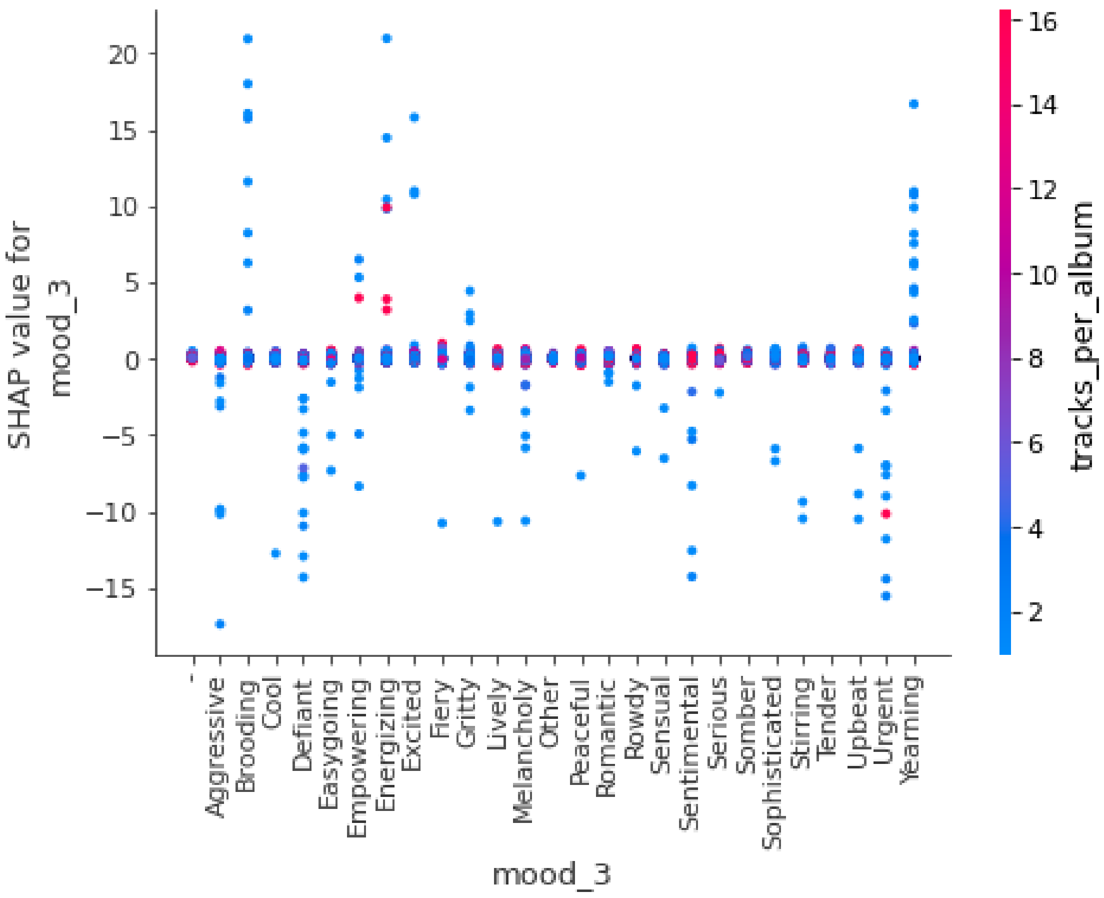


```
Passing the fontdict parameter of _set_ticklabels() positionally is deprecated
since Matplotlib 3.3; the parameter will become keyword-only two minor releases
later.
```

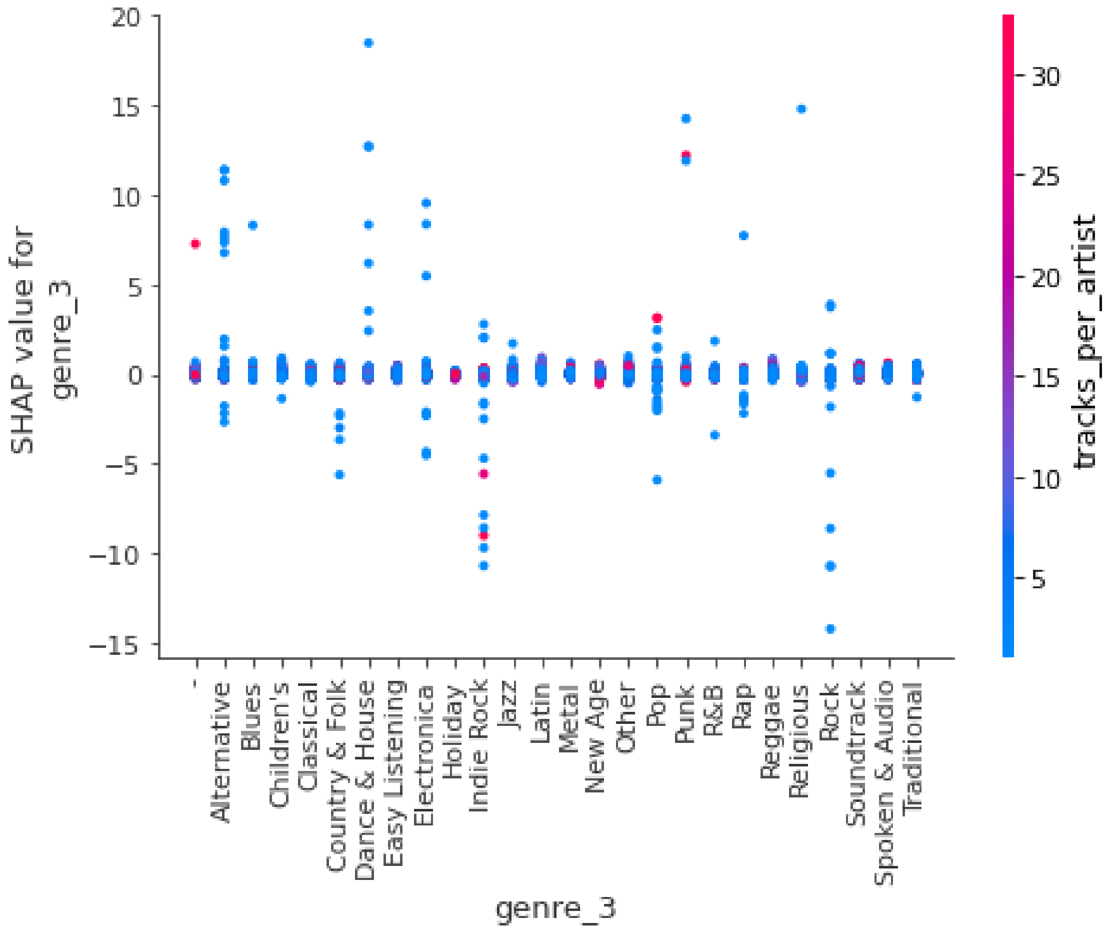

SHAP value for mood_2
40
20
0
−20
−40
-
Aggressive
Brooding
Cool
Defiant
Easygoing
Empowering
Energizing
Excited
Fiery
Gritty
Lively
Melancholy
Other
Peaceful
Romantic
Rowdy
Sensual
Sentimental
Serious
Somber
Sophisticated
Stirring
Tender
Upbeat
Urgent
Yearning
mood_2
16
14
12
10
8
6
4
2
tracks_per_album



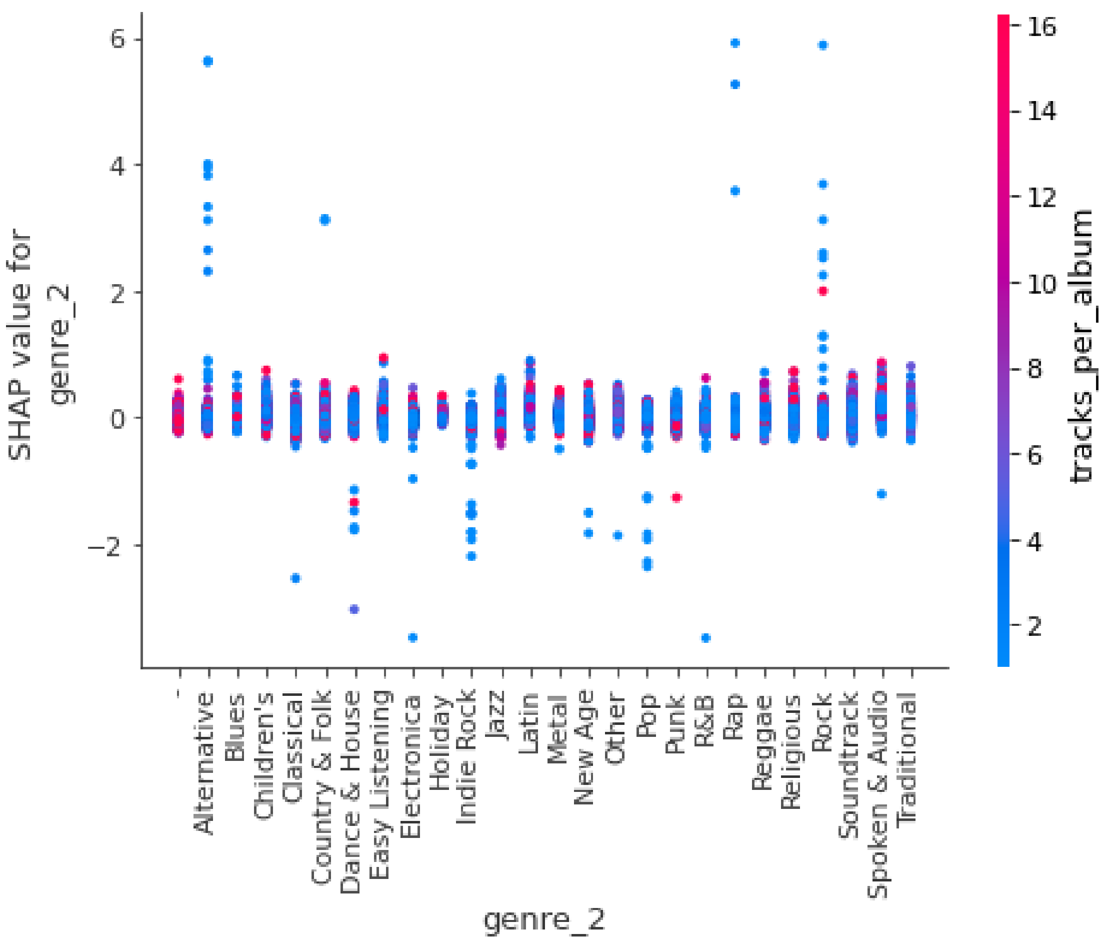
```
Passing the fontdict parameter of _set_ticklabels() positionally is deprecated
since Matplotlib 3.3; the parameter will become keyword-only two minor releases
later.
```

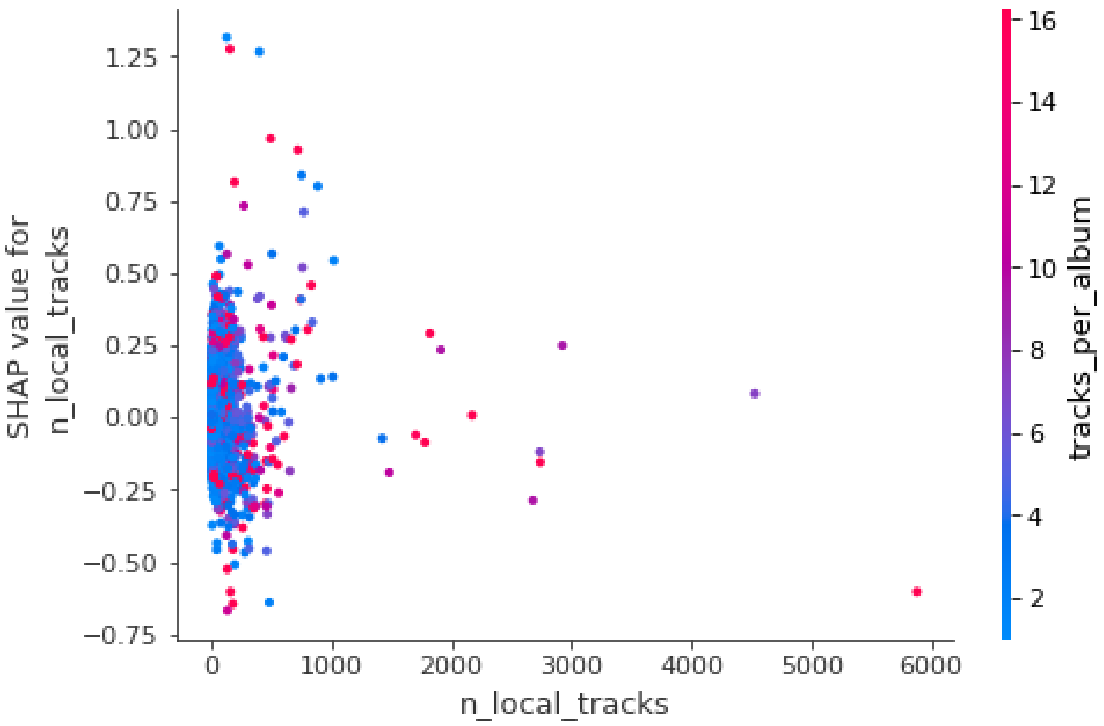


```
Passing the fontdict parameter of _set_ticklabels() positionally is deprecated
since Matplotlib 3.3; the parameter will become keyword-only two minor releases
later.
```



```
Passing the fontdict parameter of _set_ticklabels() positionally is deprecated
since Matplotlib 3.3; the parameter will become keyword-only two minor releases
later.
```


SHAP value for genre_2
6
4
2
0
−2
genre_2
-
Alternative
Blues
Children's
Classical
Country & Folk
Dance & House
Easy Listening
Electronica
Holiday
Indie Rock
Jazz
Latin
Metal
New Age
Other
Pop
Punk
R&B
Rap
Reggae
Religious
Rock
Soundtrack
Spoken & Audio
Traditional
tracks_per_album
16
14
12
10
8
6
4
2


SHAP value for n_local_tracks
n_local_tracks
tracks_per_album
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
1000
2000
3000
4000
5000
6000
16
14
12
10
8
6
4
2