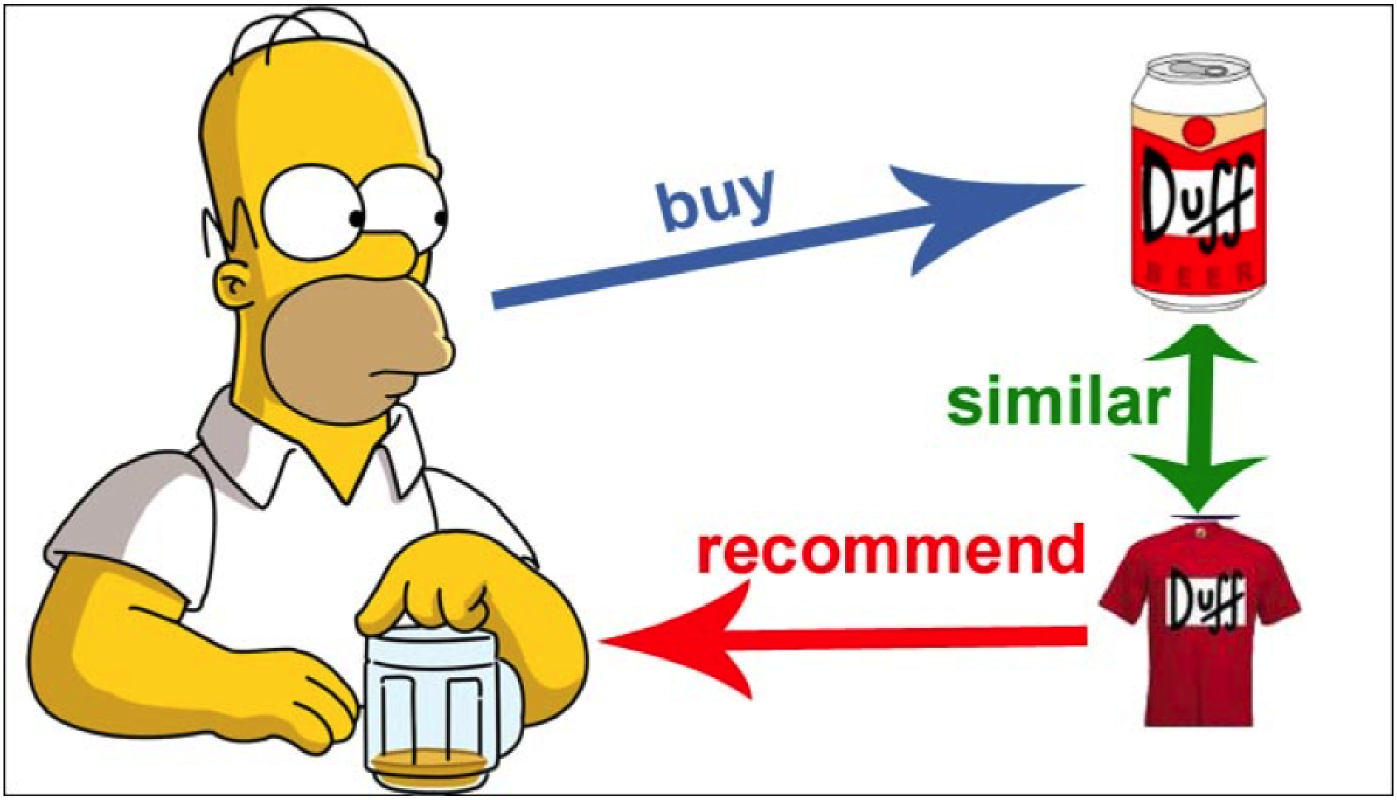
Introduction

CA4015 Assignment 3/4

For our final assignment as part of the CA4015: Advanced Machine Learning module we were given the task of developing a recommender system. We would have to take an existing dataset and develop a recommendation system based off the most modern methods employed today. These methods were provided to us using an existing google colab book and it was up to us to implement and deploy these methods to our own data as we seen right.



Dataset

We would be provided the lastFM dataset. This dataset consists of 5 files namely:

- artists.dat = information about music artists listened and tagged by the users in the data.
- tags.dat = information regarding available tags in the data.
- user_artists.dat = information abouut artists listened to by each user and listening count (weight) for each user/artist pair specified.
- user_taggedartists.dat = files contain the tag assignments of artists provided by each particular user and timestamp for each tag assigned.
- user_friends.dat = the users which are deemed as "friends".

The dataset was mostly clean but required a few minor fixes such as re-indexing among others.

Outline of process

To start we will do some basic analysis of our data: what users listen to the most songs, what artists are most popular and so forth. Once this is done we will merge some of our dataframes together and begin to develop our models. We disregarded the "softmax" model in our process and implemented the regularized matrix model and basic model as per the colab provided. I wanted to compare this model to something else and this is what I did, comparing it to a system which utilises a neural network and works with feature embeddings. Finally, I tested our regularized model on my own spotify account to see what artist recommendations it could make to me based on my favourite artists. We also tried to cluster our data on artist's highest listened score and the number of unique users these artists had.

Links

- Git repo

Recommender System

In [1]:

```
from __future__ import print_function

import numpy as np
import pandas as pd
import collections
from mpl_toolkits.mplot3d import Axes3D
from IPython import display
from matplotlib import pyplot as plt
from IPython.display import display
import seaborn as sns
import sklearn
import sklearn.manifold
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
tf.logging.set_verbosity(tf.logging.ERROR)
```

```
WARNING:tensorflow:From C:\Users\user\anaconda3\lib\site-packages\tensorflow\python\compat
\v2_compat.py:96: disable_resource_variables (from tensorflow.python.ops.variable_scope) i
s deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from collections import Counter, defaultdict
from operator import itemgetter
import tensorflow as tf
from tensorflow import keras
from pylab import rcParams
from pylab import savefig
import lightfm
from lightfm import LightFM
from lightfm.data import Dataset
from lightfm import cross_validation
from lightfm.evaluation import precision_at_k
from lightfm.evaluation import recall_at_k
from lightfm.cross_validation import random_train_test_split
from scipy.sparse import csr_matrix
import scipy
import recmetrics

from sklearn.model_selection import train_test_split
from collections import Counter, defaultdict
from sklearn.metrics import accuracy_score
import matplotlib.ticker as ticker
from math import sqrt
from sklearn.metrics import mean_squared_error
```

```
C:\Users\user\anaconda3\lib\site-packages\lightfm\_lightfm_fast.py:9: UserWarning: LightFM
was compiled without OpenMP support. Only a single thread will be used.
  warnings.warn(
```

In [3]:

```
# Install Altair and activate its colab renderer.
#print("Installing Altair...")
#!pip install git+git://github.com/altair-viz/altair.git
#import altair as alt
#alt.data_transformers.enable('default', max_rows=None)
```

```
#alt.renderers.enable('colab')
#print("Done installing Altair.")

# Install spreadsheets and import authentication module.
#USER_RATINGS = False
#!pip install --upgrade -q gspread
#from google.colab import auth
#import gspread
#from oauth2client.client import GoogleCredentials
```

In [4]:

```
import matplotlib.pyplot as plt
import os
import warnings
from keras.layers import Input, Embedding, Flatten, Dot, Dense, Concatenate
from keras.models import Model
```

Firstly, I converted my data files from .dat to .csv format. I did this via excel using the data tab and it's "get external data" option and extracted it from text. All the delimiting was done by default and I had my files in a delimited format.

To start I read in all of my files and fixed the index for user ID's and artist ID's. This would save us a lot of problems later on when we try to join these dataframes together so doing it initially made the most sense.

In [5]:

```
df = pd.read_csv("data/hetrec2011-lastfm-2k/artists.csv")
df
```

Out[5]:

	id	name	url	pictureURL
0	1	MALICE MIZER	http://www.last.fm/music/MALICE+MIZER	http://userserve-ak.last.fm/serve/252/10808.jpg
1	2	Diary of Dreams	http://www.last.fm/music/Diary+of+Dreams	http://userserve-ak.last.fm/serve/252/3052066.jpg
2	3	Carpathian Forest	http://www.last.fm/music/Carpathian+Forest	http://userserve-ak.last.fm/serve/252/40222717...
3	4	Moi dix Mois	http://www.last.fm/music/Moi+dix+Mois	http://userserve-ak.last.fm/serve/252/54697835...
4	5	Bella Morte	http://www.last.fm/music/Bella+Morte	http://userserve-ak.last.fm/serve/252/14789013...
...	...	...	...	...
17627	18741	Diamanda GalÃ¡s	http://www.last.fm/music/Diamanda+Gal%C3%A1s	http://userserve-ak.last.fm/serve/252/16352971...
17628	18742	Aya RL	http://www.last.fm/music/Aya+RL	http://userserve-ak.last.fm/serve/252/207445.jpg
17629	18743	Coptic Rain	http://www.last.fm/music/Coptic+Rain	http://userserve-ak.last.fm/serve/252/344868.jpg
17630	18744	Oz Alchemist	http://www.last.fm/music/Oz+Alchemist	http://userserve-ak.last.fm/serve/252/29297695...
17631	18745	Grzegorz Tomczak	http://www.last.fm/music/Grzegorz+Tomczak	http://userserve-ak.last.fm/serve/252/59486303...

17632 rows × 4 columns

In [6]:

```
df['id'] = pd.to_numeric(df['id'])
```

```
lst = []
m = np.array(df['id'])
for i in range(0,17632):
    #print(i)
    if i not in df.id.values:
        lst.append(i)

len(lst)
```

Out[6]: 965

Just taking a quick look at the data I could see the last few artist ID's were 18743... and I later found out this would prove problematic. This for loop above demonstrates that in the range of 0 to 17632 (the number of unique artist ID's as per our readME) that there are 965 missing values for this range. This is something we will rectify in all our files.

In [7]:
```
newart = []
for i in range(0, 17632):
    newart.append(i)
```

In [8]:
```
newart = np.array(newart)
df['artID'] = newart.tolist()
```

In [9]:
```
df.drop(columns=['id'], inplace=True)
df.head()
```

Out[9]:

	name	url	pictureURL	artID
0	MALICE MIZER	http://www.last.fm/music/MALICE+MIZER	http://userserve-ak.last.fm/serve/252/10808.jpg	0
1	Diary of Dreams	http://www.last.fm/music/Diary+of+Dreams	http://userserve-ak.last.fm/serve/252/3052066.jpg	1
2	Carpathian Forest	http://www.last.fm/music/Carpathian+Forest	http://userserve-ak.last.fm/serve/252/40222717...	2
3	Moi dix Mois	http://www.last.fm/music/Moi+dix+Mois	http://userserve-ak.last.fm/serve/252/54697835...	3
4	Bella Morte	http://www.last.fm/music/Bella+Morte	http://userserve-ak.last.fm/serve/252/14789013...	4

Our artist ID's are fixed for this file now. There appears to be some unclean names such as "Diamanda GalÃ¡s" among others just taking an initial look at the data here.

In [10]:
```
df1 = pd.read_csv("data/hetrec2011-lastfm-2k/tags.csv")
df1
```

Out[10]:

	tagID	tagValue
0	1	metal
1	2	alternative metal
2	3	goth rock
3	4	black metal
4	5	death metal
...	...	...
11941	12644	suomi

	tagID	tagValue
11942	12645	symbiosis
11943	12646	sverige
11944	12647	eire
11945	12648	electro latino

11946 rows × 2 columns

This file is exclusively dealing with tags. We can disregard changing the index for now.

In [11]:

```
df2 = pd.read_csv("data/hetrec2011-lastfm-2k/user_artists.csv")
df2
```

Out[11]:

	userID	artistID	weight
0	2	51	13883
1	2	52	11690
2	2	53	11351
3	2	54	10300
4	2	55	8983
...	...	...	...
92829	2100	18726	337
92830	2100	18727	297
92831	2100	18728	281
92832	2100	18729	280
92833	2100	18730	263

92834 rows × 3 columns

Due to the fact values are repeated here we can't simply do what we did before to fix the ID values. We also have the presence of userID's which has a similar index problem. We will implement a dictionary to fix these values and map the old values to our new values.

In [12]:

```
df2['artistID'].min()
```

Out[12]:

```
1
```

In [13]:

```
# Since the ids start at 2, we get them to start at 0. We also need to have the max value
df2["userID"] = df2["userID"].apply(lambda x: str(x-2))
df2["artistID"] = df2["artistID"].apply(lambda x: str(x-1))
```

In [14]:

```
df2['userID'] = df2['userID'].astype(int)
xyz =  np.array(df2['userID'])
#zzz = np.array(played['userID'])
vals = []
for i in range(len(xyz)):
    v = xyz[i]
```

```
        if v not in vals:
            vals.append(v)
        else:
            continue
```

In [15]:

```
vals[-1]
```

Out[15]: 2098

In [16]:

```
unique_list = list(set(vals))
unique_list.sort()
unique_list[-1]
```

Out[16]: 2098

In [17]:

```
usenew = []
for i in range(0, 1892):
    usenew.append(i)

usenew[-1]
```

Out[17]: 1891

In [18]:

```
keys = unique_list
values = usenew
dictionary = dict(zip(keys, values))
#print(dictionary) # {'a': 1, 'b': 2, 'c': 3}
```

In [19]:

```
s = df2['userID']

df2['userID'] = s.map(dictionary)
```

In [20]:

```
df2.head()
```

Out[20]:

	userID	artistID	weight
0	0	50	13883
1	0	51	11690
2	0	52	11351
3	0	53	10300
4	0	54	8983

In [21]:

```
df2['artistID'] = df2['artistID'].astype(int)
xyz =  np.array(df2['artistID'])
#zzz = np.array(played['userID'])
vals = []
for i in range(len(xyz)):
    v = xyz[i]
    if v not in vals:
        vals.append(v)
    else:
        continue
```

In [22]:
```
unique_list = list(set(vals))
unique_list.sort()
unique_list[0]
```

Out[22]: 0

In [23]:
```
usenew = []
for i in range(0, 17632):
    usenew.append(i)

usenew[-1]
```

Out[23]: 17631

In [24]:
```
keys = unique_list
values = usenew
diction = dict(zip(keys, values))
#print(diction) # {'a': 1, 'b': 2, 'c': 3}
```

In [25]:
```
s = df2['artistID']

df2['artistID'] = s.map(diction)
```

In [26]:
```
df2['weight'].max()
```

Out[26]: 352698

In [27]:
```
df2
```

Out[27]:

	userID	artistID	weight
0	0	45	13883
1	0	46	11690
2	0	47	11351
3	0	48	10300
4	0	49	8983
...	...	...	...
92829	1891	17615	337
92830	1891	17616	297
92831	1891	17617	281
92832	1891	17618	280
92833	1891	17619	263

92834 rows × 3 columns

Our dataframe "df2" is now adjusted correctly.

In [28]:
```
df3 = pd.read_csv("data/hetrec2011-lastfm-2k/user_friends.csv")
df3
```

Out[28]:

	userID	friendID
0	2	275
1	2	428
2	2	515
3	2	761
4	2	831
...	...	...
25429	2099	1801
25430	2099	2006
25431	2099	2016
25432	2100	586
25433	2100	607

25434 rows × 2 columns

In [29]:
```
# Since the ids start at 2, we get them to start at 0. We also need to have the max value
df3["userID"] = df3["userID"].apply(lambda x: str(x-2))
df3["friendID"] = df3["friendID"].apply(lambda x: str(x-2))

df3['userID'] = pd.to_numeric(df3['userID'])
df3['friendID'] = pd.to_numeric(df3['friendID'])
```

In [30]:
```
df3['friendID'].max()
```

Out[30]:
```
2098
```

In [31]:
```
df3['friendID'].nunique()
```

Out[31]:
```
1892
```

In [32]:
```
xyz =  np.array(df3['userID'])
#zzz = np.array(df2['userID'])
vals = []
for i in range(len(xyz)):
    if xyz[i] not in vals:
        vals.append(xyz[i])
```

In [33]:
```
unique_list = list(set(vals))
unique_list.sort()
unique_list[-1]
```

Out[33]:
```
2098
```

In [34]:
```
usenew = []
for i in range(0, 1892):
```

```python
    usenew.append(i)

usenew[-1]
```

Out[34]: 1891

In [35]:

```python
keys = unique_list
values = usenew
dictionary = dict(zip(keys, values))
#print(dictionary) # {'a': 1, 'b': 2, 'c': 3}
```

In [36]:

```python
s = df3['userID']

df3['userID'] = s.map(dictionary)
```

In [37]:

```python
o = df3['friendID']

df3['friendID'] = o.map(dictionary)
```

In [38]:

```python
df3['friendID'].max()
```

Out[38]: 1891

In [39]:

```python
df3.isnull().values.any()
```

Out[39]: False

In [40]:

```python
df4 = pd.read_csv("data/hetrec2011-lastfm-2k/user_taggedartists-timestamps.csv")
df4
```

Out[40]:

	userID	artistID	tagID	day	month	year
0	2	52	13	1	4	2009
1	2	52	15	1	4	2009
2	2	52	18	1	4	2009
3	2	52	21	1	4	2009
4	2	52	41	1	4	2009
...	...	...	...	...	...	...
186474	2100	16437	4	1	7	2010
186475	2100	16437	292	1	5	2010
186476	2100	16437	2087	1	7	2010
186477	2100	16437	2801	1	5	2010
186478	2100	16437	3335	1	7	2010

186479 rows × 6 columns

In [41]:

```python
df5 = pd.read_csv("data/hetrec2011-lastfm-2k/user_taggedartists.csv")
```

```
df5.head()
```

Out[41]:

	userID	artistID	tagID	day	month	year
0	2	52	13	1	4	2009
1	2	52	15	1	4	2009
2	2	52	18	1	4	2009
3	2	52	21	1	4	2009
4	2	52	41	1	4	2009

Our last two dataframes seem exactly the same. Let's check this before we delete anything.

In [42]:

```
def checkequality(A, B):

    df11 = A.sort_index(axis=1)
    df11 = df11.sort_values(df11.columns.tolist()).reset_index(drop=True)

    df22 = B.sort_index(axis=1)
    df22 = df22.sort_values(df22.columns.tolist()).reset_index(drop=True)
    return (df11 == df22).values.all()

a = checkequality(df4, df5)
print (a)
```

```
True
```

Two of our files are exactly the same. We can delete one of these accordingly.

In [43]:

```
del df5
```

In [44]:

```
# Since the ids start at 2, we get them to start at 0. We also need to have the max value
df4["userID"] = df4["userID"].apply(lambda x: str(x-2))
df4["artistID"] = df4["artistID"].apply(lambda x: str(x-1))

df4['artistID'] = df4['artistID'].astype(int)
df4['userID'] = df4['userID'].astype(int)
```

In [45]:

```
xyz =  np.array(df4['artistID'])
#zzz = np.array(played['userID'])
vals = []
for i in range(len(xyz)):
    v = xyz[i]
    if v not in vals:
        vals.append(v)
    else:
        continue
```

In [46]:

```
unique_list = list(set(vals))
unique_list.sort()
print(unique_list[-1])

usenew = []
for i in range(0, 17632):
    usenew.append(i)

usenew[-1]
```

```
18743
17631
```

Out[46]:

In [47]:

```
keys = unique_list
values = usenew
diction = dict(zip(keys, values))
#print(diction) # {'a': 1, 'b': 2, 'c': 3}
```

In [48]:

```
s = df4['artistID']

df4['artistID'] = s.map(diction)
```

In [49]:

```
s = df4['userID']

df4['userID'] = s.map(dictionary)
#print(dictionary)
```

In [50]:

```
df4
```

Out[50]:

	userID	artistID	tagID	day	month	year
0	0	49	13	1	4	2009
1	0	49	15	1	4	2009
2	0	49	18	1	4	2009
3	0	49	21	1	4	2009
4	0	49	41	1	4	2009
...	...	...	...	...	...	...
186474	1891	11288	4	1	7	2010
186475	1891	11288	292	1	5	2010
186476	1891	11288	2087	1	7	2010
186477	1891	11288	2801	1	5	2010
186478	1891	11288	3335	1	7	2010

186479 rows × 6 columns

In [51]:

```
df4.drop(columns=['day', 'month', 'year'], inplace=True)
```

In [52]:

```
tags = pd.merge(df1, df4, how="inner", left_on="tagID", right_on="tagID")
tags.isnull().values.any()
tags['artistID'].max()
```

Out[52]:

```
12522
```

All of our dataframes have the correct index for artist ID's and user ID's now. This will help us avoid any errors with our recommender model now.

Methodology

Now that all our data files are read in and in the appropriate format we will begin our end to end process. These are as follows:

1. Data cleaning and processing

2. Visualization of trends in the data

3. Fitting our Model

4. Evaluating our Model

Cleaning and Processing

Initial analysis and cleaning

In [53]:
```
df['name'].value_counts()
```

Out[53]:
```
MALICE MIZER                      1
BEAT!BEAT!BEAT!                   1
ãƒˆã‚¯ãƒžãƒ«ã‚·ãƒ¥ãƒ¼ã‚´                 1
Thao with The Get Down Stay Down   1
ãƒªã‚¢ãƒ»ãƒ‡ã‚£ã‚¾ãƒ³                 1
                                  ..
Innerpartysystem                  1
Helia                             1
Devil Sold His Soul               1
Nevea Tears                       1
Grzegorz Tomczak                  1
Name: name, Length: 17632, dtype: int64
```

Let's check all our dataframes for null values to start.

In [54]:
```
dfs = [df, df1, df2, df3, df4]
na = []
for i in range(len(dfs)):
    if dfs[i].isnull().values.any() > 0:
        na.append(dfs[i])
```

In [55]:
```
na
```

Out[55]:
```
[                    name                                      url  \
 0           MALICE MIZER       http://www.last.fm/music/MALICE+MIZER
 1        Diary of Dreams    http://www.last.fm/music/Diary+of+Dreams
 2      Carpathian Forest  http://www.last.fm/music/Carpathian+Forest
 3           Moi dix Mois       http://www.last.fm/music/Moi+dix+Mois
 4            Bella Morte        http://www.last.fm/music/Bella+Morte
 ...                  ...                                          ...
 17627     Diamanda GalÃ¡s  http://www.last.fm/music/Diamanda+Gal%C3%A1s
 17628              Aya RL              http://www.last.fm/music/Aya+RL
 17629         Coptic Rain         http://www.last.fm/music/Coptic+Rain
 17630        Oz Alchemist        http://www.last.fm/music/Oz+Alchemist
 17631    Grzegorz Tomczak    http://www.last.fm/music/Grzegorz+Tomczak

                                        pictureURL  artID
 0        http://userserve-ak.last.fm/serve/252/10808.jpg      0
 1      http://userserve-ak.last.fm/serve/252/3052066.jpg      1
 2      http://userserve-ak.last.fm/serve/252/40222717...      2
```

```
3      http://userserve-ak.last.fm/serve/252/54697835...      3
4      http://userserve-ak.last.fm/serve/252/14789013...      4
...                                                   ...    ...
17627  http://userserve-ak.last.fm/serve/252/16352971...  17627
17628   http://userserve-ak.last.fm/serve/252/207445.jpg  17628
17629   http://userserve-ak.last.fm/serve/252/344868.jpg  17629
17630  http://userserve-ak.last.fm/serve/252/29297695...  17630
17631  http://userserve-ak.last.fm/serve/252/59486303...  17631

[17632 rows x 4 columns]]
```

The only dataframe with nulls is our artists dataframe. Let's investigate this further to see if there is any important missing values such as ID's etc.

In [56]:

```
df.dtypes
```

Out[56]:

```
name          object
url           object
pictureURL    object
artID          int64
dtype: object
```

In [57]:

```
features_with_na = [features for features in df.columns if df[features].isnull().sum() > (

for feature in features_with_na:
    print(feature, np.round(df[feature].isnull().mean(), 4), '% missing values')
    print(features_with_na)
```

```
pictureURL 0.0252 % missing values
['pictureURL']
```

This is a positive result as there are very few null values in the dataframe and the small amount that exist are in a column of lesser important that we will not need to impute missing values for.

In [58]:

```
played = pd.merge(df, df2, how="inner", left_on="artID", right_on="artistID")
played.rename(columns={"weight": "played"}, inplace=True)
```

We will drop the pictureURL column as there is not much information to be gained and there is some nulls present.

In [59]:

```
played.drop(columns=['pictureURL'], inplace=True)
```

Analysis and Visualization

In [209…

```
mean = played['played'].mean()
print("The mean number of times a user plays a song is: " + str(mean))
```

```
The mean number of times a user plays a song is: 745.2439300256372
```

In [210…

```
median = played['played'].median()
print("The median number of times a user plays a song is: " + str(median))
```

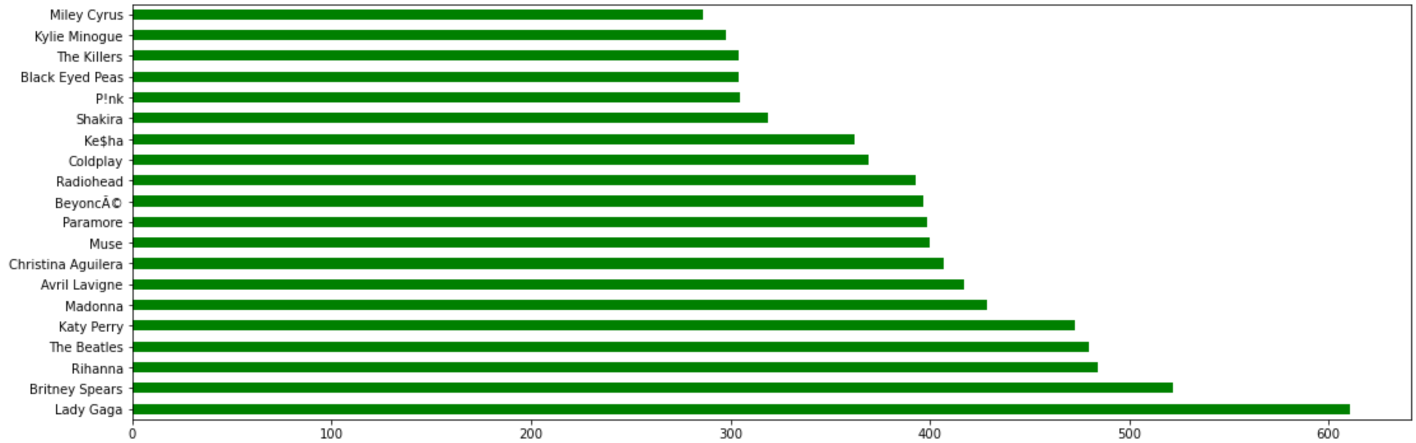
```
The median number of times a user plays a song is: 260.0
```

In [62]:

```
played['name'].value_counts()[:20].plot(kind='barh', color='green', figsize=(18,6))
```

Out[62]:

```
<AxesSubplot:>
```



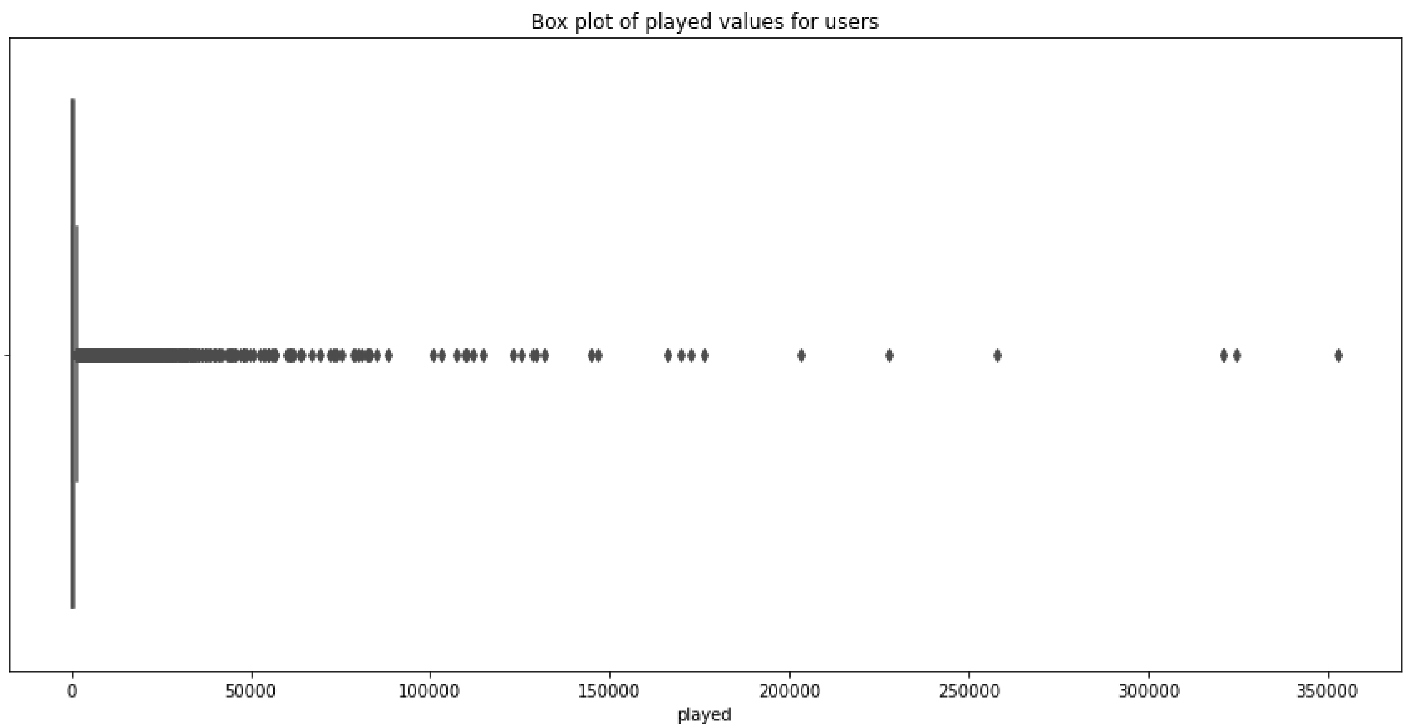
Lady Gaga is by a long distance the most popular artist going by the number of unqiue users listening to her, with approximately 100 more users listening to her in contrast to our second ranked artist.

In [197…

```
plt.figure(figsize=[15,7])
sns.boxplot(x=played['played'], color="gold").set(title='Box plot of played values for use
```

Out[197…

```
[Text(0.5, 1.0, 'Box plot of played values for users')]
```



There appears to be quite a lot of outliers here in the played column. Some users have obviously played their artists songs far more times than the average. We double check our values for mean and median earlier and can confirm there are quite a few outliers here.

In [64]:

```
played.describe()
```

Out[64]:

	artID	userID	artistID	played
count	92834.000000	92834.000000	92834.000000	92834.00000
mean	3235.736724	944.222483	3235.736724	745.24393
std	4197.216910	546.751074	4197.216910	3751.32208
min	0.000000	0.000000	0.000000	1.00000

	artID	userID	artistID	played
25%	430.000000	470.000000	430.000000	107.00000
50%	1237.000000	944.000000	1237.000000	260.00000
75%	4266.000000	1416.000000	4266.000000	614.00000
max	17631.000000	1891.000000	17631.000000	352698.00000

Let's now plot some information regarding our artists.

In [65]:
```
grouped_multiple = played.groupby(['artistID', 'name']).agg({'played': ['mean', 'median',
grouped_multiple.columns = ['mean', 'med', 'max', 'sum']
grouped_multiple = grouped_multiple.reset_index()
#grouped_multiple.sort('price_mean', ascending=False)
grouped_multiple = pd.DataFrame(grouped_multiple)
```

In [66]:
```
artdf = grouped_multiple.sort_values(by=['sum'], ascending=False)
```

In [67]:
```
artdf
```

Out[67]:

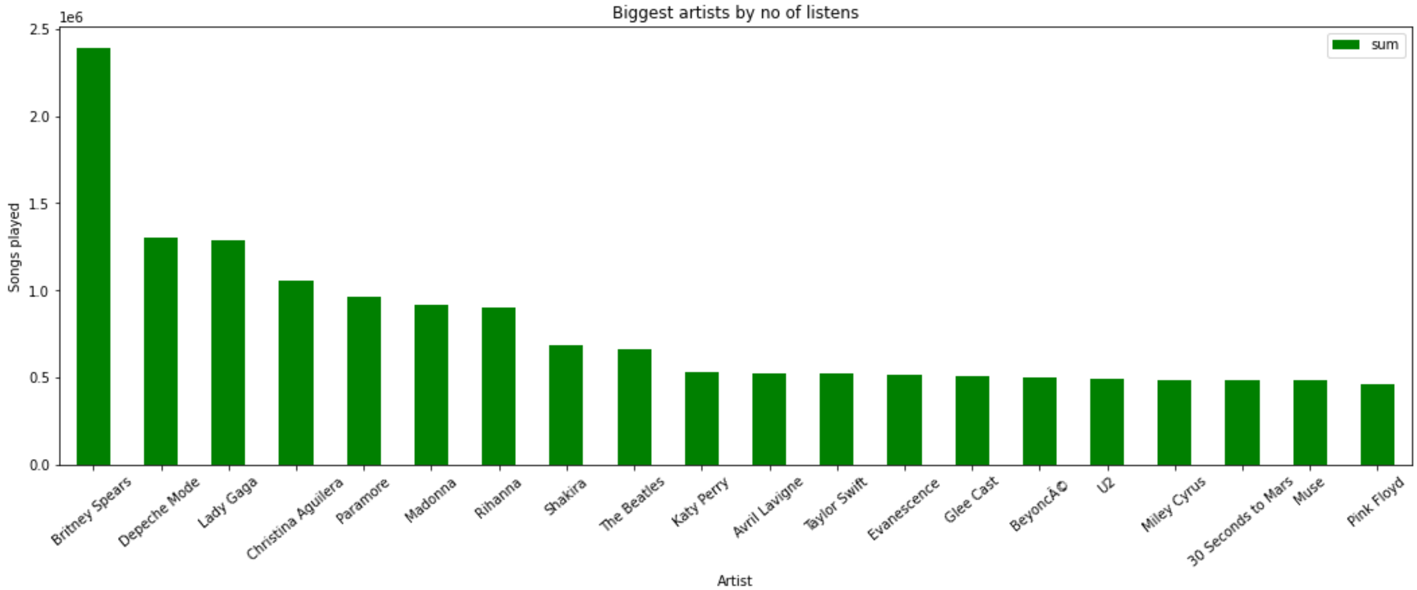
	artistID	name	mean	med	max	sum
283	283	Britney Spears	4584.559387	1000.5	131733	2393140
66	66	Depeche Mode	4614.567376	567.0	352698	1301308
83	83	Lady Gaga	2113.563011	590.0	114672	1291387
286	286	Christina Aguilera	2600.503686	739.0	176133	1058405
492	492	Paramore	2414.659148	417.0	227829	963449
...	...	...	...	...	...	...
16522	16522	K-Precise	1.000000	1.0	1	1
13713	13713	ZÃœNDER	1.000000	1.0	1	1
13712	13712	Evil Masquerade	1.000000	1.0	1	1
16239	16239	Gosling	1.000000	1.0	1	1
16241	16241	Kalson	1.000000	1.0	1	1

17632 rows × 6 columns

In [68]:
```
pt2 = artdf.head(20)
```

In [69]:
```
pt2.plot.bar(x = 'name', y = 'sum', rot = 40, figsize=(18, 6), color='green', xlabel='Arti
```

Out[69]:
```
<AxesSubplot:title={'center':'Biggest artists by no of listens'}, xlabel='Artist', ylabel
='Songs played'>
```



Despite Lady Gaga having the clear higher number of unique users listening to her she is only third in the most played artist by a distance with Britney Spears having the most amount of times her songs were played. This graph appears to suggest that this dataset is heavily leaned towards the most popular artists. From 'Shakira' on there appears to be a consistent base of artists with 500,000 or more plays. We will look at the same plot for users now before we come back to this.

In [201…

```
grouped_multiple = played.groupby(['userID']).agg({'played': ['mean', 'median', 'max', 'su
grouped_multiple.columns = ['mean', 'med', 'max', 'sum']
grouped_multiple = grouped_multiple.reset_index()
#grouped_multiple.sort('price_mean', ascending=False)
grouped_multiple = pd.DataFrame(grouped_multiple)
```

In [202…

```
userdf = grouped_multiple.sort_values(by=['sum'], ascending=False)
```

In [203…

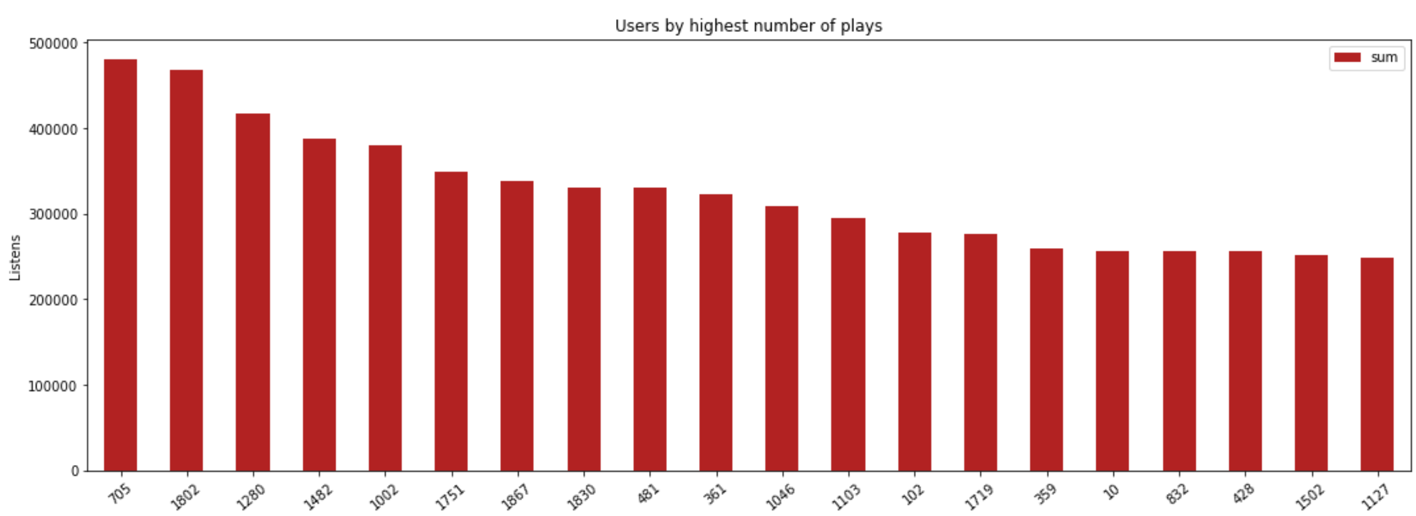
```
pt3 = userdf.head(20)
```

In [205…

```
pt3.plot.bar(x = 'userID', y = 'sum', rot = 40, figsize=(18, 6), color='firebrick', xlabel
```

Out[205…

```
<AxesSubplot:title={'center':'Users by highest number of plays'}, xlabel='User', ylabel='L
istens'>
```



Comparing users to artists there doesn't seem to be an as obvious presence of outliers here. The two users with the highest 'played' values are noticeably ahead of rest but not to the extent as with artists. Due to there being a much smaller cohort of users to artists (1892 to 17632 respectively) it is fair to say that users may have a more even distribution with regards to songs played.

In [74]:

```
artdf['mean']
```

Out[74]:

```
283      4584.559387
66       4614.567376
83       2113.563011
286      2600.503686
492      2414.659148
            ...
16522       1.000000
13713       1.000000
13712       1.000000
16239       1.000000
16241       1.000000
Name: mean, Length: 17632, dtype: float64
```

In [75]:

```
played.shape
```

Out[75]:

```
(92834, 6)
```

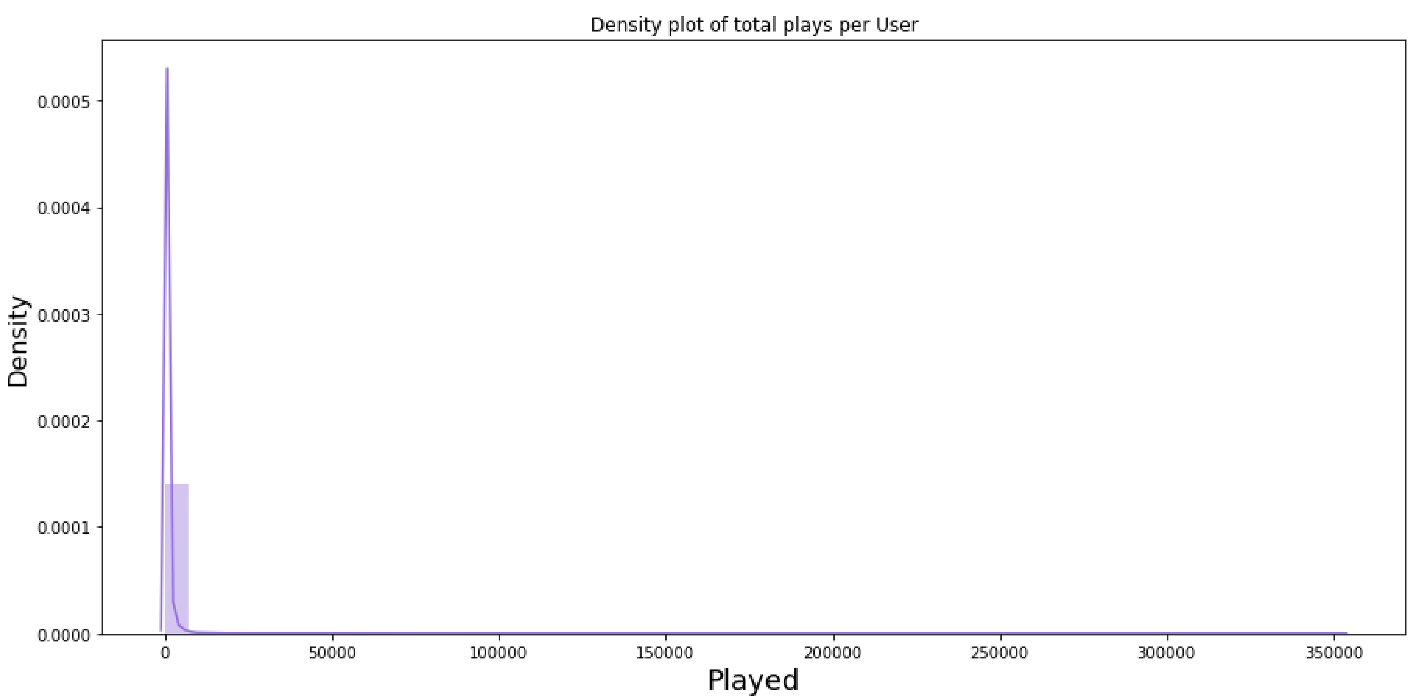
In [195…

```
plt.figure(figsize=[15,7])
sns.distplot(played['played'], color="mediumpurple").set(title='Density plot of total play
plt.xlabel('Played', fontsize=18)
plt.ylabel('Density', fontsize=16)
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) or `his
tplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[195…

```
Text(0, 0.5, 'Density')
```



The majority of values seem around the 1k or less mark. There are a lot of outlier values however going as far as 350,000 for the most extreme values. This confirms our earlier boxplot looking at these values in a clear manner.

In [77]:
```
xyz = pd.DataFrame(played['name'].value_counts())
```
In [78]:
```
xyz = xyz.reset_index()
```
In [79]:
```
xyz.rename(columns={'index': 'name', 'name': 'unique'}, inplace=True)
```
In [80]:
```
xyz
```
Out[80]:

	name	unique
0	Lady Gaga	611
1	Britney Spears	522
2	Rihanna	484
3	The Beatles	480
4	Katy Perry	473
...	...	...
17627	Karmina	1
17628	Alexandre Desplat & Aaron Zigman	1
17629	Burning Brides	1
17630	ozzy	1
17631	Grzegorz Tomczak	1

17632 rows × 2 columns

In [81]:
```
merged_df = artdf.merge(xyz, how = 'inner', on = ['name', 'name'])
```
In [82]:
```
merged_df
```
Out[82]:

	artistID	name	mean	med	max	sum	unique
0	283	Britney Spears	4584.559387	1000.5	131733	2393140	522
1	66	Depeche Mode	4614.567376	567.0	352698	1301308	282
2	83	Lady Gaga	2113.563011	590.0	114672	1291387	611
3	286	Christina Aguilera	2600.503686	739.0	176133	1058405	407
4	492	Paramore	2414.659148	417.0	227829	963449	399
...	...	...	...	...	...	...	...
17627	16522	K-Precise	1.000000	1.0	1	1	1
17628	13713	ZÃœNDER	1.000000	1.0	1	1	1
17629	13712	Evil Masquerade	1.000000	1.0	1	1	1
17630	16239	Gosling	1.000000	1.0	1	1	1
17631	16241	Kalson	1.000000	1.0	1	1	1

17632 rows × 7 columns

In [83]:

```
percent = []
val = merged_df['unique']
total = played['userID'].nunique()
percent = []
for i in range(len(val)):
    y = val[i] / total
    percent.append(y)
    #print(y)
```

In [84]:

```
percent = np.array(percent)
artdf['Percentage'] = percent.tolist()
```

In [85]:

```
artdf
```

Out[85]:

	artistID	name	mean	med	max	sum	Percentage
283	283	Britney Spears	4584.559387	1000.5	131733	2393140	0.275899
66	66	Depeche Mode	4614.567376	567.0	352698	1301308	0.149049
83	83	Lady Gaga	2113.563011	590.0	114672	1291387	0.322939
286	286	Christina Aguilera	2600.503686	739.0	176133	1058405	0.215116
492	492	Paramore	2414.659148	417.0	227829	963449	0.210888
...	...	...	...	...	...	...	...
16522	16522	K-Precise	1.000000	1.0	1	1	0.000529
13713	13713	ZÃœNDER	1.000000	1.0	1	1	0.000529
13712	13712	Evil Masquerade	1.000000	1.0	1	1	0.000529
16239	16239	Gosling	1.000000	1.0	1	1	0.000529
16241	16241	Kalson	1.000000	1.0	1	1	0.000529

17632 rows × 7 columns

In [86]:

```
artdf['unique'] = merged_df['unique'].values
```

In [87]:

```
artdf
```

Out[87]:

	artistID	name	mean	med	max	sum	Percentage	unique
283	283	Britney Spears	4584.559387	1000.5	131733	2393140	0.275899	522
66	66	Depeche Mode	4614.567376	567.0	352698	1301308	0.149049	282
83	83	Lady Gaga	2113.563011	590.0	114672	1291387	0.322939	611
286	286	Christina Aguilera	2600.503686	739.0	176133	1058405	0.215116	407
492	492	Paramore	2414.659148	417.0	227829	963449	0.210888	399
...	...	...	...	...	...	...	...	...
16522	16522	K-Precise	1.000000	1.0	1	1	0.000529	1

	artistID	name	mean	med	max	sum	Percentage	unique
13713	13713	ZÃœNDER	1.000000	1.0	1	1	0.000529	1
13712	13712	Evil Masquerade	1.000000	1.0	1	1	0.000529	1
16239	16239	Gosling	1.000000	1.0	1	1	0.000529	1
16241	16241	Kalson	1.000000	1.0	1	1	0.000529	1

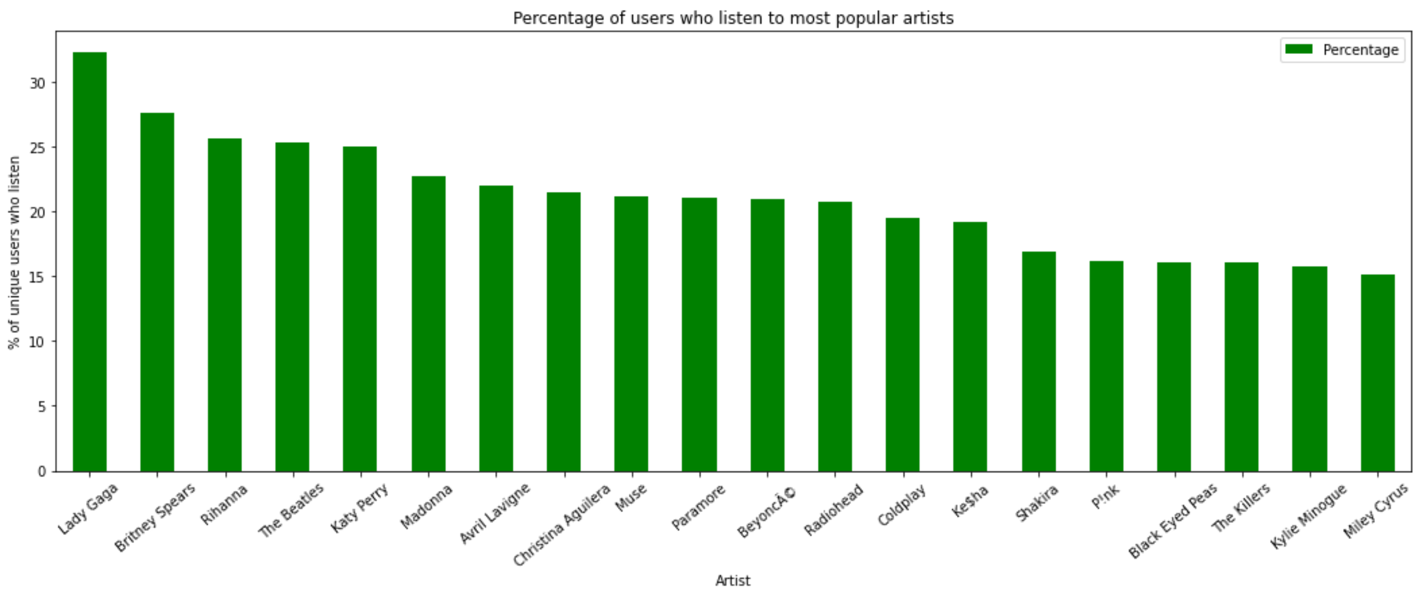
17632 rows × 8 columns

In [88]:
```
artdf['Percentage'] = artdf['Percentage'].multiply(100)
```

In [89]:
```
artdf = artdf.sort_values(by=['Percentage'], ascending=False)
```

In [90]:
```
pt2 = artdf.head(20)
pt2.plot.bar(x = 'name', y = 'Percentage', rot = 40, figsize=(18, 6), color='green', xlabe
```

Out[90]:
```
<AxesSubplot:title={'center':'Percentage of users who listen to most popular artists'}, xl
abel='Artist', ylabel='% of unique users who listen'>
```



This further seems to confirm our data is more geared towards the top. With such high percentages in relative terms of unique users listening to these artists it may cause issues such as the "cold-start" problem for our recommender. By this I mean with so many popular artists with such a high percent of users (and what appears to be fairly similar artists/genres) the recommender may struggle to recommend new or unknown artists to users. This is certainly the problem we seek to avoid. Let's check this information further below checking how many artists have between 1% and 5% of the total users listening to them.

In [91]:
```
values = [1, 2, 3, 4, 5]
for i in range(len(values)):
    x = len(artdf[artdf['Percentage'] <= values[i]])
    print("The percentage of artists with " + str(values[i]) + "% or less users listening
```

```
The percentage of artists with 1% or less users listening to them is 16794.
The percentage of artists with 2% or less users listening to them is 17200.
The percentage of artists with 3% or less users listening to them is 17350.
The percentage of artists with 4% or less users listening to them is 17430.
The percentage of artists with 5% or less users listening to them is 17497.
```

We can tell on the whole of the 17632 artists that there are actually very few who are listened to by a wide audience. There are less than 1000 artists who have more than 1% of users listening to them. This confirms our data is probably leaned very heavily towards the most popular artists such as Britney Spears or Lady Gaga as per our barchart above. Our below density plot confirms this.

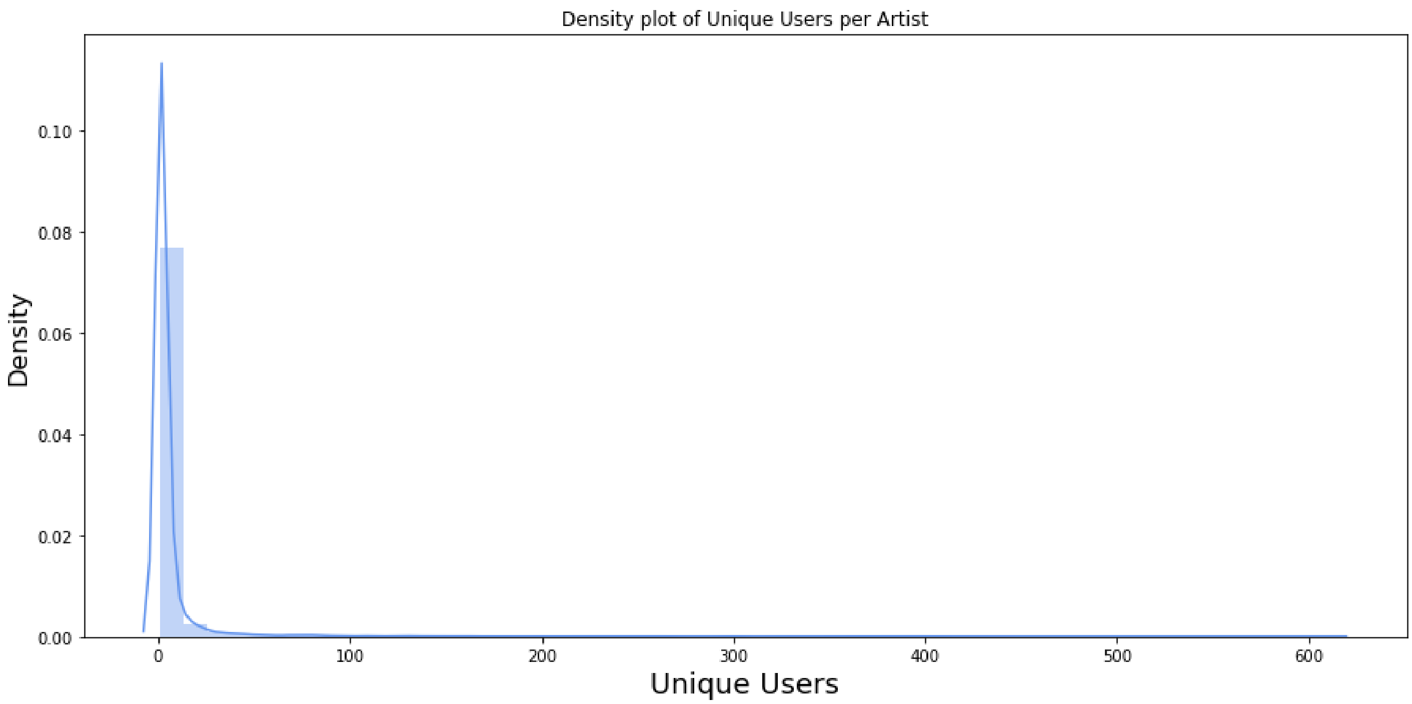
In [200…

```
plt.figure(figsize=[15,7])
sns.distplot(artdf['unique'], color="cornflowerblue").set(title='Density plot of Unique Us
plt.xlabel('Unique Users', fontsize=18)
plt.ylabel('Density', fontsize=16)
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt yo
ur code to use either `displot` (a figure-level function with similar flexibility) or `his
tplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[200…

```
Text(0, 0.5, 'Density')
```

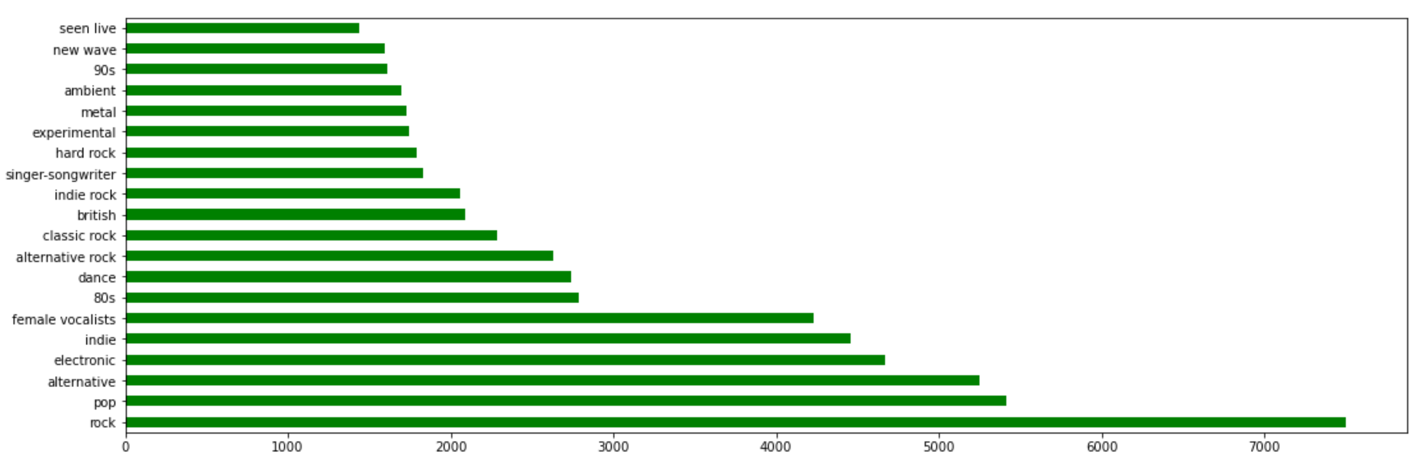


In [95]:

```
# Let's check most popular tags
tags['tagValue'].value_counts()[:20].plot(kind='barh', color='green', figsize=(18,6))
```

Out[95]:

```
<AxesSubplot:>
```



Sparse Representation of Played matrix

In [96]:
```
# Calculate sparsity of matrix
def calculate_sparsity(M):
    matrix_size = float(M.shape[0] * M.shape[1]) # Number of total possible interactions b
    num_plays = len(M.nonzero()[1]) # Number of times any artist has been interacted with
    sparsity = 100 * (1 - float(num_plays / matrix_size))
    return sparsity
```

Normalising our played column

Next, one of the crucial aspects of our recommender system would be dealing with our played column. With such a diverse number of values from the range of 0 to over 350,000 we would have to deal with these appropriately. Our system would not be able to handle values of such a high nature and when I tried to run this I would get very high train errors and "nan" values for test error. I looked at a variety of different ways to normalize this value as a result and I would only incorporate two of these into my dataframe. I looked at capping any values above the 2,000 mark in our played column at 2,000 but errors persisted with this approach. As a result the methods I looked at were:

1) Simple Normalization - normalizing all values based off the highest value in the "played" column.

2) User based Normalization - grouping our played column by users and normalizing each user based off their own max value. I implemented this as a column called "playedUserNorm". This had the best results and was the column I implemented below.

3) Play Count Scaled - here I would take each value in the column and take it away from the minimum value in the column. I would then divide this by the max value of the column minus the minimum value. I implemented this with the column "playCountScaled".

4) Robust Scaling method - here we would scale each feature of the data set by subtracting the median and then dividing by the interquartile range. I tried this method but the results were poor and implementing it took a long time to run.

In [97]:
```
sm = played['played'].groupby(played['userID']).max()
artss = np.array(played['userID'])
playzz = np.array(played['played'])
#artss[-1]
newnorm = []
for i in range(len(playzz)):
    index = artss[i]
    val = playzz[i] / sm[index]
    newnorm.append(val)
```

In [98]:
```
newnorm = np.array(newnorm)

#add newnorm array as new column in DataFrame
played['playedUserNorm'] = newnorm.tolist()
```

In [99]:
```
played['playedUserNorm'].max()
```

Out[99]:
```
1.0
```

In [100…
```
pc = played.played
play_count_scaled = (pc - pc.min()) / (pc.max() - pc.min())

played = played.assign(playCountScaled=play_count_scaled)
```

In [101…

```
# !!! here is our 1) simple normalisation

# played["playBasicNorm"] = played["played"] / played["played"].max()
```

In [102…

```
# played['playCountScaled'].equals(played['playBasicNorm'])
```

In [103…

```
played.head()
```

Out[103…

	name	url	artID	userID	artistID	played	playedUserNorm	playCountScale
0	MALICE MIZER	http://www.last.fm/music/MALICE+MIZER	0	31	0	212	0.055775	0.0005
1	MALICE MIZER	http://www.last.fm/music/MALICE+MIZER	0	256	0	483	0.065394	0.0013
2	MALICE MIZER	http://www.last.fm/music/MALICE+MIZER	0	729	0	76	0.025149	0.0002
3	Diary of Dreams	http://www.last.fm/music/Diary+of+Dreams	1	130	1	1021	0.150902	0.0028
4	Diary of Dreams	http://www.last.fm/music/Diary+of+Dreams	1	240	1	152	0.154315	0.0004

In [104…

```
# !!! here is our attempt at robust scaling as per 4)

#newcol = []
#pl = np.array(played['played'])
#for i in range(len(pl)):
#    val = (pl[i] - played['played'].median())  / (played['played'].quantile(0.75) - playe
#    newcol.append(val)
```

In [105…

```
#newcol = np.array(newcol)

#add newnorm array as new column in DataFrame
#played['playedRobust'] = newcol.tolist()
```

We will now begin to build the model. The first step is building a sparse matrix as input for our models. A sparse matrix is a dataset in which most of the entries are zero, one such example would be a large diagonal matrix. In our case this would involve our dataset of userID, artistID and played columns. We will do this by using the SparseTensor function as part of the tensorflow library.

In [106…

```
def build_rating_sparse_tensor(ratings_df):
  # ========================= Complete this section ============================
    indices = ratings_df[['userID', 'artID']].values
    values = ratings_df['playedUserNorm'].values
  # ============================================================================

    return tf.SparseTensor(
      indices=indices,
      values=values,
      dense_shape=[len(played.userID.unique()), len(played.artID.unique())])
```

In [107…

```
len(played.userID.unique())
```

Out[107… 1892

In [108…

```
def sparse_mean_square_error(sparse_ratings, user_embeddings, artist_embeddings):
    predictions = tf.reduce_sum(
    tf.gather(user_embeddings, sparse_ratings.indices[:, 0]) * tf.gather(artist_embeddings
      axis=1)
    loss = tf.losses.mean_squared_error(sparse_ratings.values, predictions)
    return loss
```

Building the Model

In [109…

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
class CFModel(object):

  def __init__(self, embedding_vars, loss, metrics=None):

    self._embedding_vars = embedding_vars
    self._loss = loss
    self._metrics = metrics
    self._embeddings = {k: None for k in embedding_vars}
    self._session = None

  @property
  def embeddings(self):
    """The embeddings dictionary."""
    return self._embeddings

  def train(self, num_iterations = 100, learning_rate = 1.0, plot_results=True,
            optimizer=tf.train.GradientDescentOptimizer):

    with self._loss.graph.as_default():
      opt = optimizer(learning_rate)
      train_op = opt.minimize(self._loss)
      local_init_op = tf.group(
          tf.variables_initializer(opt.variables()),
          tf.local_variables_initializer())
      if self._session is None:
        self._session = tf.Session()
        with self._session.as_default():
          self._session.run(tf.global_variables_initializer())
          self._session.run(tf.tables_initializer())
          tf.train.start_queue_runners()

    with self._session.as_default():
      local_init_op.run()
      iterations = []
      metrics = self._metrics or ({},)
      metrics_vals = [collections.defaultdict(list) for _ in self._metrics]

      # Train and append results.
      for i in range(num_iterations + 1):
        _, results = self._session.run((train_op, metrics))
        if (i % 10 == 0) or i == num_iterations:
          print("\r iteration %d: " % i + ", ".join(
                ["%s=%f" % (k, v) for r in results for k, v in r.items()]),
                end='')
          iterations.append(i)
```

```
        for metric_val, result in zip(metrics_vals, results):
          for k, v in result.items():
            metric_val[k].append(v)

    for k, v in self._embedding_vars.items():
      self._embeddings[k] = v.eval()

    if plot_results:
      # Plot the metrics.
      num_subplots = len(metrics) + 1
      fig = plt.figure()
      fig.set_size_inches(num_subplots * 10, 8)
      for i, metric_vals in enumerate(metrics_vals):
        ax = fig.add_subplot(1, num_subplots, i + 1)
        for k, v in metric_vals.items():
          ax.plot(iterations, v, label = k)
        ax.set_xlim([1, num_iterations])
        ax.legend()
    return results
```

Build and Run the Model

In [110…

```
from sklearn.model_selection import train_test_split
def build_model(ratings, embedding_dim=3, init_stddev=1.):

  # Split the ratings DataFrame into train and test.
    #train_ratings, test_ratings = train_test_split(ratings, test_size=0.5)
    train_ratings, test_ratings = split_dataframe(ratings)
  # SparseTensor representation of the train and test datasets.
    A_train = build_rating_sparse_tensor(train_ratings)
    A_test = build_rating_sparse_tensor(test_ratings)
  # Initialize the embeddings using a normal distribution.
    U = tf.Variable(tf.random.normal(
      [A_train.dense_shape[0], embedding_dim], stddev=init_stddev))
    V = tf.Variable(tf.random.normal(
      [A_train.dense_shape[1], embedding_dim], stddev=init_stddev))
    train_loss = sparse_mean_square_error(A_train, U, V)
    test_loss = sparse_mean_square_error(A_test, U, V)
    metrics = {
      'train_error': train_loss,
      'test_error': test_loss
    }
    embeddings = {
      "userID": U,
      "artID": V
    }
    return CFModel(embeddings, train_loss, [metrics])
```

In [111…

```
def split_dataframe(df, holdout_fraction=0.3):

    test = df.sample(frac=holdout_fraction, replace=False)
    train = df[~df.index.isin(test.index)]
    return train, test
```

In [112…

```
# take the relevant columns
xyz = played[['userID', 'artID', 'playedUserNorm']]
```

In [113…

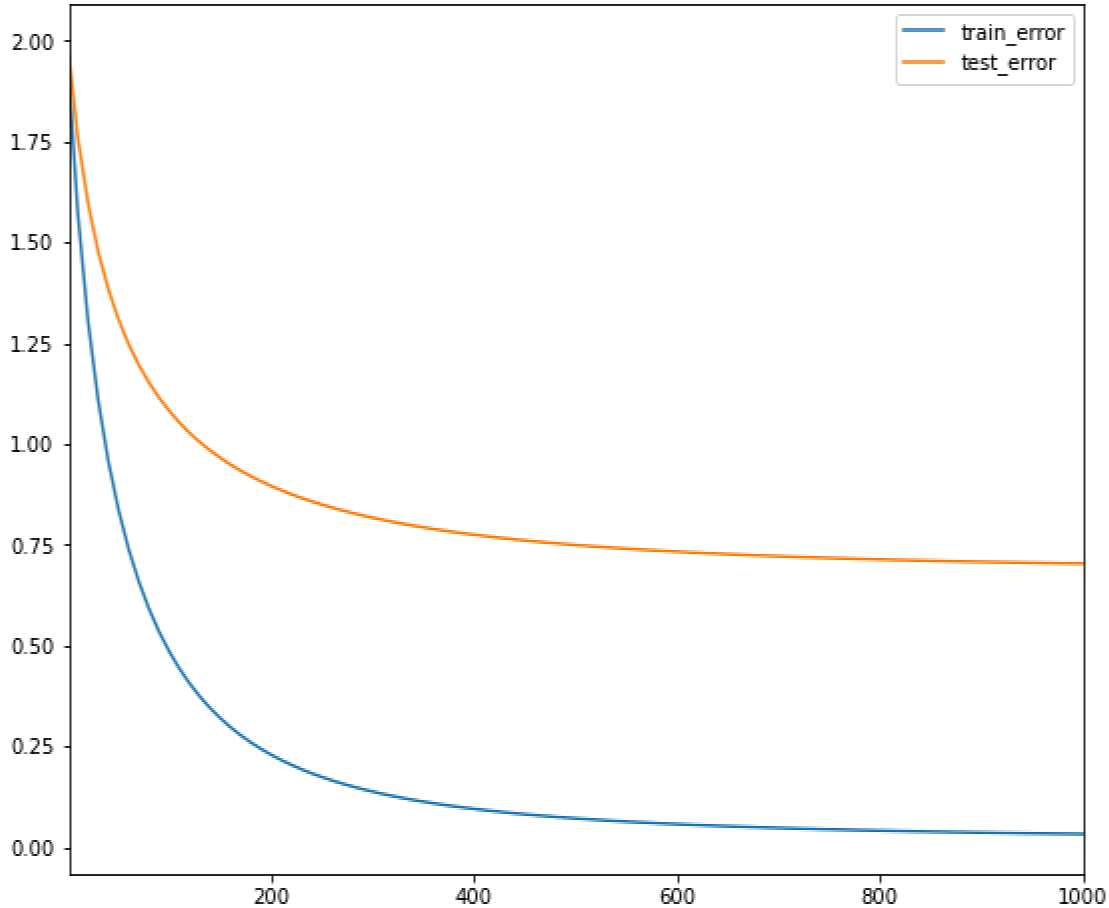
```
model = build_model(xyz,  embedding_dim=30, init_stddev=0.5)
```

```
model.train(num_iterations=1000, learning_rate=10.)
```

```
 iteration 1000: train_error=0.032232, test_error=0.703122
```

Out[113…

```
[{'train_error': 0.03223204, 'test_error': 0.70312166}]
```



I tried to build my model but the high values for listens was giving me errors like so "InvalidArgumentError: indices[4073] = 2077 is not in [0, 1892)".

This was why I changed the artist and user ID's to 1.) start from zero and 2.) increment by 1 until the length of unique values - 1. I had to normalise my weights then as I kept returning nan values for train and test errors above.

Inspect Embeddings

In [114…

```
DOT = 'dot'
COSINE = 'cosine'
def compute_scores(query_embedding, item_embeddings, measure=DOT):
  u = query_embedding
  V = item_embeddings
  if measure == COSINE:
    V = V / np.linalg.norm(V, axis=1, keepdims=True)
    u = u / np.linalg.norm(u)
  scores = u.dot(V.T)
  return scores
```

In [115…

```
from IPython import display
def artist_neighbors(model, title_substring, measure=DOT, k=6):
  ids =  df[df['name'].str.contains(title_substring)].index.values
  titles = df.iloc[ids]['name'].values
  if len(titles) == 0:
    raise ValueError("Found no artist with title %s" % title_substring)
```

```
    print("Nearest neighbors of : %s." % titles[0])
    if len(titles) > 1:
      print("[Found more than one matching artist. Other candidates: {}]".format(
          ", ".join(titles[1:])))
    artistID = ids[0]
    scores = compute_scores(
        model.embeddings["artID"][artistID], model.embeddings["artID"],
        measure)
    score_key = measure + ' score'
    df7 = pd.DataFrame({
        score_key: list(scores),
        'names': df['name'],
    })
    display.display(df7.sort_values([score_key], ascending=False).head(k))
```

Similarity Scores for our model

Let's test our model on some well known artists and see what recommendations it returns. The two similarity measures will be the dot product and cosine similarity. Higher values of both are better. To start with the cosine similarity can be defined as the cosine of the angle between two n-dimensional vectors in an n-dimensional space. The cosine similarity formula corresponds as:

$$similarity(A, B) = {A \cdot B \over |A| \times |B|}$$
Values range between -1 and 1, where -1 is perfectly dissimilar and 1 is perfectly similar. The dot product can be defined in a couple of different ways. It is seen in the cosine similarity formula. The dot product is defined as is equal to the product of the magnitude of the two vectors and the cosecant of the angle between the two vectors. One way it is notated is as:

$$ A \cdot B = |A| |B| \times \cosθ $$
It can also be denoted in this format:

$$ A \cdot B = \sum_{i=1}^n a_i b_i$$
Where: a = 1st vector, b = 2nd vector, n = dimension of the vector space, a_i = component of vector a, b_i = component of vector b

In [116…

```
artist_grp = ['Lady Gaga', 'The Killers', 'Black Eyed Peas', 'Rihanna', 'Gwen Stefani', 'A
for art in range(len(artist_grp)):
    artist_neighbors(model, artist_grp[art], DOT)
    artist_neighbors(model, artist_grp[art], COSINE)
```

```
Nearest neighbors of : Lady Gaga.
[Found more than one matching artist. Other candidates: Lady Gaga VS Christina Aguilera, B
eyoncÃ© e Lady Gaga, Lady Gaga feat BeyoncÃ©]
```

	dot score	names
16102	0.943077	Campbell Brothers
13035	0.832530	Maia Haag-Wackernagel, Alan Mueller & Tristan ...
5760	0.825537	Ã˜
4383	0.820294	Darvin
9676	0.804961	Triumph
16115	0.750073	BOGULTA


```
Nearest neighbors of : Lady Gaga.
[Found more than one matching artist. Other candidates: Lady Gaga VS Christina Aguilera, B
eyoncÃ© e Lady Gaga, Lady Gaga feat BeyoncÃ©]
```

	cosine score	names
83	1.000000	Lady Gaga
294	0.742180	Katy Perry
16115	0.652633	BOGULTA
4420	0.643241	Diwali
286	0.632153	Christina Aguilera
8623	0.596271	Javier BarrÃa

```
Nearest neighbors of : The Killers.
[Found more than one matching artist. Other candidates: Arctic Monkeys vs The Killers]
```

	dot score	names
17286	0.733957	Prem Joshua
16178	0.686957	Ahmet Ã‡alÄ±ÅŸÄ±r
12406	0.663858	Limewax & The Panacea
11130	0.643447	Rainer Maria
11207	0.638483	Airiel
16657	0.602654	Silent Hill

```
Nearest neighbors of : The Killers.
[Found more than one matching artist. Other candidates: Arctic Monkeys vs The Killers]
```

	cosine score	names
223	1.000000	The Killers
16178	0.623625	Ahmet Ã‡alÄ±ÅŸÄ±r
16657	0.622205	Silent Hill
12406	0.621347	Limewax & The Panacea
11072	0.615820	Susan Sarandon
16893	0.605768	Spearmint

```
Nearest neighbors of : Black Eyed Peas.
[Found more than one matching artist. Other candidates: The Black Eyed Peas, Juanes feat.B
lack Eyed Peas]
```

	dot score	names
2610	0.368419	Paul Simon
15637	0.326157	Vordr
12832	0.301931	Shadowtransit
14210	0.299878	Deivos
9521	0.267173	Ð□Ð»Ð¸Ð½Ð° Ð“Ñ€Ð¾Ñ□Ñƒ
11434	0.263688	Carl Davis

```
Nearest neighbors of : Black Eyed Peas.
[Found more than one matching artist. Other candidates: The Black Eyed Peas, Juanes feat.B
lack Eyed Peas]
```

	cosine score	names
300	1.000000	Black Eyed Peas

	cosine score	names
15637	0.705540	Vordr
2610	0.665875	Paul Simon
14210	0.662941	Deivos
9289	0.622263	John Popper
2258	0.620250	Tera Melos

```
Nearest neighbors of : Rihanna.
[Found more than one matching artist. Other candidates: Rihanna (feat. Drake), Jay-Z, Bon
o, The Edge & Rihanna, RihannaÌ€, Sean Paul ft. Rihanna, Rihanna-remixado REnan, \Eminem f
_ Rihanna]
```

	dot score	names
13677	0.782369	Addison Park
13035	0.703576	Maia Haag-Wackernagel, Alan Mueller & Tristan ...
11434	0.692614	Carl Davis
17473	0.680227	Ronnie Day
2258	0.677641	Tera Melos
7864	0.674369	Beyond the Void

```
Nearest neighbors of : Rihanna.
[Found more than one matching artist. Other candidates: Rihanna (feat. Drake), Jay-Z, Bon
o, The Edge & Rihanna, RihannaÌ€, Sean Paul ft. Rihanna, Rihanna-remixado REnan, \Eminem f
_ Rihanna]
```

	cosine score	names
282	1.000000	Rihanna
294	0.712618	Katy Perry
13677	0.626325	Addison Park
1091	0.617115	Chiodos
2258	0.613278	Tera Melos
456	0.605855	Blue

```
Nearest neighbors of : Gwen Stefani.
[Found more than one matching artist. Other candidates: Panic! at the Disco feat. Britney
Spears and Gwen Stefani]
```

	dot score	names
16141	1.126387	Essie Jain
10301	1.059236	Trop Tard
8025	1.041304	Berry Weight
4883	1.039641	Black
6672	1.038205	Vomitory
15995	0.967132	U.S. Bombs

```
Nearest neighbors of : Gwen Stefani.
[Found more than one matching artist. Other candidates: Panic! at the Disco feat. Britney
Spears and Gwen Stefani]
```

	cosine score	names
519	1.000000	Gwen Stefani
11681	0.607283	TÃ³paz
8025	0.595262	Berry Weight
3092	0.595082	Sean Lennon
326	0.591879	Kelly Rowland
4883	0.569846	Black

```
Nearest neighbors of : AC/DC.
```

	dot score	names
2901	0.894110	Ghostface Killah
7942	0.860074	The Campbell Brothers
2432	0.855714	Phil Vassar
14774	0.845297	Prophetic Dream
16168	0.836701	Abu Ali
14479	0.818580	Holy Terror

```
Nearest neighbors of : AC/DC.
```

	cosine score	names
700	1.000000	AC/DC
16168	0.655257	Abu Ali
192	0.633288	System of a Down
6913	0.612368	The "K"
14479	0.601628	Holy Terror
2432	0.589472	Phil Vassar

While our cosine score results provide very strong recommendations using the dot product model here produces mediocre results at best. This also factors in that the dot product is a very popular method for recommender systems and as a result we will need to incorporate further methods to get better results. We will attempt to use a regularized matrix. The key point of this being that regularization is to enforce conditions, for example sparsity or smoothness, that can produce stable predictive functions and in our case improve our model. Overall, I would have to say this model could definitely be improved upon. Let's try an adaptation of our approach.

Regularized Matrix

We are going to incorporate a regularized matrix into our model. A regularized matrix is utilised to enforce conditions, for example sparsity or smoothness, that can produce stable predictive functions.

In [117…

```
def gravity(U, V):
  return 1. / (U.shape[0].value * V.shape[0].value) * tf.reduce_sum(
      tf.matmul(U, U, transpose_a = True) * tf.matmul(V, V, transpose_a = True))

def build_regularized_model(data, embedding_dim = 3, regularization_coeff = .1, gravity_co
  # Split the ratings DataFrame into train and test.
```

```
    train_ratings, test_ratings = split_dataframe(xyz)
    # SparseTensor representation of the train and test datasets.
    A_train = build_rating_sparse_tensor(train_ratings)
    A_test = build_rating_sparse_tensor(test_ratings)
    U = tf.Variable(tf.random_normal(
        [A_train.dense_shape[0], embedding_dim], stddev = init_stddev))
    V = tf.Variable(tf.random_normal(
        [A_train.dense_shape[1], embedding_dim], stddev = init_stddev))

    error_train = sparse_mean_square_error(A_train, U, V)
    error_test = sparse_mean_square_error(A_test, U, V)
    gravity_loss = gravity_coeff * gravity(U, V)
    regularization_loss = regularization_coeff * (
        tf.reduce_sum(U * U) / U.shape[0].value + tf.reduce_sum(V * V) / V.shape[0].value)
    total_loss = error_train + regularization_loss + gravity_loss
    losses = {
        'train_error_observed': error_train,
        'test_error_observed': error_test,
    }
    loss_components = {
        'observed_loss': error_train,
        'regularization_loss': regularization_loss,
        'gravity_loss': gravity_loss,
    }
    embeddings = {"userID": U, "artID": V}

    return CFModel(embeddings, total_loss, [losses, loss_components]), U, V
```

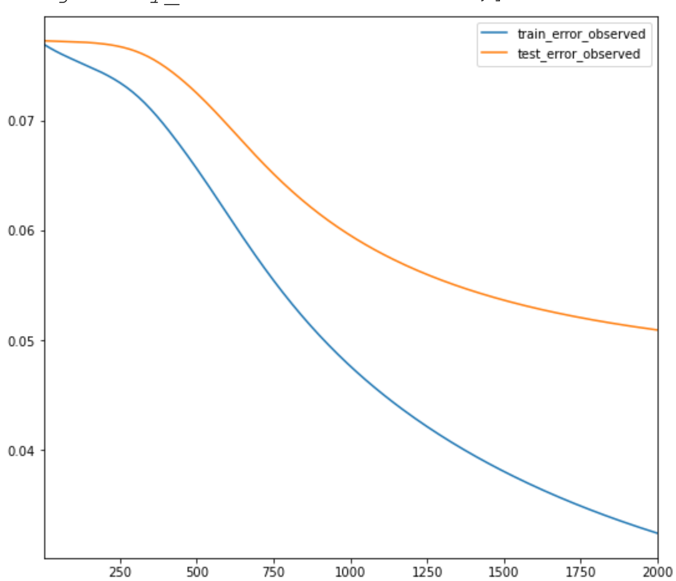
In [118…

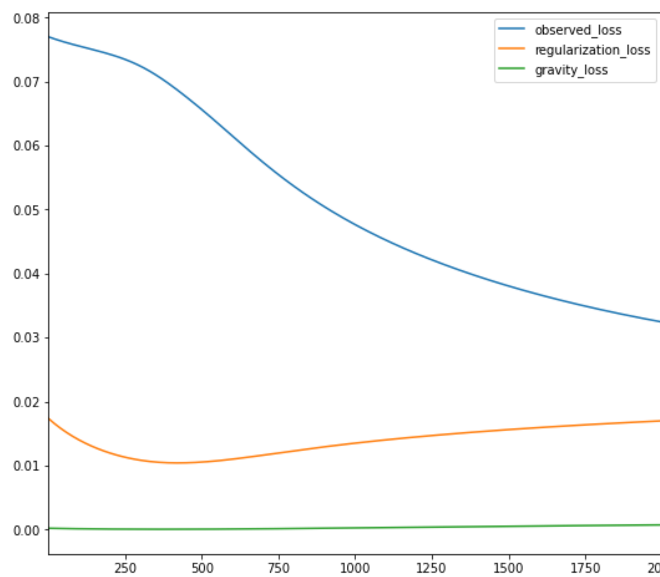
```
reg_model, u, v  = build_regularized_model(xyz, regularization_coeff=0.1, gravity_coeff=1.
reg_model.train(num_iterations=2000, learning_rate=20.)
```

```
iteration 2000: train_error_observed=0.032436, test_error_observed=0.050948, observed_los
s=0.032436, regularization_loss=0.016993, gravity_loss=0.000735
```

Out[118…

```
[{'train_error_observed': 0.032435928, 'test_error_observed': 0.050948296},
 {'observed_loss': 0.032435928,
  'regularization_loss': 0.016992753,
  'gravity_loss': 0.0007352273}]
```





Let's test our model on the same group of artists from before to get a better understanding of it's recommendations. We will see if these are more valid recommendations than before.

In [119…

```
for art in range(len(artist_grp)):
    artist_neighbors(reg_model, artist_grp[art], DOT)
    artist_neighbors(reg_model, artist_grp[art], COSINE)
```

```
Nearest neighbors of : Lady Gaga.
[Found more than one matching artist. Other candidates: Lady Gaga VS Christina Aguilera, B
eyoncÃ© e Lady Gaga, Lady Gaga feat BeyoncÃ©]
```

	dot score	names
83	12.503246	Lady Gaga
283	3.605851	Britney Spears
282	3.276118	Rihanna
673	2.891368	Glee Cast
460	2.481231	Ke$ha
338	2.474091	Taylor Swift

```
Nearest neighbors of : Lady Gaga.
[Found more than one matching artist. Other candidates: Lady Gaga VS Christina Aguilera, B
eyoncÃ© e Lady Gaga, Lady Gaga feat BeyoncÃ©]
```

	cosine score	names
83	1.000000	Lady Gaga
2087	0.718462	å€–ç”°ä¾†æœª
525	0.695442	Taio Cruz
1028	0.690625	Nicki Minaj
1447	0.672937	Far East Movement
458	0.661972	3OH!3

```
Nearest neighbors of : The Killers.
[Found more than one matching artist. Other candidates: Arctic Monkeys vs The Killers]
```

	dot score	names
223	2.222633	The Killers
201	1.791475	Arctic Monkeys
221	1.775246	The Beatles
222	1.355042	Kings of Leon
418	1.355003	The Strokes
148	1.291294	Radiohead

```
Nearest neighbors of : The Killers.
[Found more than one matching artist. Other candidates: Arctic Monkeys vs The Killers]
```

	cosine score	names
223	1.000000	The Killers
2507	0.810717	Mando Diao
2594	0.781756	Razorlight
716	0.779417	ATB
3604	0.739116	Richard Ashcroft
415	0.735183	MÃ³veis Coloniais de Acaju

```
Nearest neighbors of : Black Eyed Peas.
[Found more than one matching artist. Other candidates: The Black Eyed Peas, Juanes feat.B
lack Eyed Peas]
```

	dot score	names
283	1.228968	Britney Spears
83	1.091771	Lady Gaga
282	1.091387	Rihanna
294	0.870181	Katy Perry
673	0.848807	Glee Cast
455	0.783625	Miley Cyrus

```
Nearest neighbors of : Black Eyed Peas.
[Found more than one matching artist. Other candidates: The Black Eyed Peas, Juanes feat.B
lack Eyed Peas]
```

	cosine score	names
300	1.000000	Black Eyed Peas
343	0.863236	The Pussycat Dolls
284	0.853573	Jordin Sparks
322	0.850070	David Guetta
292	0.834922	Lily Allen
1444	0.832557	Selena Gomez

```
Nearest neighbors of : Rihanna.
[Found more than one matching artist. Other candidates: Rihanna (feat. Drake), Jay-Z, Bon
o, The Edge & Rihanna, RihannaÌ€, Sean Paul ft. Rihanna, Rihanna-remixado REnan, \Eminem f
_ Rihanna]
```

	dot score	names
282	5.767070	Rihanna
83	3.276118	Lady Gaga
283	3.061000	Britney Spears
807	2.095273	A Day to Remember
289	1.855651	BeyoncÃ©
294	1.813558	Katy Perry


```
Nearest neighbors of : Rihanna.
[Found more than one matching artist. Other candidates: Rihanna (feat. Drake), Jay-Z, Bon
o, The Edge & Rihanna, RihannaÌ€, Sean Paul ft. Rihanna, Rihanna-remixado REnan, \Eminem f
_ Rihanna]
```


	cosine score	names
282	1.000000	Rihanna
256	0.797839	Mary J. Blige
462	0.725640	Usher
308	0.725307	Ciara
10728	0.697500	Marques Houston
3280	0.691525	Drake

```
Nearest neighbors of : Gwen Stefani.
[Found more than one matching artist. Other candidates: Panic! at the Disco feat. Britney
```

```
Spears and Gwen Stefani]
```

	dot score	names
283	1.148909	Britney Spears
83	0.813766	Lady Gaga
61	0.780915	Madonna
286	0.710928	Christina Aguilera
282	0.680168	Rihanna
673	0.644868	Glee Cast

```
Nearest neighbors of : Gwen Stefani.
[Found more than one matching artist. Other candidates: Panic! at the Disco feat. Britney
Spears and Gwen Stefani]
```

	cosine score	names
519	1.000000	Gwen Stefani
292	0.921257	Lily Allen
314	0.914428	Fergie
478	0.877245	Ne-Yo
2504	0.870947	Robyn
522	0.852184	Pixie Lott

```
Nearest neighbors of : AC/DC.
```

	dot score	names
221	1.594293	The Beatles
157	1.374671	Pink Floyd
700	1.333336	AC/DC
1403	1.240909	Led Zeppelin
167	1.173284	Placebo
45	1.166058	Duran Duran

```
Nearest neighbors of : AC/DC.
```

	cosine score	names
700	1.000000	AC/DC
624	0.827144	Neil Young
726	0.806348	Alice Cooper
6582	0.777992	John Frusciante
1342	0.760324	Tenacious D
4526	0.751988	Cat Stevens

These are actually very good recommendations produced by our recommender system based off each users unique normalised values based on the highest listened value they obtained. Our regularized model is much superior on initial inspection than our standard model with much better recommendations all around. Our test error has also decreased noticeably here. Let's test this further on one of the artists here: AC/DC, who would be popular but as should earlier not in the top 20. Let's try verify our results with the appropriate tag information.

In [226…

```
played[played['name'] == 'AC/DC'].head(1)
```

Out[226…

	name	url	artID	userID	artistID	played	playedUserNorm	playCountScale
33561	AC/DC	http://www.last.fm/music/AC%252FDC	700	16	700	853	0.390032	0.00241

In [225…

```
# id = 700
tags[tags['artistID'] == 700]
```

Out[225…

	tagID	tagValue	userID	artistID
15539	24	pop	40	700
15826	24	pop	128	700
15899	24	pop	143	700
16728	24	pop	439	700
18720	24	pop	1249	700
20586	24	pop	1777	700
27507	39	dance	439	700
27987	39	dance	803	700
30645	49	female vocalist	1249	700
36172	73	rock	439	700
66013	109	pop rock	439	700
66309	109	pop rock	1249	700
67482	127	seen live	533	700
68791	130	female vocalists	40	700
69004	130	female vocalists	143	700
69851	130	female vocalists	439	700
71432	130	female vocalists	1249	700
94543	206	<3	439	700
96151	213	beautiful	439	700
99038	234	amazing	439	700
99713	238	sexy	439	700
99802	238	sexy	803	700
107429	308	latin	44	700
107463	308	latin	143	700
107700	308	latin	1790	700
108777	319	spanish	143	700
110247	349	teen pop	439	700
112174	356	latin pop	141	700
112178	356	latin pop	439	700

	tagID	tagValue	userID	artistID
112267	357	dance pop	439	700
112911	368	mexican	803	700
112914	368	mexican	1777	700
122001	508	love	439	700
142428	1080	mexico	1014	700
145046	1296	superstar	1790	700
148454	1554	romantica	1014	700
149434	1727	glam	533	700
156511	2670	fashion	1790	700
158167	3067	luxo	339	700
158171	3068	depre music	339	700
172125	7071	dulce maria	988	700
172133	7075	belinda	1014	700
184879	12030	puta de mierda ojala y te mueras con un pinche...	1777	700
185051	12090	international	1790	700

Taking AC/DC as an example, the tags appear to be very misleading! It is blatantly obvious these tags are a poor reflection of this artist and potentially many other well known artists where there could be the prospect of more spam comments like so. Lesser known artists may not be subject to these same misleading comments as with smaller listening bases the listeners would be more reliable as they are probably big fans of these not so mainstream artists. It is for the better probably that we did not try use tag information for our recommender. This shows how good our recommendations actually were using this regularized model. It must be said that recommendations such as Led Zeppelin make plenty of sense. We will now move on and see how another recommender system compares to our models based off the google colab provided.

Alternate method - Recommender based on Neural Network

Here we will try to implement another type of recommender system and see does it produce equally as good of results. Our alternate model is based off a neural network to make predictions for users based off listening numbers. As per this article on Investopedia , a neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. This will help us uncover any listening patterns found in our data by users.

In [120…
```
sub = played[['userID', 'artID', 'playedUserNorm']]
```

In [121…
```
sub.head()
```

Out[121…

	userID	artID	playedUserNorm

	userID	artID	playedUserNorm
0	31	0	0.055775
1	256	0	0.065394
2	729	0	0.025149
3	130	1	0.150902
4	240	1	0.154315

In [122…

```
train, test = train_test_split(sub, test_size=0.3, train_size=0.7)
```

In [123…

```
n_users = len(sub.userID.unique())
n_users
```

Out[123…

```
1892
```

In [124…

```
n_artist = len(sub.artID.unique())
n_artist
```

Out[124…

```
17632
```

In [125…

```
# creating artist embedding path
artist_input = Input(shape=[1], name="Artist-Input")
artist_embedding = Embedding(n_artist + 1, 5, name="Artist-Embedding")(artist_input)
artist_vec = Flatten(name="Flatten-Artist")(artist_embedding)

# creating user embedding path
user_input = Input(shape=[1], name="User-Input")
user_embedding = Embedding(n_users + 1, 5, name="User-Embedding")(user_input)
user_vec = Flatten(name="Flatten-Users")(user_embedding)

# performing dot product and creating model
prod = Dot(name="Dot-Product", axes=1)([artist_vec, user_vec])
model = Model([user_input, artist_input], prod)
model.compile('adam', 'mean_squared_error')
```

In [126…

```
from keras.models import load_model

if os.path.exists('regression_model.h5'):
    model = load_model('regression_model.h5')
else:
    history = model.fit([train.userID, train.artID], train.playedUserNorm, epochs=5, verbo
    model.save('regression_model.h5')
    plt.plot(history.history['loss'])
    plt.xlabel("Epochs")
    plt.ylabel("Training Error")
```

In [127…

```
model.evaluate([test.userID, test.artID], test.playedUserNorm)
```

Out[127…

```
11.725609324350035
```

In [128…

```
predictions = model.predict([test.userID.head(10), test.artID.head(10)])
```

```
[print(predictions[i], test.playedUserNorm.iloc[i]) for i in range(0,10)]
```

```
[3.8004408] 0.029461077844311376
[1.3423139] 0.1961297071129707
[0.56713766] 0.04888453738023638
[4.5146008] 0.04554384711555635
[1.4169158] 0.6901547579848535
[0.18357943] 0.014084507042253521
[4.2971926] 0.042230252968508
[6.7329855] 0.05062413314840499
[1.868871] 0.3402061855670103
[5.921677] 0.1063786974310819
```

Out[128…
```
[None, None, None, None, None, None, None, None, None, None]
```

Neural Network

In [129…
```
# creating book embedding path
artist_input = Input(shape=[1], name="Artist-Input")
artist_embedding = Embedding(n_artist + 1, 5, name="Artist-Embedding")(artist_input)
artist_vec = Flatten(name="Flatten-Artists")(artist_embedding)

# creating user embedding path
user_input = Input(shape=[1], name="User-Input")
user_embedding = Embedding(n_users + 1, 5, name="User-Embedding")(user_input)
user_vec = Flatten(name="Flatten-Users")(user_embedding)

# concatenate features
conc = Concatenate()([artist_vec, user_vec])

# add fully-connected-layers
fc1 = Dense(128, activation='relu')(conc)
fc2 = Dense(32, activation='relu')(fc1)
out = Dense(1)(fc2)

# Create model and compile it
model2 = Model([user_input, artist_input], out)
model2.compile('adam', 'mean_squared_error')
```

In [130…
```
from keras.models import load_model

if os.path.exists('regression_model2.h5'):
    model2 = load_model('regression_model2.h5')
else:
    history = model2.fit([train.userID, train.artID], train.playedUserNorm, epochs=5, verb
    model2.save('regression_model2.h5')
    plt.plot(history.history['loss'])
    plt.xlabel("Epochs")
    plt.ylabel("Training Error")
```

In [131…
```
model2.evaluate([test.userID, test.artID], test.playedUserNorm)
```

Out[131…
```
1895890.8385179169
```

In [132…
```
predictions = model2.predict([test.userID.head(10), test.artID.head(10)])

[print(predictions[i], test.playedUserNorm.iloc[i]) for i in range(0,10)]
```

```
[260.3584] 0.029461077844311376
```

```
[240.24399] 0.1961297071129707
[1202.9689] 0.04888453738023638
[395.3229] 0.045543847115556 35
[287.8999] 0.6901547579848535
[232.51555] 0.014084507042253521
[317.00592] 0.042230252968508
[257.6052] 0.05062413314840499
[250.70871] 0.3402061855670103
[1010.34717] 0.1063786974310819
```

Out[132…

```
[None, None, None, None, None, None, None, None, None, None]
```

Visualizing Embeddings

Next, we will visualize our artist embeddings. As per this article here, embeddings can be defined as "vector representations of an entity. Each item in the vector represents a feature or a combination of features for that entity".

In [133…

```python
# Extract embeddings
artist_em = model.get_layer("Artist-Embedding")
artist_em_weights = artist_em.get_weights()[0]
```

In [134…

```python
artist_em_weights[:5]
```

Out[134…

```
array([[ 0.04103883, -0.2710777 ,  0.28193265,  0.01121594, -0.21962109],
       [ 0.42227656, -0.44878516,  0.46648487, -0.12221594, -0.40105212],
       [-0.18079431, -0.13342255,  0.1553544 , -0.3333101 , -0.00778394],
       [ 0.150518  , -0.19249448,  0.1592992 ,  0.12258738, -0.12723902],
       [ 0.2513393 , -0.21272472,  0.1814355 , -0.22018886, -0.13649407]],
      dtype=float32)
```

PCA

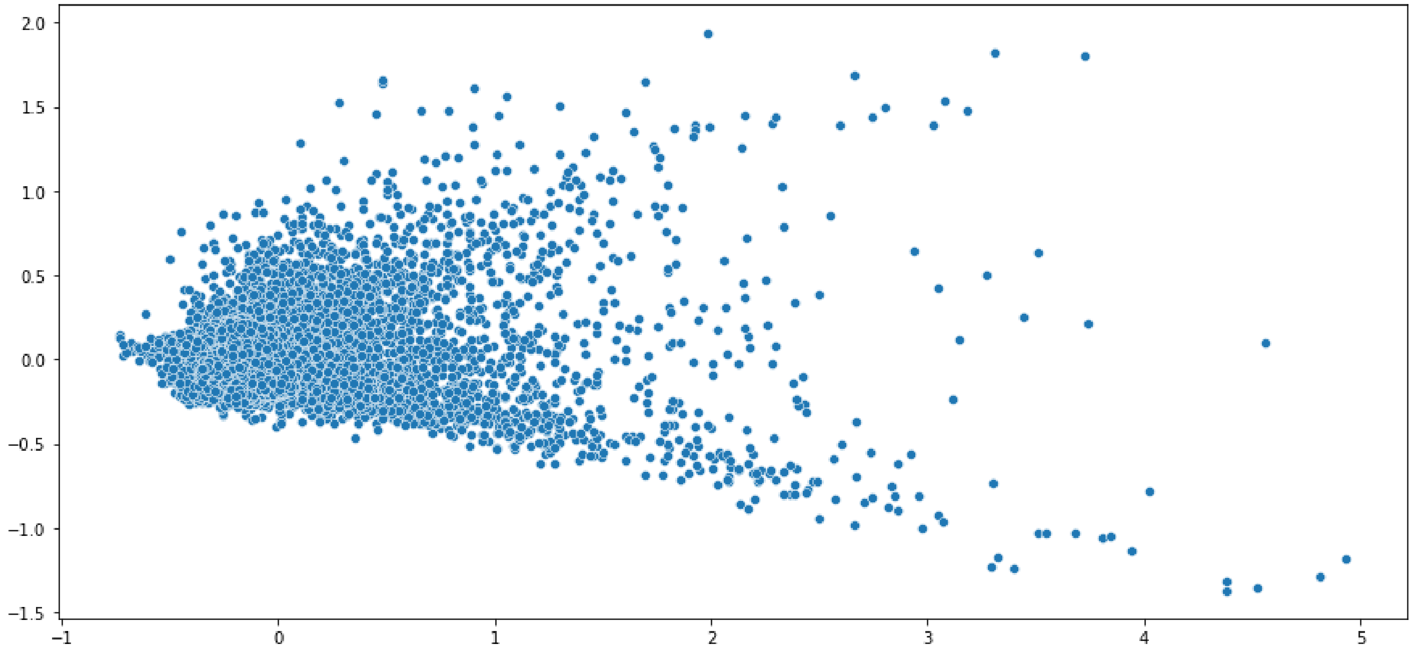
Let's perform principal component analysis (PCA) on our artist embedddings. PCA is defined as the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest. It is commonly used in exploratory data analysis.

In [135…

```python
from sklearn.decomposition import PCA
import seaborn as sns

pca = PCA(n_components=2)
pca_result = pca.fit_transform(artist_em_weights)
plt.figure(figsize=[15,7])
sns.scatterplot(x=pca_result[:,0], y=pca_result[:,1])
```

Out[135…

```
<AxesSubplot:>
```



In [136…

```
artist_em_weights = artist_em_weights / np.linalg.norm(artist_em_weights, axis = 1).reshap
artist_em_weights[0][:10]
np.sum(np.square(artist_em_weights[0]))
```

Out[136…

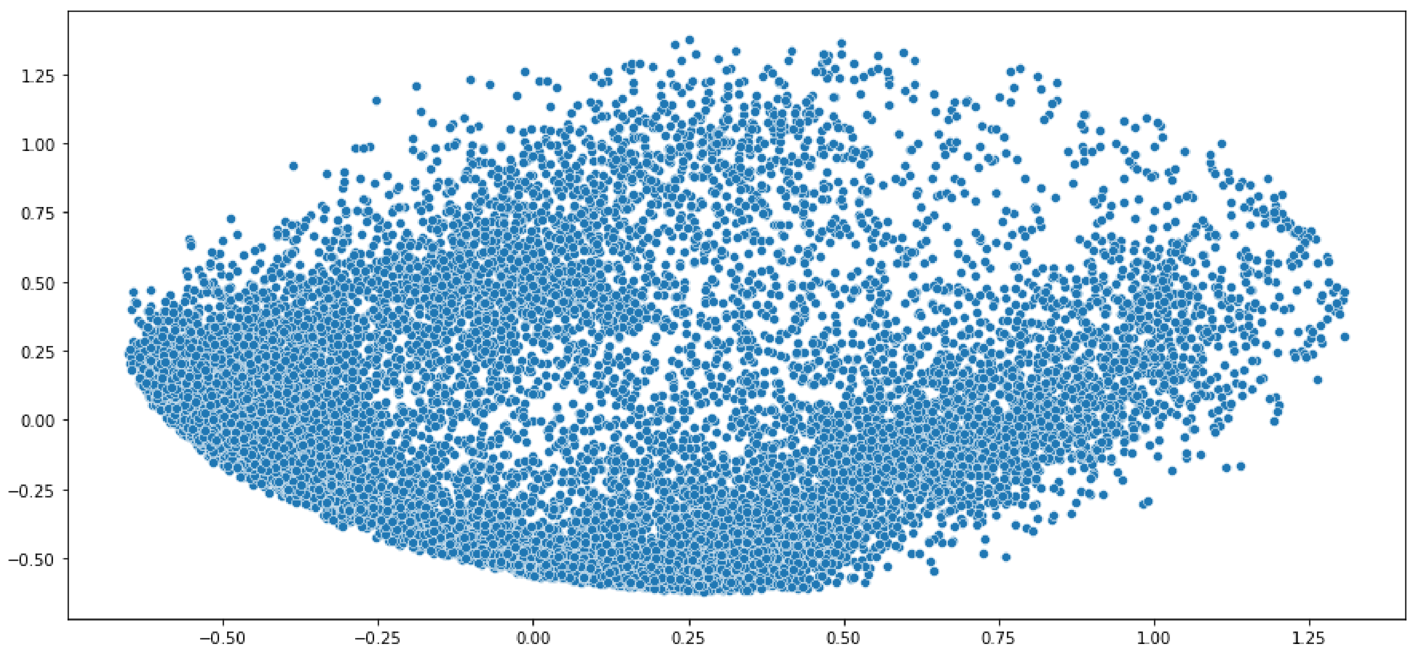
```
1.0
```

In [137…

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(artist_em_weights)
plt.figure(figsize=[15,7])
sns.scatterplot(x=pca_result[:,0], y=pca_result[:,1])
```

Out[137…

```
<AxesSubplot:>
```



TSNE

Let's now look at the t-Distributed Stochastic Neighbor Embeddings (TSNE) for artists. This allows us to see how the artist embeddings are arranged in a high-dimensional space.

In [138…

```
from sklearn.manifold import TSNE
```

```
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tnse_results = tsne.fit_transform(artist_em_weights)
```

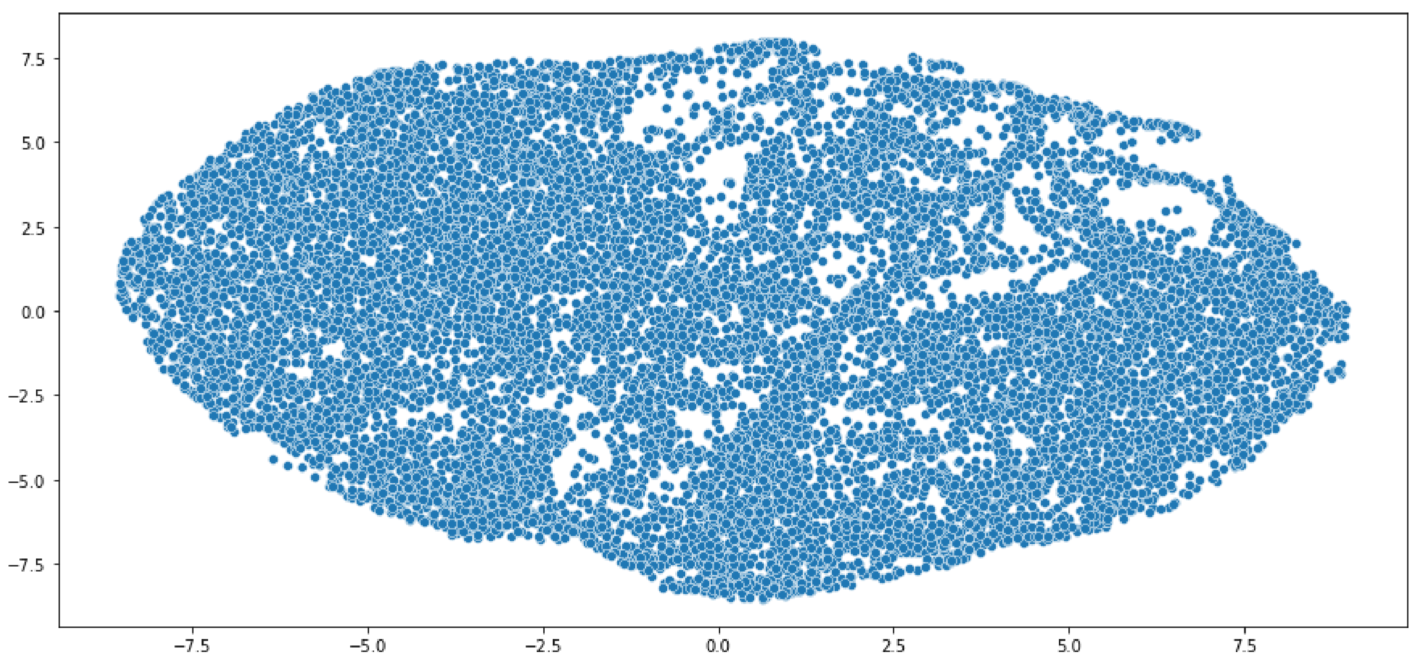
```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 17633 samples in 0.034s...
[t-SNE] Computed neighbors for 17633 samples in 1.308s...
[t-SNE] Computed conditional probabilities for sample 1000 / 17633
[t-SNE] Computed conditional probabilities for sample 2000 / 17633
[t-SNE] Computed conditional probabilities for sample 3000 / 17633
[t-SNE] Computed conditional probabilities for sample 4000 / 17633
[t-SNE] Computed conditional probabilities for sample 5000 / 17633
[t-SNE] Computed conditional probabilities for sample 6000 / 17633
[t-SNE] Computed conditional probabilities for sample 7000 / 17633
[t-SNE] Computed conditional probabilities for sample 8000 / 17633
[t-SNE] Computed conditional probabilities for sample 9000 / 17633
[t-SNE] Computed conditional probabilities for sample 10000 / 17633
[t-SNE] Computed conditional probabilities for sample 11000 / 17633
[t-SNE] Computed conditional probabilities for sample 12000 / 17633
[t-SNE] Computed conditional probabilities for sample 13000 / 17633
[t-SNE] Computed conditional probabilities for sample 14000 / 17633
[t-SNE] Computed conditional probabilities for sample 15000 / 17633
[t-SNE] Computed conditional probabilities for sample 16000 / 17633
[t-SNE] Computed conditional probabilities for sample 17000 / 17633
[t-SNE] Computed conditional probabilities for sample 17633 / 17633
[t-SNE] Mean sigma: 0.059542
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.886734
[t-SNE] KL divergence after 300 iterations: 3.414036
```

In [139…

```
plt.figure(figsize=[15,7])
sns.scatterplot(x=tnse_results[:,0], y=tnse_results[:,1])
```

Out[139…

```
<AxesSubplot:>
```



There are no clusters here that jump out at me here to delve into further. They are all of a similar distribution and no cluster appears massively obvious.

Making Recommendations

In [140…

```
# Creating dataset for making recommendations for a user
artist_data = np.array(list(set(sub.artID)))
```

```
artist_data
```

Out[140…

```
array([    0,      1,      2, ..., 17629, 17630, 17631])
```

Let's pick a random user to generate recommendations for. We will go with the user of ID equalling 83.

In [141…

```
user = np.array([83 for i in range(len(artist_data))])
user
```

Out[141…

```
array([83, 83, 83, ..., 83, 83, 83])
```

In [142…

```
predictions = model.predict([user, artist_data])

predictions = np.array([a[0] for a in predictions])

recommended_artist_ids = (predictions).argsort()[:10]

recommended_artist_ids
```

Out[142…

```
array([8131, 8135, 8324, 2754, 8139, 8317, 8151, 8136, 8315, 2729],
      dtype=int64)
```

In [143…

```
predictions[recommended_artist_ids]
```

Out[143…

```
array([-1.1105492 , -1.0870131 , -1.0431927 , -0.9887378 , -0.98679465,
       -0.9674966 , -0.9587735 , -0.9441766 , -0.9026226 , -0.89995944],
      dtype=float32)
```

In [144…

```
df[df['artID'].isin(recommended_artist_ids)]
```

Out[144…

	name	url	pictureURL	artID
2729	Deathspell Omega	http://www.last.fm/music/Deathspell+Omega	http://userserve-ak.last.fm/serve/252/54245367...	2729
2754	Dark Funeral	http://www.last.fm/music/Dark+Funeral	http://userserve-ak.last.fm/serve/252/13074411...	2754
8131	Nahash	http://www.last.fm/music/Nahash	http://userserve-ak.last.fm/serve/252/93322.jpg	8131
8135	Krabaras	http://www.last.fm/music/Krabaras	http://userserve-ak.last.fm/serve/252/44372179...	8135
8136	DiktatÅ«ra	http://www.last.fm/music/Diktat%C5%ABra	http://userserve-ak.last.fm/serve/252/165489.jpg	8136
8139	Luctus	http://www.last.fm/music/Luctus	http://userserve-ak.last.fm/serve/252/8688989.jpg	8139
8151	Anubi	http://www.last.fm/music/Anubi	http://userserve-ak.last.fm/serve/252/27058983...	8151
8315	Gallhammer	http://www.last.fm/music/Gallhammer	http://userserve-ak.last.fm/serve/252/607745.jpg	8315
8317	Ossastorium	http://www.last.fm/music/Ossastorium	http://userserve-ak.last.fm/serve/252/51798.jpg	8317
8324	Dissimulation	http://www.last.fm/music/Dissimulation	http://userserve-ak.last.fm/serve/252/16340589...	8324

Let's check does our recommender produce novel results. In our recommender system as mentioned earlier it can be hard to gauge good recommenders in a formal manner (not a subjective manner such as a user rating the recommendations) as there is no ratings just listening figures. Users would eventually tire of being suggested the same artists so checking our recommender to see if it produces novel results is a good barometer of it's quality to start.

In [145…
```
test = played[played['userID'] == 83]
```

In [146…
```
test[test['artID'].isin(recommended_artist_ids)]
```

Out[146…

name	url	artID	userID	artistID	played	playedUserNorm	playCountScaled

Our recommender has produced completely novel results here for the user. This is potentially a good starting point for our recommender. Let's delve further into it's results looking at the recommendations for this particular user.

In [147…
```
usert = tags[tags['userID'] == 83]
```

In [148…
```
m = usert['tagValue'].unique()
```

In [149…
```
tagtest =  pd.merge(usert, test, how="inner", left_on=['userID', 'artistID'], right_on=['u
```

In [150…
```
tagtest
```

Out[150…

	tagID	tagValue	userID	artistID	name	url	artID	played	playedUserNorm	pl
0	102	hip-hop	83	288	Leona Lewis	http://www.last.fm/music/Leona+Lewis	288	76	0.575758	
1	103	rap	83	288	Leona Lewis	http://www.last.fm/music/Leona+Lewis	288	76	0.575758	
2	167	rnb	83	288	Leona Lewis	http://www.last.fm/music/Leona+Lewis	288	76	0.575758	
3	304	hip hop	83	288	Leona Lewis	http://www.last.fm/music/Leona+Lewis	288	76	0.575758	

In [151…
```
zzzz = df[df['artID'].isin(recommended_artist_ids)]
```

In [152…
```
zzzz1 = pd.merge(zzzz, tags, how="inner", left_on=['artID'], right_on=['artistID'])
```

In [206…
```
zzzz1.head()
```

Out[206…

	name	url	pictureURL	artID	tagID	tagValue	userII
0	Deathspell Omega	http://www.last.fm/music/Deathspell+Omega	http://userserve-ak.last.fm/serve/252/54245367…	2729	16	new wave	66

	name	url	pictureURL	artID	tagID	tagValue	userID
1	Deathspell Omega	http://www.last.fm/music/Deathspell+Omega	http://userserve-ak.last.fm/serve/252/54245367...	2729	16	new wave	152
2	Deathspell Omega	http://www.last.fm/music/Deathspell+Omega	http://userserve-ak.last.fm/serve/252/54245367...	2729	16	new wave	156
3	Deathspell Omega	http://www.last.fm/music/Deathspell+Omega	http://userserve-ak.last.fm/serve/252/54245367...	2729	18	electronic	26
4	Deathspell Omega	http://www.last.fm/music/Deathspell+Omega	http://userserve-ak.last.fm/serve/252/54245367...	2729	18	electronic	45

Let's Evaluate our alternate system's recommendations

Let's try calculate precision here for values. We will look to see what tags have been assigned by user 83 that has been assigned to the recommended artists our new system has left them. If at least one tag is found in both the user and the recommended artists tags we assign a score of 1 (relevant) to our user. I will use a method of "precision at K". As per Wikipedia#Precision_at_k), it is defined as "Precision at k documents (P@k) is still a useful metric (e.g., P@10 or "Precision at 10" corresponds to the number of relevant results among the top 10 retrieved documents), but fails to take into account the positions of the relevant documents among the top k". It is commonly used for evaluating music recommender systems. Out of the first K artists recommended we see how many of these are deemed relevant. As mentioned earlier our relevance can be deemed as an artist who has been given a tag by other users that is also a tag given by the user in question, user 83. I feel this is the most practical method of evaluating this system. Anyone with experience of using Spotify or SoundCloud would see that for recommendations you would only look at the first few maybe and lose interest after that. It is also hoped the first few recommendations are of a higher quality and more likely to attract a user's attention.

In [154...

```
d = {}
valz = np.array(zzzz1['tagValue'])
vals = np.array(zzzz1['name'])
for i in range(len(valz)):
    if vals[i] not in d:
        d[vals[i]] = ""
    else:
        continue
print(d)
```

```
{'Deathspell Omega': '', 'Dark Funeral': '', 'Nahash': '', 'Krabaras': '', 'DiktatÅ«ra':
'', 'Luctus': '', 'Anubi': '', 'Gallhammer': '', 'Ossastorium': '', 'Dissimulation': ''}
```

In [155...

```
valz = np.array(zzzz1['tagValue'])
vals = np.array(zzzz1['name'])
for i in range(len(valz)):
    if vals[i] in d and valz[i] in m:
        if valz[i] not in d[vals[i]]:
            d[vals[i]] += valz[i] + ","
    else:
        continue
```

In [156...

```
def strip_dict(d):
    return dict((k.strip(), v.strip()) for k, v in d.items())
```

In [157...

```
strip_dict(d)
```

Out[157…
```
{'Deathspell Omega': 'electronic,dance,electropop,',
 'Dark Funeral': 'rock,female vocalists,',
 'Nahash': '',
 'Krabaras': '',
 'DiktatÅ«ra': '',
 'Luctus': '',
 'Anubi': '',
 'Gallhammer': 'rock,',
 'Ossastorium': 'electronic,',
 'Dissimulation': ''}
```

In [158…
```
for key, value in d.items():
    d[key] = value.split(",")
```

In [159…
```
for k, v in d.items():
    v.pop()
```

In [160…
```
d
```

Out[160…
```
{'Deathspell Omega': ['electronic', 'dance', 'electropop'],
 'Dark Funeral': ['rock', 'female vocalists'],
 'Nahash': [],
 'Krabaras': [],
 'DiktatÅ«ra': [],
 'Luctus': [],
 'Anubi': [],
 'Gallhammer': ['rock'],
 'Ossastorium': ['electronic'],
 'Dissimulation': []}
```

In [161…
```
l1 = list(d.items())
#print(l1)
count = 0
for i in range(len(l1)):
    if len(l1[i][1]) > 0:
        count += 1
    else:
        continue
print("Precision at k equal 10 for user 83 with tags information is: " + str(count / len(l
```

```
Precision at k equal 10 for user 83 with tags information is: 0.4
```

It is interesting to note that for this user there is a precision score of 0.4. While as discussed below we thought the results were poor there appears to be some sense to our neural network recommender after all.

Looking at the results at face value the recommendations didn't seem incredibly accurate for this user. Going by the tags they left they appear to be a keen fan of 'rnb', 'hip-hop' and 'rap'. We also see they like 'pop' and 'soul' music. However, the genres returned here didn't appear to align with this whatsoever. Initially it appeared to have just returned the most popular genres (rock was most popular tag from our earlier analysis) and misses the point. It could be definitely interpreted as good for a user to return completely different types of music opening their eyes to new artists. Admittedly, the user had a high normalised played value of over 50% for this artist (Leona Lewis) they left tags for so obviously it indicates they are a big fan of these genres / artist(s). However, with our precision value here for the top 5 recommended artists we see some sense in the recommendations provided. While they are maybe not massively obvious they do appear to make some type of sense now. The first recommended artist for example 'Dark Funeral' has two tags in common and the fact these tags were given multiple times gives this a more reliable outlook, the same is also true of 'Deathspell Omega' with 3 relevant tags found here.

Spotify Recommender system with regularized model

Here I will attempt to test my recommender system on my own personal spotify account to see what recommendations it provides. While it must be noted my music taste is not exactly mainstream for the most part, this should be a good test of the sturdiness of the recommender system. We will be using our first recommender system based on user and artist ID and each users uniquely normalised listening column. We will do this using the Tekore python library. This is the most popular spotify api library alongside "spotipy" but I felt tekore was more appropriate for the work at hand here.

In [162…

```
import tekore as tk
```

In [163…

```
# covering these details
client_id = '#'
client_secret = '#'
```

In [164…

```
# always use this link
redirect_url = 'https://example.com/callback'
```

In [165…

```
conf = (client_id, client_secret, redirect_url)
token = tk.prompt_for_user_token(*conf, scope = tk.scope.every)

spotify = tk.Spotify(token)

# paste in link from new webpage opened up by this spotify call above into cell below
```

```
Opening browser for Spotify login...
Please paste redirect URL: https://example.com/callback?code=AQD9_AD_lHtNQWuCI-9Glvyl_soA5
Z-PIbcHbr4yvkf51OOXJcpPB6hMR06-B69k-hxYijjtwu0XRNBLKRzJR0DIfoJ4ZERlu10lJu3soE108wn098icc1i
QwBK_Msnp8mXPNaKbviGagzMf9ZKhXhKsyXaBLFNDkl77l4JhQ3HvB-79mWXlKu0mHDO9XCZYeHiCmKt3CG9o4C4lq
acfvs-ouyRJlFlFT1jUY10YCLy6T2Q6V4cWaOH8lrl2Zq_TJQ8NQstfKSm04P055mRz1tOUcsomJ6cw7dc8j4V1SzX
58NT3Uo3QNe9n8Z87yIZt9LguS8iGOuD6nwBuz5KNX3gjtCeTpYqi0z99univtb19w-bycPZ8esnKfQHwSJZTveL4F
VeiH12101Sre6rZvAW-jYZpUwuSPA09mfjsaUvYajPpMb_MbxMAODjzkvPfInqqyaZxbzLX0fxDfsUfYhJNItBNZP9
CHX8F_tTkucw8a5f15NYAuOFmymxy19oYQPvQfZylN3MVRhqDFIO3Qpysl_aP_SJLaUws6B2cHk1bq0rbJOC5NNBKa
POFy10qKam4qsg-7XjgjnkBtegTQXegNlTg81hCVJvDK-lMj22qqUbVBwf7q-EcR2V_0_cV8WVdPANgqIuAe3EMRlk
z5dA7_jkO8iRjOTiuPn1n6T9RvGo&state=Dv_sgeFAZ4pq8LyhTf-nb6sZq8_jrosfZwbYP5DOsfI
```

In [166…

```
artists = spotify.current_user_top_artists(limit = 10)
spotify_artists = artists.items
```

In [167…

```
t = np.array(df['name'])
for i in range(len(spotify_artists)):
    if spotify_artists[i].name in t:
        artist_neighbors(reg_model, spotify_artists[i].name, DOT)
        artist_neighbors(reg_model, spotify_artists[i].name, COSINE)
    else:
        print(str(spotify_artists[i].name) + " is not in the LastFM data.")
```

```
Nearest neighbors of : Calvin Harris.
```

	dot score	names
61	0.638375	Madonna
167	0.636582	Placebo

	dot score	names
49	0.559306	Kylie Minogue
492	0.545712	Paramore
1089	0.540387	BjÃ¶rk
148	0.536804	Radiohead

```
Nearest neighbors of : Calvin Harris.
```

	cosine score	names
1400	1.000000	Calvin Harris
1401	0.828489	Pendulum
414	0.813956	The Kills
1919	0.810205	Adele
166	0.803162	Garbage
5127	0.801854	Alejandro Sanz

```
Mark Blair is not in the LastFM data.
Bissett is not in the LastFM data.
Tobu is not in the LastFM data.
Nearest neighbors of : David Guetta.
[Found more than one matching artist. Other candidates: Chris Willis; David Guetta; Fergi
e; LMFAO]
```

	dot score	names
283	1.611553	Britney Spears
83	1.068002	Lady Gaga
282	0.980633	Rihanna
286	0.888733	Christina Aguilera
673	0.883012	Glee Cast
294	0.833620	Katy Perry

```
Nearest neighbors of : David Guetta.
[Found more than one matching artist. Other candidates: Chris Willis; David Guetta; Fergi
e; LMFAO]
```

	cosine score	names
322	1.000000	David Guetta
399	0.877570	The Rasmus
532	0.873988	Maroon 5
300	0.850070	Black Eyed Peas
346	0.848968	Cheryl Cole
343	0.847734	The Pussycat Dolls

```
Low Steppa is not in the LastFM data.
Uniting Nations is not in the LastFM data.
Sonny Fodera is not in the LastFM data.
Nearest neighbors of : Eric Prydz.
```

	dot score	names

	dot score	names
61	0.168536	Madonna
282	0.128524	Rihanna
83	0.120175	Lady Gaga
49	0.089464	Kylie Minogue
283	0.086551	Britney Spears
167	0.081481	Placebo

```
Nearest neighbors of : Eric Prydz.
```

	cosine score	names
2494	1.000000	Eric Prydz
12502	0.643813	I Would Set Myself on Fire for You
3803	0.580515	Elefante
13600	0.552894	Rob Rock
5275	0.535097	Dale Hawkins
12744	0.528561	The Devlins

```
Pete Heller's Big Love is not in the LastFM data.
```

Personal Opinions on these recommendations

As referenced earlier some of these artists aren't exactly mainstream, massively popular artists to a wider audience, hence the limited number of found artists. Calvin Harris and David Guetta are popular artists and the recommendations made by our recommender are actually good between dot score and cosine scores. The recommendations for Eric Prydz are not quite as good it has to be said especially for cosine scores. This may be due to lack of appropriate artists similar to Eric Prydz in the LastFM data as it would be a more obscure artist. The cosine and dot scores provided for this artists would appear to suggest this as they are noticeably lower than previous scores. However, on the whole I definitely find these recommendations quite good. These artists recommended are definitely not in my more popular artists and it is a good sign of the recommender system it is showing new artists. The recommendations for David Guetta and Calvin Harris are in my opinion very logical. There is artists that have collaborated with the pair in both sets of recommendations and other artists of a similar ilk who are not in my top artists. The Eric Prydz results do highlight some of the limitations of our approach. With the values scaled from just 0 to 1 scores with similar listening values will be aligned regardless of genre of user's preferences. It must also be noted this dataset is from 2011 so the recommendations are based off a slightly more limited market of artists too which is why some of these artists could not be found and why some recommendations may be slightly more predictable.

Evaluation

We will now try to evaluate our results and methods attempted so far. I will look at using methods such as Recall, Precision, Coverage and F1 to validate our methods. I feel methods such as mean absolute error (MAE) and root mean square error (RMSE) are not suitable to our data provided. These look at the difference between the actual and predicted values, which are not really applicable to the data here. The predicted artist could be number 1 and the actual artist could be 2 and this would appear as a very good result using this method. However, the difference in genres and similarity of artist could be completely different and hence the result is misleading. We will look to implement evaluation using the "LightFM" python library.

In [168…

```
import implicit
from tqdm import tqdm_notebook as tqdm
import matplotlib.ticker as ticker
from matplotlib import rc
from pandas.api.types import CategoricalDtype
import string
import re
import random
import math
from math import sqrt
from math import log
from collections import Counter, defaultdict
from operator import itemgetter
from pylab import rcParams
from pylab import savefig
```

In [175…

```
# Create sparse matrix from dataframe object
def create_sparse_matrix(data):
  #get unique user ids and unique artist ids
  users = list(np.sort(data.userID.unique()))
  artists = list(data.artistID.unique())
  plays = list(data.playCountScaled)

  cat_type = CategoricalDtype(categories=users, ordered=True)
  rows = data.userID.astype(cat_type).cat.codes

  cat_type = CategoricalDtype(categories=artists, ordered=True)
  cols = data.artistID.astype(cat_type).cat.codes
  # we get the rows (user ids) and columns (artist ids) and populate them using plays
  plays_sparse = scipy.sparse.csr_matrix((plays, (rows, cols)), shape=(len(users),len(arti
  return plays_sparse
```

In [170…

```
# Calculate sparsity of matrix
def calculate_sparsity(M):
  matrix_size = float(M.shape[0] * M.shape[1]) # Number of total possible interactions be
  num_plays = len(M.nonzero()[0]) # Number of times any artist has been interacted with
  sparsity = 100 * (1 - float(num_plays / matrix_size))
  return sparsity
```

In [171…

```
def evaluate_lightfm(model, original, train, test, user_features=None, item_features=None,
    print("Evaluating LightFM...")
    print("Calculating Coverage...")
    catalog = []
    for user in tqdm(range(0, original.shape[0])):
      #get scores for this particular user for all artists
      rec_scores = model.predict(user,np.arange(original.shape[1]),user_features=user_feat
      #get top k items to recommend
      rec_items = (-rec_scores).argsort()[:20]
      #calculate coverage
      #coverage calculation
      for recs in rec_items:
        if recs not in catalog:
          catalog.append(recs)

    coverage = len(catalog)/float(original.shape[1])
    print("Calculating Recall at k...")
    recall = recall_at_k(model, test, user_features = user_features, item_features = item_
    print("Calculating Precision at k...")
    precision = precision_at_k(model, test, user_features = user_features, item_features =
```

```
    f1 = (2 * precision * recall) / (precision + recall)
    return coverage, precision, recall, f1
```

In [174…

```
playedx = played[['userID','artistID', 'playedUserNorm']]
playedx.columns = ['userID','artistID', 'playedUserNorm']

#create sparse matrix like earlier
plays_sparse_light = create_sparse_matrix(playedx).astype('float')
print('Matrix Sparsity:', calculate_sparsity(plays_sparse_light))

train_ratings, test_ratings = split_dataframe(xyz)
# SparseTensor representation of the train and test datasets.
A_train = build_rating_sparse_tensor(train_ratings)
A_test = build_rating_sparse_tensor(test_ratings)

train_light, test_light = lightfm.cross_validation.random_train_test_split(plays_sparse_li
model_fm_vanilla = LightFM(learning_rate=0.05, loss='bpr')

#train model
print("Fitting model...")
model_fm_vanilla.fit(train_light, epochs=10)

#evaluate model
coverage, precision, recall, f1 = evaluate_lightfm(model_fm_vanilla, plays_sparse_light, t
print("Precision:", precision * 100, '%')
print("Recall:", recall * 100, '%')
print("Coverage:", coverage * 100, '%')
print("F1:", f1 * 100, '%')
```

```
Matrix Sparsity: 99.72171848800758
Fitting model...
Evaluating LightFM...
Calculating Coverage...
C:\Users\user\AppData\Local\Temp/ipykernel_24744/314659252.py:5: TqdmDeprecationWarning: T
his function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for user in tqdm(range(0, original.shape[0])):

Calculating Recall at k...
Calculating Precision at k...
Precision: 13.034957647323608 %
Recall: 7.578250467064537 %
Coverage: 0.8507259528130671 %
F1: 9.584357110337042 %
```

In [176…

```
playedx = played[['userID','artistID', 'playCountScaled']]
playedx.columns = ['userID','artistID', 'playCountScaled']

#create sparse matrix like earlier just compatible with lightFM
plays_sparse_light = create_sparse_matrix(playedx).astype('float')
print('Matrix Sparsity:', calculate_sparsity(plays_sparse_light))

# split up data like we did earlier with our split_dataframe function
train_light, test_light = lightfm.cross_validation.random_train_test_split(plays_sparse_li
model_fm_vanilla = LightFM(learning_rate=0.05, loss='bpr')

#train model
print("Fitting model...")
model_fm_vanilla.fit(train_light, epochs=10)

#evaluate model
coverage, precision, recall, f1 = evaluate_lightfm(model_fm_vanilla, plays_sparse_light, t
print("Precision:", precision * 100, '%')
```

```
print("Recall:", recall * 100, '%')
print("Coverage:", coverage * 100, '%')
print("F1:", f1 * 100, '%')
```

```
Matrix Sparsity: 99.72362497745786
Fitting model...
Evaluating LightFM...
Calculating Coverage...
```

```
C:\Users\user\AppData\Local\Temp/ipykernel_24744/314659252.py:5: TqdmDeprecationWarning: T
his function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for user in tqdm(range(0, original.shape[0])):
```

```
Calculating Recall at k...
Calculating Precision at k...
Precision: 15.01588225364685 %
Recall: 8.751779254405273 %
Coverage: 0.8053539019963702 %
F1: 11.058360684709108 %
```

Our chosen accuracy when normalising with each user's max score is actually worse than our play count scaled metric. However, when I ran this with playCountScaled used for our recommender the recommendations were dreadful with very low dot and cosine scores. Of the sample of artists and my spotify recommendations the recommendations using the playedUserNorm column seemed far more accurate. This is probably because the scores are better scaled between 0 and 1, identifying popular artists more easily with values closer to 1 and also less known artists would have a better chance of being noticed if a handful of users listened to them a lot. However, it could also be said that it may produce more novel recommendations like our alternate approach. By this I mean that while on initial inspection the results seemed poor the recommender actually makes relevant recommendations to the user that they have not seen before.

Clustering Attempt

In [177…

```
merged_df.head()
```

Out[177…

	artistID	name	mean	med	max	sum	unique
0	283	Britney Spears	4584.559387	1000.5	131733	2393140	522
1	66	Depeche Mode	4614.567376	567.0	352698	1301308	282
2	83	Lady Gaga	2113.563011	590.0	114672	1291387	611
3	286	Christina Aguilera	2600.503686	739.0	176133	1058405	407
4	492	Paramore	2414.659148	417.0	227829	963449	399

In [178…

```
mergeddf_sub = merged_df[['artistID', 'mean', 'max', 'sum', 'unique']]
```

In [179…

```
mergeddf_sub.head()
```

Out[179…

	artistID	mean	max	sum	unique
0	283	4584.559387	131733	2393140	522
1	66	4614.567376	352698	1301308	282
2	83	2113.563011	114672	1291387	611

	artistID	mean	max	sum	unique
3	286	2600.503686	176133	1058405	407
4	492	2414.659148	227829	963449	399

In [180…

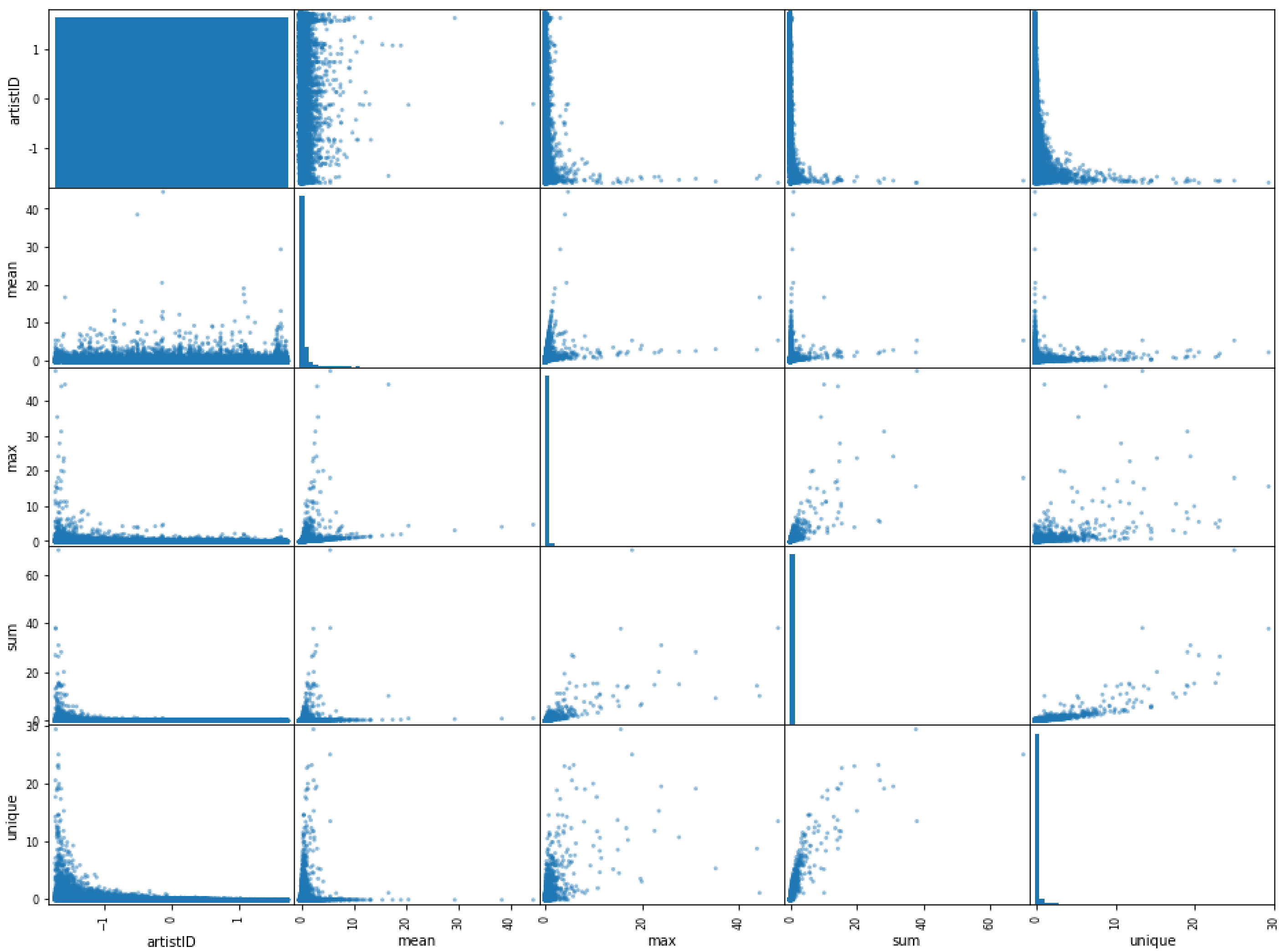
```
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(mergeddf_sub)
X_scaled = scaler.transform(mergeddf_sub)
X_scaled.std(axis=0)

standard_all = pd.DataFrame(X_scaled)
```

In [181…

```
standard_all = standard_all.rename(columns={0:'artistID', 1: 'mean', 2: 'max', 3: 'sum',
                                            4: 'unique'})
```

In [237…

```
pd.plotting.scatter_matrix(standard_all, figsize=(16,12), hist_kwds=dict(bins=50), cmap="
plt.show()
```

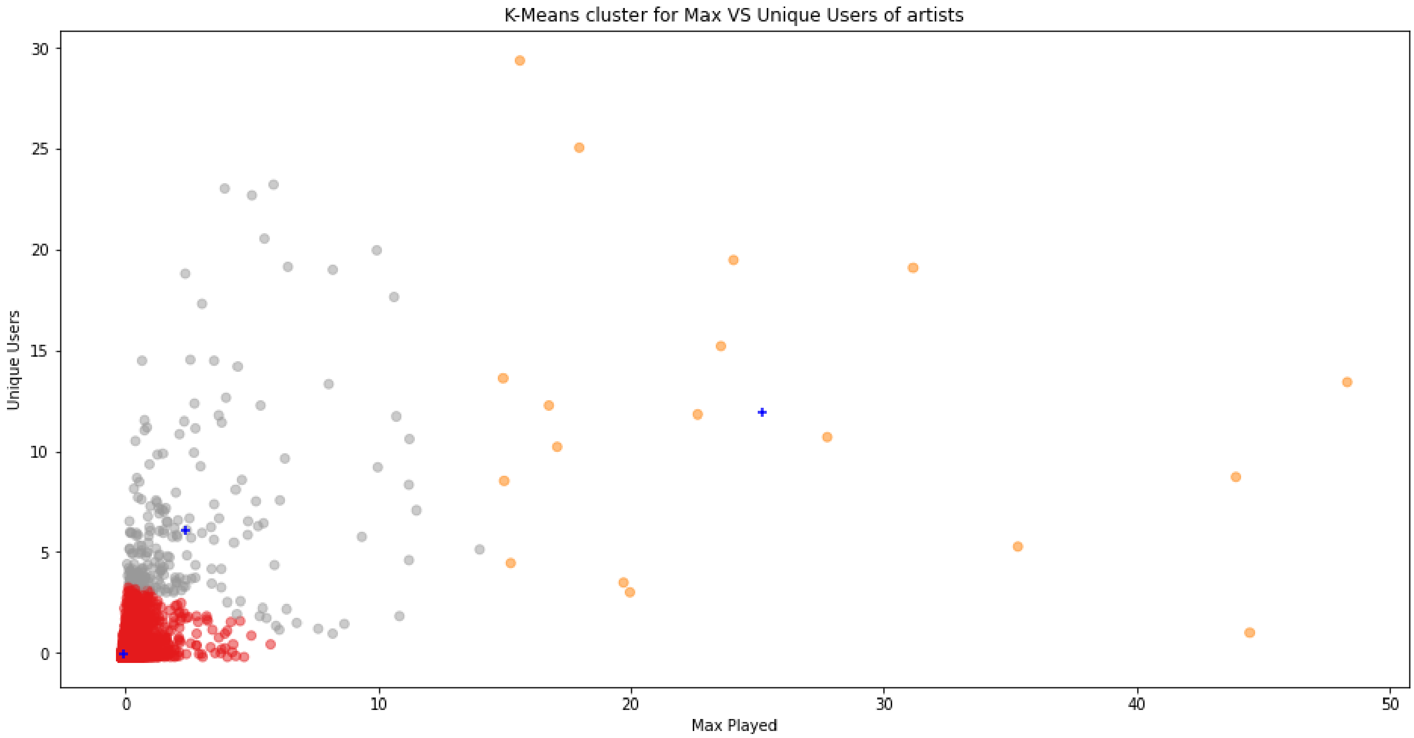


In [238…

```
kmeans_margin_standard = KMeans(n_clusters=3).fit(standard_all[["max", "unique"]])
centroids_betas_standard = kmeans_margin_standard.cluster_centers_
```

In [242…

```
plt.figure(figsize=(16,8))
plt.scatter(standard_all['max'], standard_all['unique'], c= kmeans_margin_standard.labels_
plt.scatter(centroids_betas_standard[:, 0], centroids_betas_standard[:, 1], c='blue', mark
plt.title('K-Means cluster for Max VS Unique Users of artists')
plt.xlabel('Max Played')
plt.ylabel('Unique Users')
plt.show()
```



Our results here are not amazing it must be said. The only things we can deduce from this is that artists with low numbers of unqiue users listening to them very much tend to have lower values for highest listened value. There also seems to be the case that can be made from this cluster that even artists with high values for both highest played value and number of unique users that there isn't a major correlation between the pair.

Conclusions

Difference of systems

Our systems certainly differed in how they were used and implemented. Our NN based system which utilised artist and user embeddings was developed solely for existing users to make recommendations for them. This was maybe a potential drawback of this system. Generating personalized data can be done but could be a timely process for this system while trying to implement spotify api libraries such as the one I used for my regularized model could potentially be tricky. I feel on the whole the regularized model was the best throwing up some very solid recommendations despite what appeared to be some misleading tag information.

Future Work

One thing I would definitely like to do in the future is to generate data from my soundcloud or spotify data and append it to the dataset provided as an example for my NN system. This is something I definitely considered doing but time constraints prevented me from doing so. This way I could designate myself a user ID and generate recommendations for myself. I feel this would allow for a more practical test of the system. I would also like to test another system I found more thoroughly. I have left it on my git repository but decided against using it for the final submission as I already had enough systems tested and used. I would also like to test novelty of the regularized model in more detail. Again time constraints were a limitation of this process and I would like to test this the same way as I did for the NN system as this is a key part of recommender systems. This file can be found here.