KKbox Music Recommendation

Group 6: Leo Li, Suzy Gao, Yuting Gong, Yu Yang, Zhilin Yang

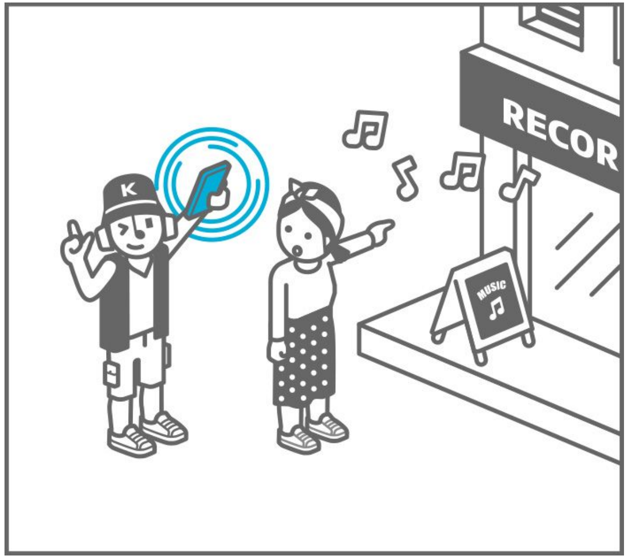
Agenda

- Introduce KKbox & Problem Definition
- Dataset Overview (size, variables)
- Data Inspection & Cleaning & Visualization
- Analysis (models, insights)
- Conclusions & Recommendations

KKbox Music

About KKbox:
KKbox is a leading music streaming company in Taiwan since 2004, holding the world's most comprehensive Asia-Pop music library with over 30 million tracks.

The service area mainly targeting the music market of Southeast Asia, focusing on regions including: Taiwan, Hong Kong, Malaysia, Singapore, etc. It is working on a freemium basis, and in 2015 it announced to have more than 10 million users, with over 1600 artists in its music base.



Problem Definition

kkbox

User preferences → How do users get access to new songs?

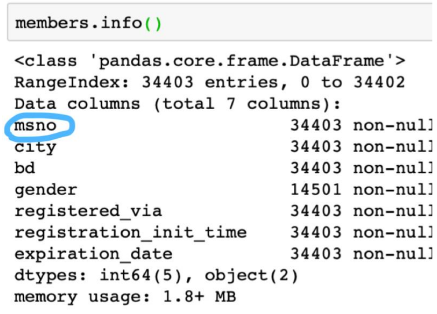
What type of songs do users prefer? Languages? Artists?

Recommendation systems → How do we know if listeners will like a new song?

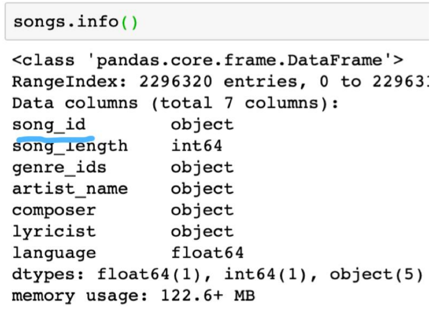
What song should be recommended to a certain user?

Dataset Overview

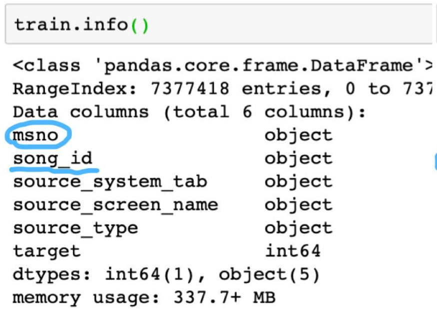
There are five datasets: train, test, songs, members, song_extra_info.



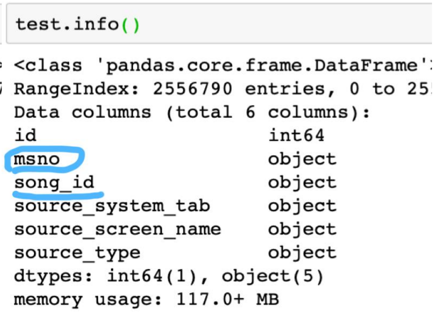
```
members.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34403 entries, 0 to 34402
Data columns (total 7 columns):
msno                      34403 non-nul
city                      34403 non-nul
bd                        34403 non-nul
gender                    14501 non-nul
registered_via            34403 non-nul
registration_init_time    34403 non-nul
expiration_date           34403 non-nul
dtypes: int64(5), object(2)
memory usage: 1.8+ MB
```



```
songs.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2296320 entries, 0 to 22963
Data columns (total 7 columns):
song_id        object
song_length    int64
genre_ids      object
artist_name    object
composer       object
lyricist       object
language       float64
dtypes: float64(1), int64(1), object(5)
memory usage: 122.6+ MB
```



```
train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7377418 entries, 0 to 737
Data columns (total 6 columns):
msno                  object
song_id               object
source_system_tab     object
source_screen_name    object
source_type           object
target                int64
dtypes: int64(1), object(5)
memory usage: 337.7+ MB
```



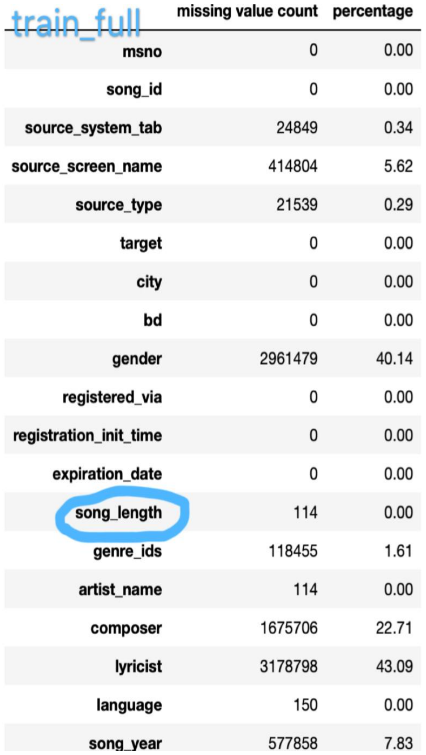
```
test.info()

<class 'pandas.core.frame.DataFrame'
RangeIndex: 2556790 entries, 0 to 25
Data columns (total 6 columns):
id                    int64
msno                  object
song_id               object
source_system_tab     object
source_screen_name    object
source_type           object
dtypes: int64(1), object(5)
memory usage: 117.0+ MB
```

Merge datasets (train, members, songs) & (test, members, songs) according to msno (member id) and song_id:

- train_full: 19 variables, 7 million+ observations
- test_full: 19 variables, 2 million+ observations

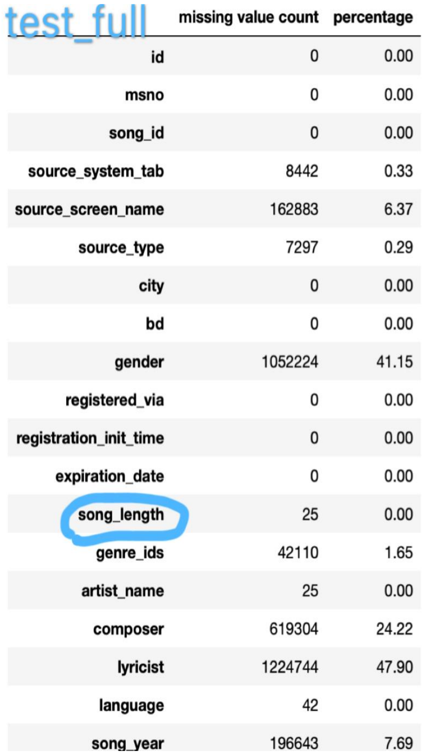
Key Variables:
song_id, song_length, registration_init_time, Expiration_date, artist_name, composer

Data Cleaning-Part 1



train_full

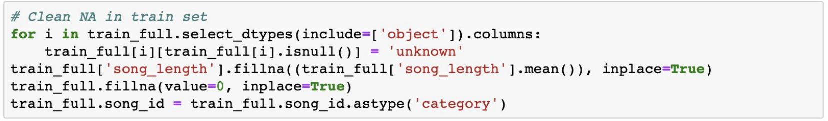
	missing value count	percentage
msno	0	0.00
song_id	0	0.00
source_system_tab	24849	0.34
source_screen_name	414804	5.62
source_type	21539	0.29
target	0	0.00
city	0	0.00
bd	0	0.00
gender	2961479	40.14
registered_via	0	0.00
registration_init_time	0	0.00
expiration_date	0	0.00
song_length	114	0.00
genre_ids	118455	1.61
artist_name	114	0.00
composer	1675706	22.71
lyricist	3178798	43.09
language	150	0.00
song_year	577858	7.83



test_full

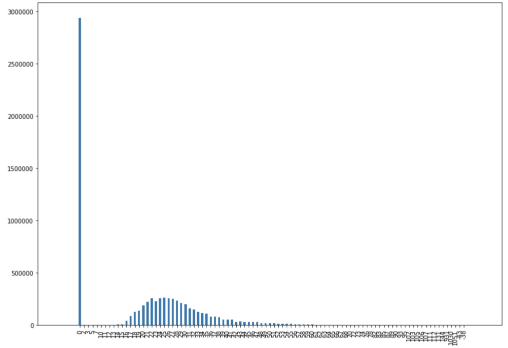
	missing value count	percentage
id	0	0.00
msno	0	0.00
song_id	0	0.00
source_system_tab	8442	0.33
source_screen_name	162883	6.37
source_type	7297	0.29
city	0	0.00
bd	0	0.00
gender	1052224	41.15
registered_via	0	0.00
registration_init_time	0	0.00
expiration_date	0	0.00
song_length	25	0.00
genre_ids	42110	1.65
artist_name	25	0.00
composer	619304	24.22
lyricist	1224744	47.90
language	42	0.00
song_year	196643	7.69

1. **Fill NAs in the datasets:**
   Impute values for missing data in 'song_length'. Except 'song_length', we think other NAs are all unknown information, and thus, we would input 'unknown' for NA.



```
# Clean NA in train set
for i in train_full.select_dtypes(include=['object']).columns:
    train_full[i][train_full[i].isnull()] = 'unknown'
train_full['song_length'].fillna((train_full['song_length'].mean()), inplace=True)
train_full.fillna(value=0, inplace=True)
train_full.song_id = train_full.song_id.astype('category')
```

2. **Check and Visualize Outliers:**



50% of the age is 0, max age is 1051. So we are going to replace those outliers with mean

50% of the age is 0, max age is 1051. So we replace those outliers with mean.

Data Cleaning-Part 2

3. Fix Data Types for 'registration_init_time' and 'expiration_date':

Example Code:

```
# fix date

# registration_init_time

#train

train_full.registration_init_time = pd.to_datetime(train_full.registration_init_time, format='%Y%m%d', errors='ignore')

train_full['registration_init_time_year'] = train_full['registration_init_time'].dt.year

train_full['registration_init_time_month'] = train_full['registration_init_time'].dt.month

train_full['registration_init_time_day'] = train_full['registration_init_time'].dt.day
```

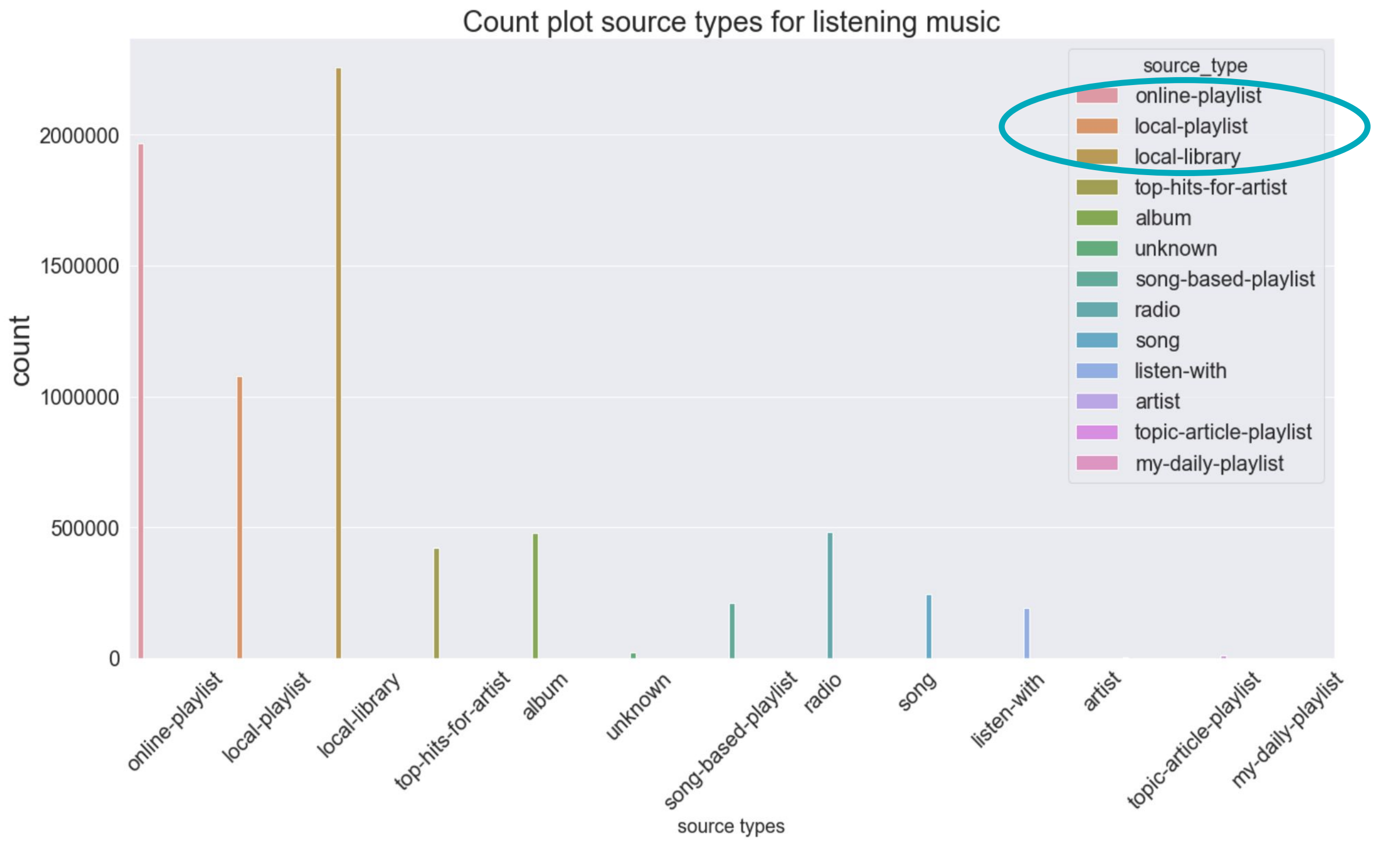
4. Export the cleaned dataset:

```
train_full.to_csv('cleaned_train.csv')
```

Data Inspection:

25 Variables
7 million + Observations

How did users discover songs?

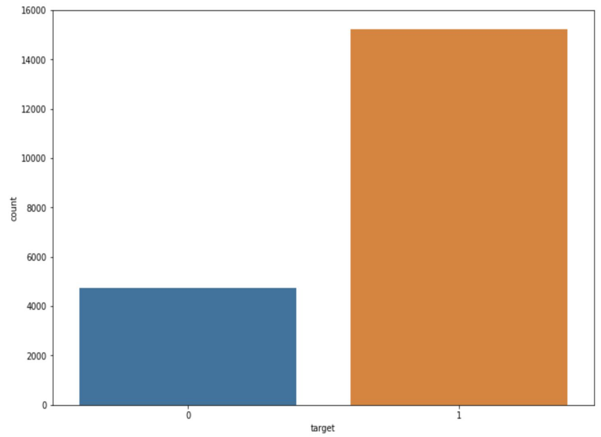


Top 3 Entry Points:

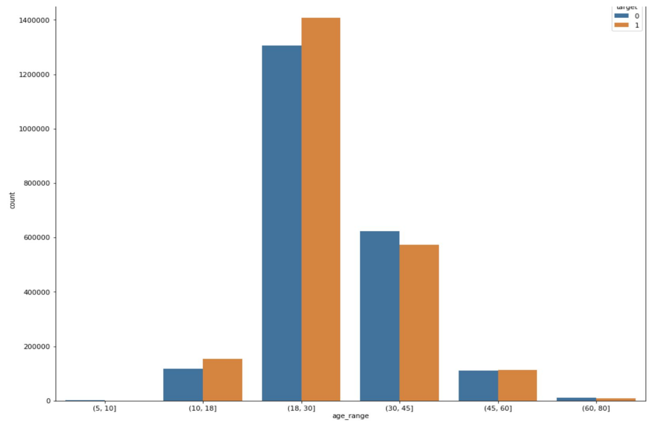
- Online Playlist
- Local Playlist
- Local Library

Exploring the Recurring Listening Events

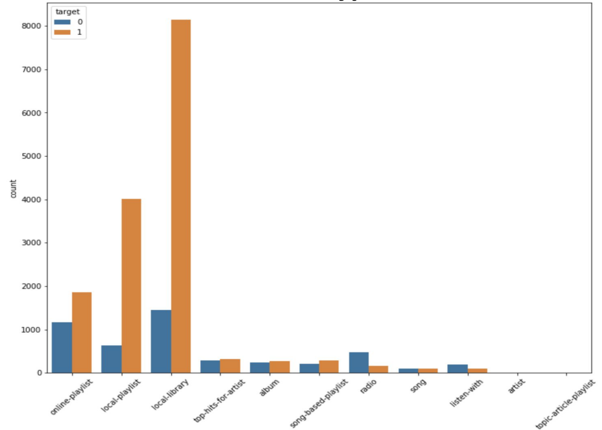
Frequency



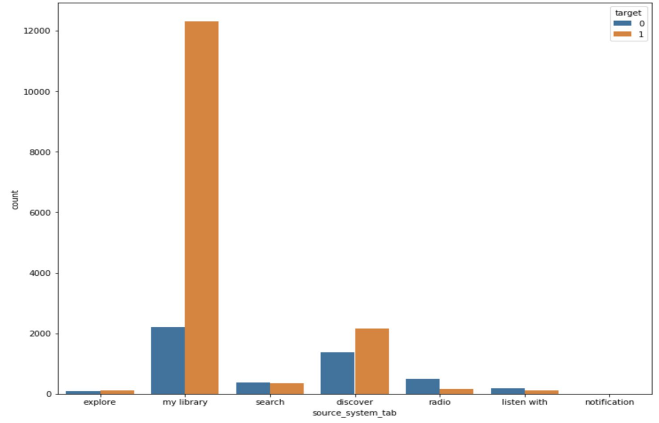
Age



Source Type



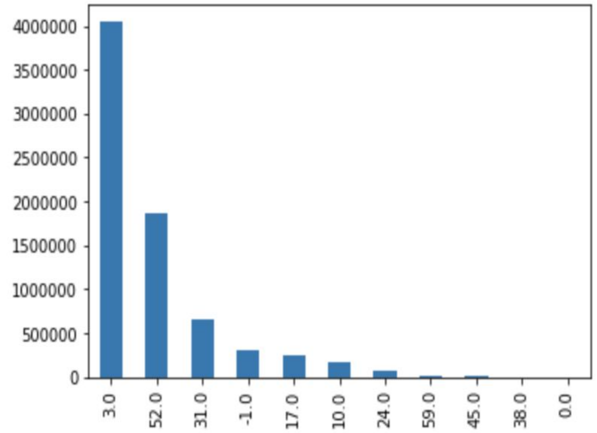
Source System Tab



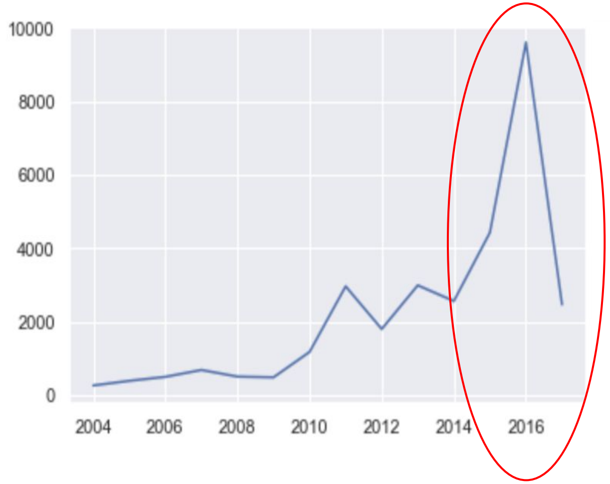
🟧 Repeated

🟦 Not repeated

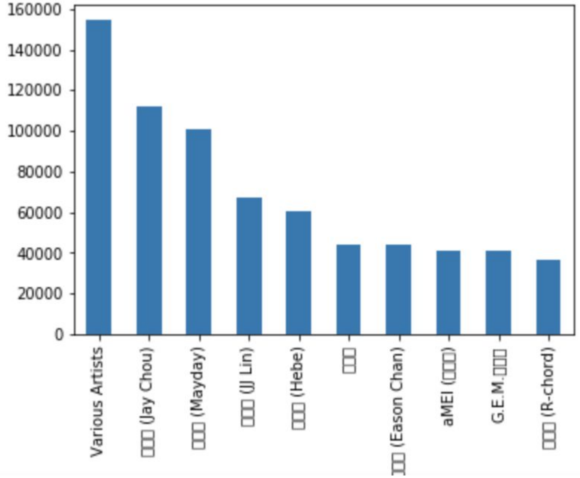
Language

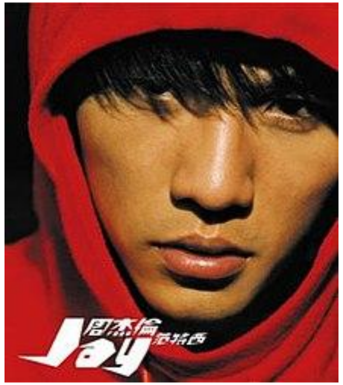


Registration Time



Artist





Feature Engineering

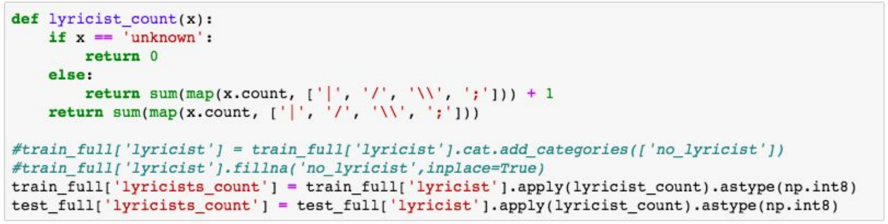
Assumptions:

- Are popular songs more likely to be liked by a new user?
- Are popular singers more likely to be liked by a new user?
- Does a song with more artists attract new users to like it more?

Created new features:

- Count_song_played
- Count_artist_played
- Artist_count
- Lyricist_count
- Composer_count
- ...

Example Code:



```
def lyricist_count(x):
    if x == 'unknown':
        return 0
    else:
        return sum(map(x.count, ['|', '/', '\\', ';'])) + 1
    return sum(map(x.count, ['|', '/', '\\', ';']))

#train_full['lyricist'] = train_full['lyricist'].cat.add_categories(['no_lyricist'])
#train_full['lyricist'].fillna('no_lyricist',inplace=True)
train_full['lyricists_count'] = train_full['lyricist'].apply(lyricist_count).astype(np.int8)
test_full['lyricists_count'] = test_full['lyricist'].apply(lyricist_count).astype(np.int8)
```

First model: Random Forest

Package: **sklearn** (python's machine learning package)

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_tree,y_tree, test_si
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

More data processing:

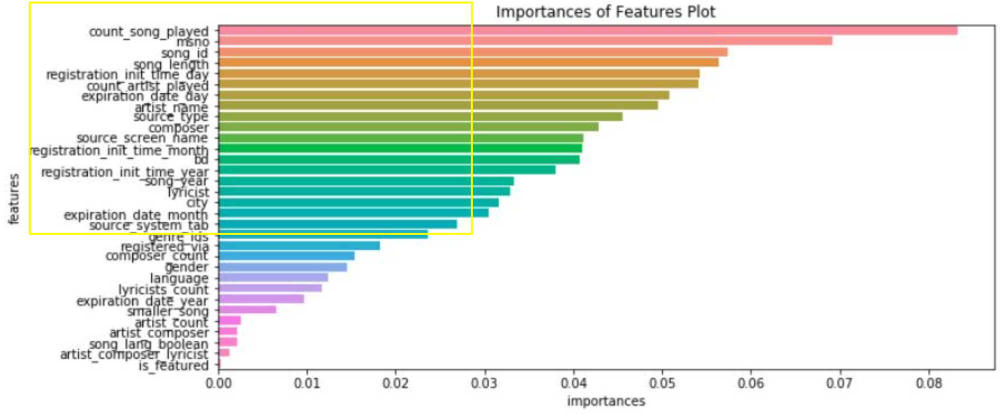
- Cat.code to deal with too many factor levels

```
# Encoding categorical features
for col in X_tree.select_dtypes(include=['category']).columns:
    X_tree[col] = X_tree[col].cat.codes
```

Random Forest model and Variable importance plot

```
rf_model = RandomForestClassifier(n_estimators=100, max_depth = 25)
rf_model.fit(X_train, y_train)
```

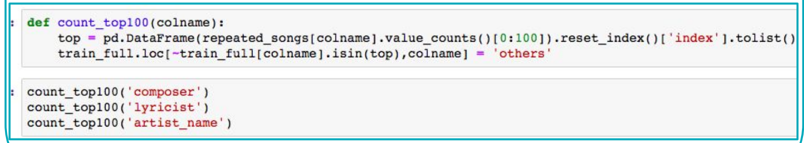
```
   micro avg       0.64      0.64      0.64     70000
   macro avg       0.64      0.64      0.64     70000
weighted avg       0.64      0.64      0.64     70000
```



Models: XGboost, LGBM

Data preparation:

- As we have a large dataset, we define a function to count Top 100.

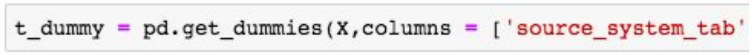


```
def count_top100(colname):
    top = pd.DataFrame(repeated_songs[colname].value_counts()[0:100]).reset_index()['index'].tolist()
    train_full.loc[~train_full[colname].isin(top),colname] = 'others'
```

```
count_top100('composer')
count_top100('lyricist')
count_top100('artist_name')
```

Dummy:

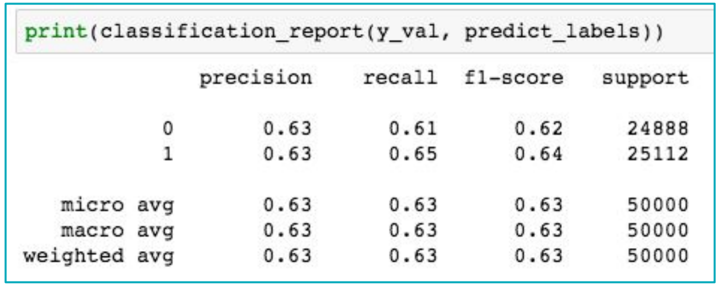
- get_dummies() to dummy categorical variables



```
t_dummy = pd.get_dummies(X,columns = ['source_system_tab'
```

Classification Report

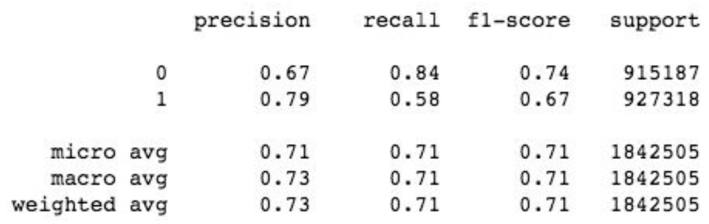
XGboost:



```
print(classification_report(y_val, predict_labels))
```

```
              precision    recall  f1-score   support

           0       0.63      0.61      0.62     24888
           1       0.63      0.65      0.64     25112

   micro avg       0.63      0.63      0.63     50000
   macro avg       0.63      0.63      0.63     50000
weighted avg       0.63      0.63      0.63     50000
```

LGBM:



```
              precision    recall  f1-score   support

           0       0.67      0.84      0.74    915187
           1       0.79      0.58      0.67    927318

   micro avg       0.71      0.71      0.71   1842505
   macro avg       0.73      0.71      0.71   1842505
weighted avg       0.73      0.71      0.71   1842505
```

Conclusion

Decisive factors:

- Count_song_play
- Song_id

Interesting Findings:

- Song_length 🔥
- Registration time
- Explore songs through online playlist
- Language of a song doesn't really matter (Asian consumers are international)

Recommendations:

1. Recommend hit songs to most users
2. But also create variety of playlists to enhance user experience for users with niche tastes
3. Next step KKBox can run further analysis to cluster customers into groups and perform analysis on each group to enhance precision