# Individual Coursework

## MSIN0097 Predictive Analytics

# COURSEWORK: WARNER MUSIC

### PREDICTING THE SUCCESS OF ARTISTS ON SPOTIFY

Please complete the sections of this Notebook with supporting code and markup analysis where appropriate. During this coursework you will:

- Understand the specific business forecast task
- Prepare a dataset, clean and impute where necessary
- Train an ensemble classifier
- Evaluate the performance and comment of success and failure modes
- Complete all necessary stages of the data science process

2000 words plus code, markup text within cells of the Notebook.

This is approximately 100 words per ACTION cell, but use the wordcount over the duration of the Notebook at your discretion.

# Assessement

**Assessment Deadlines**

Please see the Business Analytics tab on the School of Management Student Information Centre for full details of coursework submission deadlines.

- 21/02/2019

**Assessment Type**:

- 60% Individual Coursework of 2000 words plus code, markup text within cells of the Notebook. Approximately 100 words per ACTION cell.

## 0. Business Case Understanding

### INTRODUCTION

Over the last few years, the music industry has been dominated by digital streaming services, which produce vast amounts of data on listeners and their preferences.

This has required major players in the industry to adopt a data driven approach to content delivery in order to stay competitive.

Warner Music Group is looking to leverage its rich database to better understand the factors that have the most significant impact on the success of a new artist. This will allow them to optimize the allocation of resources when signing and promoting new artists.

Warner's (large) database contains several sources of data, including the streaming platforms Spotify, Amazon Live and Apple Music.

For this case study, we will be looking using the Spotify dataset to predict the success of artists. In particular, we want to understand the role of Spotify playlists on the performance of artist.

### Streaming Music

When artists release music digitally, details of how their music is streamed can be closely monitored.

Some of these details include:

- How listeners found their music (a recommendation, a playlist)
- Where and when (a routine visit to the gym, a party, while working).
- On what device (mobile / PC)
- And so on…

Spotify alone *process nearly 1 billion streams every day* (Dredge, 2015) and this streaming data is documented in detail every time a user accesses the platform.

Analyzing this data potentially enables us to gain a much deeper insight into customers' listening behavior and individual tastes.

Spotify uses it to drive their recommender systems – these tailor and individualize content as well as helping the artists reach wider and more relevant audiences.

Warner Music would like to use it to better understand the factors that influence the *future success of its artists*, *identify potentially successful acts* early on in their careers and use this analysis to make resource decisions about how they market and support their artists.

### What are Spotify Playlists and why are relevant today?

A playlist is a group of tracks that you can save under a name, listen to, and update at your leisure.

**Figure 1. Screen shot of Spotify product show artists and playlists.**

Spotify currently has more than two billion publicly available playlists, many of which are curated by Spotify's in-house team of editors.

The editors scour the web on a daily basis to remain up-to-date with the newest releases, and to create playlists geared towards different desires and needs.

Additionally, there are playlists such as Discover Weekly (https://www.spotify.com/uk/discoverweekly/) and Release Radar (https://support.spotify.com/uk/using_spotify/playlists/release-radar/) that use self-learning algorithms to study a user's listening behavior over time and recommend songs tailored to his/her tastes.

The figure below illustrates the progression of artists on Spotify Playlists:

**Figure 2. Figure to illustarte selecting artists and building audience profiles over progressively larger audiences of different playlists.**

The artist pool starts off very dense at the bottom, as new artists are picked up on the smaller playlists, and thins on the way to the top, as only the most promising of them make it through to more selective playlists. The playlists on the very top contain the most successful, chart-topping artists.

An important discovery that has been made is that certain playlists have more of an influence on the popularity, stream count and future success of an artist than others.

**Figure 3. Figure to illustrate taking song stream data and using it to predict the trajectory, and likely success, of Warner artists.**

Moreover, some playlists have been seen to be pivotal in the careers of successful artists. Artists that do make it onto one of these *key* playlists frequently go on to become highly ranked in the music charts.

It is the objective of Warner's A&R (https://en.wikipedia.org/wiki/Artists_and_repertoire) team to identify and sign artists before they achieve this level of success i.e. before they get selected for these playlists, in order to increase their ROI.

### BUSINESS PROBLEM → DATA PROBLEM

Now that we have a better understanding of the business problem, we can begin to think about how we could model this problem using data.

The first thing we can do is defining a criterion for measuring artist success.

Based on our business problem, one way in which we can do this is to create a binary variable representing the success / failure of an artist and determined by whether a song ends up on a key playlist (1), or not (0). We can then generate features for that artist to determine the impact they have on the success of an artist.

Our problem thus becomes a classification task, which can be modeled as follows:

## *Artist Feature 1 + Artist Feature 2 …. + Artist Feature N = Probability of Success*

where,

**Success (1) = Artist Features on Key Playlist**

The key playlists we will use for this case study are the 4 listed below, as recommended by Warner Analysts:

1. Hot Hits UK
2. Massive Dance Hits
3. The Indie List
4. New Music Friday

The coursework task is to take a look at the Spotify dataset to see how we might be able to set up this classification model.

Complete the code sections below to work through the project from start to finish.

In [1]:
```
# Python Project Template

# 1. Prepare Problem
# a) Load libraries
# b) Load dataset

# 2. Summarize Data
# a) Descriptive statistics
# b) Data visualizations

# 3. Prepare Data
# a) Data Cleaning
# b) Feature Selection
# c) Data Transforms

# 4. Evaluate Algorithms
# a) Split-out validation dataset
# b) Test options and evaluation metric
# c) Spot Check Algorithms
# d) Compare Algorithms

# 5. Improve Accuracy
# a) Algorithm Tuning
# b) Ensembles

# 6. Finalize Model
# a) Predictions on validation dataset
# b) Create standalone model on entire training dataset
# c) Save model for later use
```

Review "R6 - Predictive Modeling template" Notebook from week 6 on Moodle for further details of what each stage of the project should look like.

**ACTION: Guidance**

If you need to do something, instructions will appear in a box like this

## 1. Prepare the problem

Run your code on Sherlock. We have prepared some of the data for you already.

In addition, we have imported a custom module (spotfunc.py) containing useful functions written for this dataset.

In [2]:
```
# Preamble

#import sherlockml.filesystem as sfs
import pandas as pd
import random

#sfs.get('/input/spotfunc.py', 'spotfunc.py')
#sfs.get('/input/playlists_ids_and_titles.csv', 'playlists_ids_and_titles.csv')
#sfs.get('/input/new artists2015onwards.csv', 'newartists2015onwards.csv')

# Add more stuff here as necessary

# Import all required libraries
import pandas as pd


# Import custom functions from library, named 'spotfunc'
#import spotfunc as spotfunc_v2
```

## 2. Data Understanding

A year's worth of Spotify streaming data in the WMG database amounts to approximately 50 billion rows of data i.e. 50 billion streams (1.5 to 2 terabytes worth), with a total of seven years of data stored altogether (2010 till today).

For the purposes of this case study, we will be using a sample of this data. The dataset uploaded on the Sherlock server is about 16GB, containing data from 2015 - 2017. Given the limits on RAM and cores, we will be taking a further sample of this data for purposes of this case study: a 10% random sample of the total dataset, saved as 'cleaned_data.csv'.

*Note: The code for this sampling in included below, but commented out.*

We can begin with reading in the datasets we will need. We will be using 2 files:

1. Primary Spotify dataset
2. Playlist Name Mapper (only playlist IDs provided in primary dataset)

In [3]:
```
# %%time
# Sampling data to read in 10%
# sfs.get('/input/all_artists_with_date_time_detail.csv', 'client-data.csv')
# # Read in data
# # The data to load
# f = 'client-data.csv'
# # Count the lines
# num_lines = sum(1 for l in open(f))
# n = 10
# # Count the lines or use an upper bound
# num_lines = sum(1 for l in open(f))
# # The row indices to skip - make sure 0 is not included to keep the header!
# skip_idx = [x for x in range(1, num_lines) if x % n != 0]
# # Read the data
# data = pd.read_csv(f, skiprows=skip_idx )
```

Read in the data

In [4]:
```
%%time
# Read in sampled data
data = pd.read_csv('cleaned_data.csv')
print('rows:',len(data))

# Keep a copy of original data in case of changes made to dataframe
all_artists = data.copy()

# Load laylist data
playlist_ids_and_titles = pd.read_csv('playlists_ids_and_titles.csv',encoding = 'latin-1',error_bad_lines=False,warn_bad_lines=False
)

# Keep only those with 22 characters (data cleaning)
playlist_mapper = playlist_ids_and_titles[playlist_ids_and_titles.id.str.len()==22].drop_duplicates(['id'])
```

```
<string>:2: DtypeWarning: Columns (2,13) have mixed types. Specify dtype option on import or set low_memory=False.

rows: 3805499
CPU times: user 26.6 s, sys: 3.04 s, total: 29.6 s
Wall time: 28 s
```

Begin by taking a look at what the spotify data looks like:

**ACTION: Inspect the data**

Make sure you understand the data. Use methods like **`data.head()`** and **`data.info()`**

In [5]:
```
data.head()
```

Out[5]:

| | Unnamed: 0 | Unnamed: 0.1 | Unnamed: 0.1.1 | day | log_time | mobile | track_id | isrc | upc |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 9 | ('small_artists_2016.csv', 9) | 10 | 20160510T12:15:00 | True | 8f1924eab3804f308427c31d925c1b3f | USAT21600547 | 7.567991e+10 |
| **1** | 1 | 19 | ('small_artists_2016.csv', 19) | 10 | 20160510T12:15:00 | True | 8f1924eab3804f308427c31d925c1b3f | USAT21600547 | 7.567991e+10 |
| **2** | 2 | 29 | ('small_artists_2016.csv', 29) | 10 | 20160510T14:00:00 | True | 8f1924eab3804f308427c31d925c1b3f | USAT21600547 | 7.567991e+10 |
| **3** | 3 | 39 | ('small_artists_2016.csv', 39) | 10 | 20160510T10:45:00 | True | 8f1924eab3804f308427c31d925c1b3f | USAT21600547 | 7.567991e+10 |
| **4** | 4 | 49 | ('small_artists_2016.csv', 49) | 10 | 20160510T10:15:00 | True | 8f1924eab3804f308427c31d925c1b3f | USAT21600547 | 7.567991e+10 |

5 rows × 45 columns

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3805499 entries, 0 to 3805498
Data columns (total 45 columns):
Unnamed: 0           int64
Unnamed: 0.1         int64
Unnamed: 0.1.1       object
day                  int64
log_time             object
mobile               bool
track_id             object
isrc                 object
upc                  float64
artist_name          object
track_name           object
album_name           object
customer_id          object
postal_code          object
access               object
country_code         object
gender               object
birth_year           float64
filename             object
region_code          object
referral_code        float64
partner_name         object
financial_product    object
user_product_type    object
offline_timestamp    float64
stream_length        float64
stream_cached        float64
stream_source        object
stream_source_uri    object
stream_device        object
stream_os            object
track_uri            object
track_artists        object
source               float64
DateTime             object
hour                 int64
minute               int64
week                 int64
month                int64
year                 int64
date                 object
weekday              int64
weekday_name         object
playlist_id          object
playlist_name        object
dtypes: bool(1), float64(7), int64(9), object(28)
memory usage: 1.3+ GB
```

Each row in the data is a unique stream – every time a user streams a song in the Warner Music catalogue for at least 30 seconds it becomes a row in the database. Each stream counts as a ‘transaction’, the value of which is £0.0012, and accordingly, 1000 streams of a song count as a ‘sale’ (worth £1) for the artist. The dataset is comprised of listeners in Great Britain only.

Not all the columns provided are relevant to us. Lets take a look at some basic properties of the dataset, and identify the columns that are important for this study

The columns you should *focus* on for this case study are:

- Log Time – timestamp of each stream
- Artist Name(s) – some songs feature more than one artist
- Track Name
- ISRC - (Unique code identifier for that version of the song, i.e. radio edit, album version, remix etc.)
- Customer ID
- Birth Year
- Location of Customer
- Gender of Customer
- Stream Source URI – where on Spotify was the song played – unique playlist ID, an artist’s page, an album etc.

Here we create dataframe that contains the relevant columns.

In [7]:
```
keep_cols=["log_time","artist_name","year","playlist_id","playlist_name","track_name","isrc","customer_id","birth_year","region_code","gender","stream_source_uri"]
data1=data[keep_cols]
```

In [8]:
```
data1.head()
```

Out[8]:

| | log_time | artist_name | year | playlist_id | playlist_name | track_name | isrc | customer_id | birth_year | regic |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 20160510T12:15:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | 6c022a8376c10aae37abb839eb7625fe | 1968.0 | ( |
| 1 | 20160510T12:15:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | 6c022a8376c10aae37abb839eb7625fe | 1968.0 | ( |
| 2 | 20160510T14:00:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | 352292382ff3ee0cfd3b73b94ea0ff8f | 1995.0 | |
| 3 | 20160510T10:45:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | c3f2b54e76696ed491d9d8f964c97774 | 1992.0 | |
| 4 | 20160510T10:15:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | 6a06a9bbe042c73e8f1a3596ec321636 | 1979.0 | |

## EXPLORATORY ANALYSIS AND PLOTS

**ACTION: Exploratory analysis**

As demonstrated in class, explore various distribution of the data. Comment on any patterns you can see.

- Highlight on any potential uncertainties or peculiarities that you observe.
- Variables you might explore, include, but are not limited to: Age, Gender, Stream counts and playlists.
- Use figures, plots and visualization as necessary.

Now we look at the data set in more detail. At the beginning, we would like to see the age of users by using the columns 'birth_year' in the original data. From the bar chart we can see that most users were born between 1990 to 2000 which means that most users' age are between 20 to 30 years old.

In [9]:
```
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
data1['birth_year'].hist()
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7951882e48>

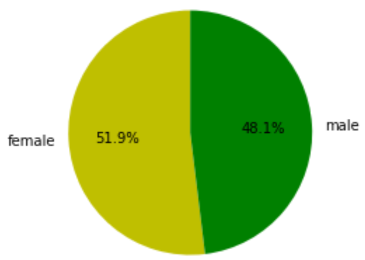
We would like to see the ratio of male and female of users then. From the pie chart below, we can see that 51.9% of all users are female and 48.1% of users are male.

In [10]:
```
data1['gender'].value_counts()
```

Out[10]:
```
female    1955719
male      1809358
Name: gender, dtype: int64
```

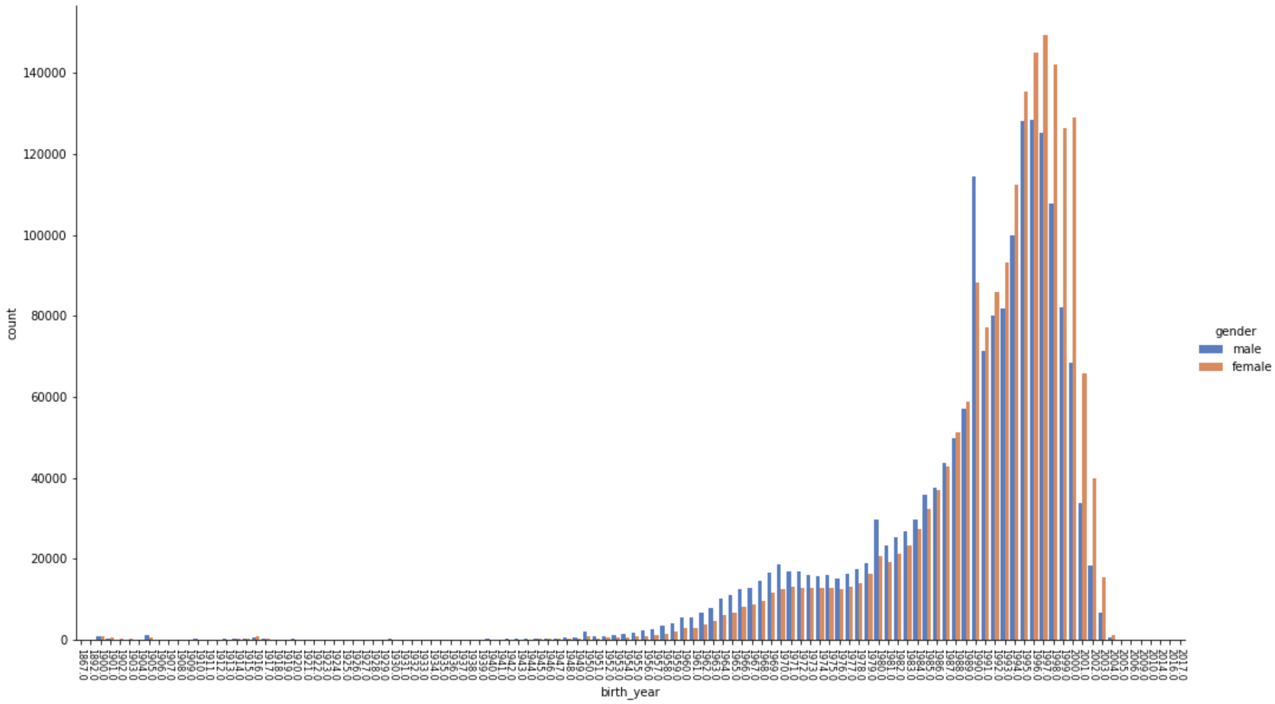
In [11]:
```
female_male=[1955719,1809358]
activities=['female','male']
colors=['y','g']
plt.pie(female_male, labels=activities, colors=colors,startangle=90,autopct='%.1f%%')
plt.show
```

Out[11]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```




If we look at the birth year by gender, the diagram below shows that for older generation, there are more men use spotify than women and for younger generation, there are more women use spotify. The reason might be men are more advanced in using technology or tend to like music more if people are older, and women are more advanced in using technology or tend to like music more if people are younger.

In [12]:
```
# Birth year by gender
sns.catplot(x="birth_year", hue="gender",
            data=data1[data1['customer_id'].isin(data1['customer_id'].unique())], kind="count",
            height=8.27, aspect=14/8.27, palette="muted")
plt.xticks(rotation=270, fontsize=8)
plt.show()
```



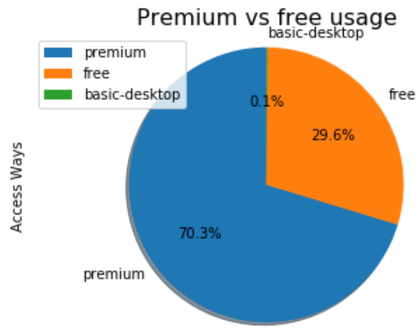

Moreover, the ratio of premium and free users is investigated.

In [13]:
```
data_access = data[data.access != "deleted"]
access = data_access["access"].value_counts()
access
```

Out[13]:
```
premium          2676048
free             1124692
basic-desktop       4753
Name: access, dtype: int64
```

In [14]:
```
# Plot access pie chart
labels = ["premium", "free", "basic-desktop"] # Add labels

access_pie = data_access["access"].value_counts().plot(kind="pie", label="Access Ways",
                                                        autopct='%1.1f%%', shadow=True, startangle=90)
access_pie.legend(labels, loc="best") # Add legends
access_pie.set_title("Premium vs free usage", fontsize=16) # Add title
access_pie.axis("equal") # Equal aspect ratio ensures that pie is drawn as a circle
```

Out[14]:
```
(-1.1134317323758902,
 1.1130837153020834,
 -1.1178022984784375,
 1.1008477284989733)
```



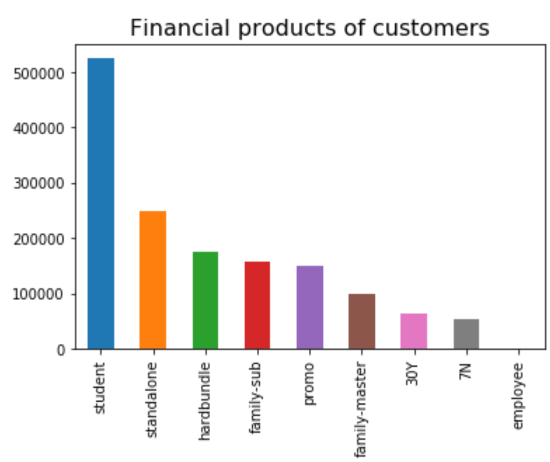

From the pie chart, we can see that 70% of customers pay for the premium service while 29.6% of them chose to use the free one. There is a very small number of people (0.1%) who use the desktop (which is also free but has the basic functions) to listen to the songs on Spotify. Therefore, there are still quite a lot potential customers that Spotify can get more revenue from.

In [15]:
```
# List the number of customers using different payment methods for premium service
data_financial = data[data.financial_product != "deleted"] # Drop the missing value (here is "deleted")
financial_product = data_financial["financial_product"].value_counts() # Number of customers with premium service paying it through
 different methods
financial_product
```

Out[15]:
```
student          525367
standalone       248559
hardbundle       176395
family-sub       156374
promo            150833
family-master     98953
30Y               64478
7N                54628
employee            807
Name: financial_product, dtype: int64
```

In [16]:
```
# Plot financial_product bar chart
financial_bar = data_financial["financial_product"].value_counts().plot(kind="bar")
financial_bar.set_title("Financial products of customers", fontsize=16) # Add title
```

Out[16]:
```
Text(0.5, 1.0, 'Financial products of customers')
```



The bar chart above shows the number of customers with premium service paying it through different payment methods. Based on that, we can observe that the top three most popular payment methods include student, stand alone, and hard bundle. Among them, the number of customers using student payment is significantly higher than other types. We assume the reason behind is because the main target market is represented by young people and most of them are students. Moreover, the special student discounts Spotify offered are relatively cheaper than other kinds of payments.

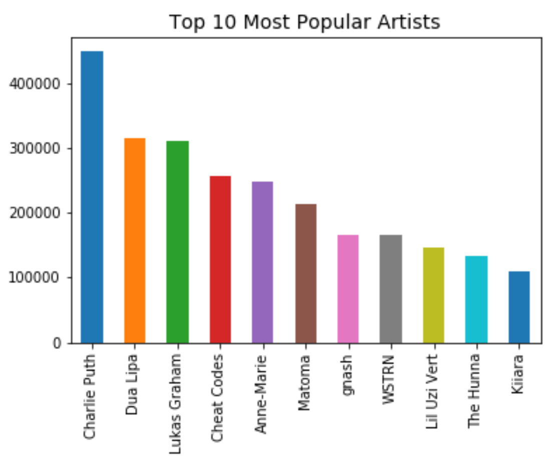
We would also like to list the top 10 most popular artists by stream counts. From the bar chart below, it can be noticed that Charlie Puth is the most popular artist who is significantly more popular than others.

In [17]:
```
#list popular artists
artists = data["artist_name"].value_counts()
artists[:11]
```

Out[17]:
```
Charlie Puth    447873
Dua Lipa        315663
Lukas Graham    311271
Cheat Codes     255820
Anne-Marie      247934
Matoma          212210
gnash           165683
WSTRN           164885
Lil Uzi Vert    146692
The Hunna       132287
Kiiara          109118
Name: artist_name, dtype: int64
```

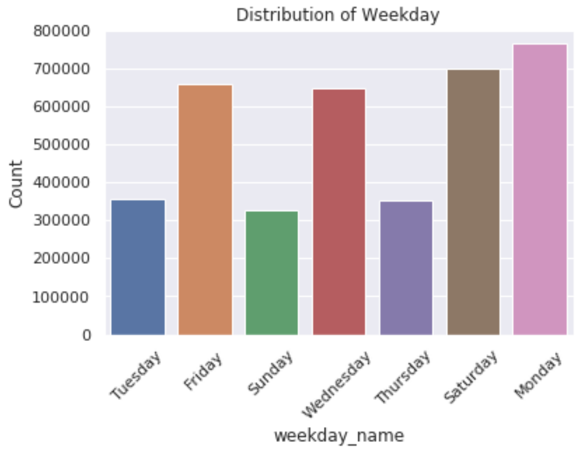
In [18]:
```
#plot popular artists
artists_bar = artists[:11].plot(kind="bar")
artists_bar.set_title("Top 10 Most Popular Artists", fontsize=14)
```

Out[18]:
```
Text(0.5, 1.0, 'Top 10 Most Popular Artists')
```




In [19]:
```
#Creating functions for drawing histogram plots

#Function for plotting of categorical variables
def categorical_plot(variable,xlabel,ylabel,title):
    sns.set(style="darkgrid")
    ax = sns.countplot(x=variable ,data=data)
    ax.set_xticklabels(ax.get_xticklabels(),rotation=45)
    #plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.show(ax)

#Functon for plotting of interval variables
def interval_plot(variable,xlabel,ylabel,title,bins):
    a=sns.distplot(data[variable], hist=True, kde=False,
                 bins=bins, color = 'blue',
                 hist_kws={'edgecolor':'black'})
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.show(a)
```

In [20]:
```
categorical_plot('weekday_name','Weekday','Count','Distribution of Weekday')
```




The bar chart above presents the distribution of weekday by the stream count. We can see that there are more people listening to music on Monday and Saturday.

---

## 3. Data Preperation and Feature Engineering

Since an important criteria for success is whether or not an artist has been on one of 4 key playlists. So we firstly define the 4 key playlists and then select the relevant playlists as the variable we are trying to model by using 'playlist_mapper'.

We can being by out data preperation by subsetting the 4 key playlists we are interested in and it can be noticed that there are 16 playlist id inside the 4 relevant playlists from the subset dataframe.

**ACTION: Dependant variable**

Set up the problem as one of classification, selecting the relevant playlists as the variable we are trying to model.

Write useful helper functions to support creating of the feature vector and target vector

In [21]:
```
# 4 key Playlists
key_playlists=["Hot Hits UK","Massive Dance Hits","The Indie List","New Music Friday"]
playlist_mapper["name"]= playlist_mapper["name"].str.upper()
playlist_mapper
```

Out[21]:

| | id | name |
|---|---|---|
| **0** | 607qZnoGjqhpWjOaJWakmx | 80ER JAREN |
| **1** | 4xP3wJiHkHfyPcGBjsZcpf | GLEE |
| **2** | 1iHOfbhKGHImcrEJXhrUdg | BEST OF 1980S |
| **3** | 08AR0IWSEfi0GCnB7b6AAW | KESÄHITIT/YHDEN HITIN IHMEET/SEKALAISTA |
| **4** | 3DeVsW7nzA3qezOMowGkeu | MÚSICAS PARA TRANSAR |
| **5** | 3ECQioqPAGUhkPp7PgP6lr | GUITAR HERO 6 - WARRIORS OF ROCK |
| **6** | 57sY0t33ZSRaJAS2ALqGFY | BEFORE PRE |
| **7** | 2vUYLeRET44sZhqEH8YqoE | B4.DA.$$ |
| **8** | 7jmQBEvJyGHPqKEl5UcEe9 | TOP TRACKS IN SWEDEN |
| **9** | 3mMc3888eoAg8sgnEzQN6s | #PLAYACHER |
| **10** | 20fZrm4df1vgbidX2TfRZZ | DOWN THE RABBIT HOLE 2016 |
| **11** | 2zASQX0dx4Wo3DFgh4dfNs | 100 ESSENTIALS |
| **12** | 4d8UPbcSLGjChEMYPMoTYF | WATCH THE STOVE ? |
| **13** | 2d7iZ08yABfObw0Ny5EGt1 | CHICAGO HOUSE MUSIC |
| **14** | 4fVl8YHzIpBiGhcEgxvhK6 | ELECTRÓNICA 2000-2016 |
| **15** | 5hHLFIci647cgwwNfpykgQ | OG VEELA SONGS |
| **16** | 6U01koIahEeETQT7ERVqIG | FRIDAY FRESH |
| **17** | 0T2xVB3F7cFR3F5eDYtrWN | AUNKK NO ZEA KONMIGO Y MAS #RETROOZ |
| **18** | 6eNJAHiYElyL0ZDZULPVGv | TUNOG KALYE |
| **19** | 2IAy5fQvooIGCE8nd9xSEB | PANCAKES |
| **20** | 5JVnupHU4BdOwnJNw3POa6 | BANGIN' IN SANGIN |
| **21** | 37i9dQZF1DZ06evO39S5CU | THIS IS: NEK |
| **22** | 0klA3IKXWnmJClwsOKjksC | DISNEY'S GREATEST CLASSICS |
| **23** | 1fm7mdOoADMy0508dlNbGE | TWITCH MUSIC LIBRARY |
| **24** | 5wR0ymENgJ86fGtPMnFNpx | VIERDAAGSE 2015 |
| **25** | 5XGo2h2IRmqGpRQNAy5spD | CALMAS |
| **26** | 2CQesR9NGKYbRmflXHxEe4 | POP LATINO |
| **27** | 72EHB4WBxLOth162hY6fDC | TINTINMARATON |
| **28** | 0dCGBp9cHWw3J44U90XUL5 | MARIMBA CHIAPAS ? MARIMBAS MEXICANAS |
| **29** | 2YPimJcWh0Pb3jYjqPBmXk | FIELD PRODUCTIONS IN THE HOUSE |
| **...** | ... | ... |
| **194525** | 4UwULfx1knj6uRxkVIxowF | FELIX'S BEDTIME PLAYLIST |
| **194526** | 37i9dQZF1DZ06evO0AI4xj | THIS IS: LEA MICHELE |
| **194527** | 39xAcb6WQSLsh42PPSuO3w | THANKSGIVING PLAYLIST 2013 |
| **194528** | 37i9dQZF1DXdLRHxHMDXUP | RUN IN THE SUN |
| **194529** | 4clluvLIJEbq9x4lliXz9j | THE NOTEBOOK SOUNDTRACK. |
| **194531** | 6BItpfhjk0sMapPsSVsrQe | TINIE TEMPAH ? MAMACITA (FEAT. WIZKID) |
| **194532** | 010UTgWTUZPygyITxlJKKA | FATHER/DAUGHTER DANCE |
| **194533** | 6Bdoy8tr4wuwt0956LWcTK | SANS SONGS |
| **194534** | 57wKvSf6MIlSHF9TymaLgb | NIMO ? LFR |
| **194535** | 1fI1nQXloXO2hz1uvcaMiv | EIER UND WURST |
| **194536** | 5aJDR7VeBq6QcYePMZMS8N | LOS AUTÉNTICOS DECADENTES |
| **194538** | 0zsas2hOjR8VwYYJQCgukr | HEAVY METAL OST (1981 FILM) |
| **194540** | 3phzy9aNi06JR84CKz66aC | PERFECT HARMONY |
| **194541** | 01kEU0zLLONJGplZDpM6QO | FEBRUARY ?? |
| **194542** | 5q3kPJ9sjhMSAgyoKlSeMG | APORED ? EVERYDAY SATUDAYL |
| **194544** | 6yXc0FnLc2Uqah7TDPOSnY | STUDIO GIBLI |
| **194545** | 68WUJufTgRx2OWFciHkib8 | INSIDE GRAHAM PARKER'S RECORD COLLECTION |
| **194547** | 5JtRcbocWdwn5hYQQ2i0Ex | YIRUMA - RIVER FLOWS IN YOU - ORIGINAL |
| **194548** | 5JfH6RWAPFykM1jjcdYP19 | TABATA CROSSFIT |
| **194549** | 55JJhmgpJIl1jsIKfv71nl | LÉO SCHWECHLEN |
| **194550** | 6H1dfMQtnFeqswIXNvmVnX | VIAJANDO CON CHESTER "EXILIADOS" |

| | id | name |
|---|---|---|
| **194551** | 4PAeP0q39YR2wSjAJt1ojF | UK GARAGE: DEEP CUTS FROM THE EARLY YEARS |
| **194552** | 427FjJ6oVgMSFGOaMccb6Z | AHORA LA DISCO LA PARTE EN DO |
| **194553** | 67QfP4ladFV0WeQGYLGbkW | ROLLING HILLS |
| **194554** | 60j4zQN6gN8Ryi5L7o1Ted | LOS TIGUERES DEL NORTE |
| **194555** | 5U2mX7AcFOLuvXsJn2Hh0x | NINA NESBITT - THE MOMENTS I'M MISSING |
| **194556** | 4JzLbL1zSJmvAFmENpu8ru | BELIEVER BY IMAGINE DRAGONS |
| **194557** | 6Wkb0oG7IvoL0fOpdzE5RZ | RACK IT UP |
| **194558** | 7B5a5I3vfjhEMrhypIO8oN | GHOST |
| **194559** | 53j5t0O1aqd4aUz6KeAcLe | BODY SAY STREAMING |

149589 rows × 2 columns

In [22]:
```
# select relevant playlists
subsetdf=playlist_mapper[playlist_mapper["name"].isin(["HOT HITS UK","MASSIVE DANCE HITS","THE INDIE LIST","NEW MUSIC FRIDAY"])]
subsetdf
```

Out[22]:

| | id | name |
|---|---|---|
| **17513** | 3DL9G1ApvJDIR4IhWIJ8AQ | NEW MUSIC FRIDAY |
| **25618** | 6FfOZSAN3N6u7v81uS7mxZ | HOT HITS UK |
| **27234** | 2mnRUIMJWqooAWlMjrlghi | NEW MUSIC FRIDAY |
| **32459** | 1EnTBEgCWiTX2YHyAzkcFn | NEW MUSIC FRIDAY |
| **35117** | 37i9dQZF1DWXJfnUiYjUKT | NEW MUSIC FRIDAY |
| **72659** | 0dLTdpGyfO0PSyYXInvRd5 | NEW MUSIC FRIDAY |
| **83239** | 6wx0wiD9V6JJ2EOh4KM3Ox | NEW MUSIC FRIDAY |
| **91903** | 35PofY2z4SqqbynOKXmdYV | NEW MUSIC FRIDAY |
| **94405** | 37i9dQZF1DX5uokaTN4FTR | MASSIVE DANCE HITS |
| **110448** | 47CSE0kHnUAZosPD3ErRwV | NEW MUSIC FRIDAY |
| **131216** | 37i9dQZF1DWVTKDs2aOkxu | THE INDIE LIST |
| **133600** | 4vGgUbD6tW2xMTABaVzCXo | NEW MUSIC FRIDAY |
| **137620** | 7lc3WCkMnfPyFbgTdVgSSh | HOT HITS UK |
| **140255** | 37i9dQZF1DX4JAvHpjipBk | NEW MUSIC FRIDAY |
| **151213** | 37i9dQZF1DWY4lFlS4Pnso | HOT HITS UK |
| **170678** | 37i9dQZF1DWT2SPAYawYcO | NEW MUSIC FRIDAY |

Then we can create our dependent variable "success". If the value of success is 0, it means that the stream can not be predicted as successful since it is not in the key playlist. If the value of success is 1, it means that the stream can be predicted as successful.

In [23]:
```
# Define Dependent Variable
successid=subsetdf["id"].unique().tolist()
successid
data1["success"]=0
data1.loc[data1["playlist_id"].isin(successid),"success"]=1
data1.head()
```

```
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/ipykernel/__main__.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/pandas/core/indexing.py:543: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
```

Out[23]:

| | log_time | artist_name | year | playlist_id | playlist_name | track_name | isrc | customer_id | birth_year | regic |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 20160510T12:15:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | 6c022a8376c10aae37abb839eb7625fe | 1968.0 | ( |
| **1** | 20160510T12:15:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | 6c022a8376c10aae37abb839eb7625fe | 1968.0 | ( |
| **2** | 20160510T14:00:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | 352292382ff3ee0cfd3b73b94ea0ff8f | 1995.0 | |
| **3** | 20160510T10:45:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | c3f2b54e76696ed491d9d8f964c97774 | 1992.0 | |
| **4** | 20160510T10:15:00 | Sturgill Simpson | 2016 | NaN | NaN | Call To Arms | USAT21600547 | 6a06a9bbe042c73e8f1a3596ec321636 | 1979.0 | |

After create the dependent variable, we would like to see that how many artists can be seem as successful. We group the original dataframe by the variable 'artist name' and apply the function into 'get_successful_artists'. A particular artist is successful if there is at least one stream turns to 1, otherwise, it turns to 0.

In [24]:
```
def get_successful_artists(data):
    df1=data['playlist_id'].unique().tolist()
    return bool(set(df1).intersection(set(successid)))

successful_artists=data1.groupby("artist_name").apply(get_successful_artists)
successful_artists=pd.DataFrame(successful_artists)
successful_artists=successful_artists.astype(int)
successful_artists.rename(columns={0:"success"},inplace=True)
successful_artists
```

Out[24]:

| | success |
|---|---|
| **artist_name** | |
| **#90s Update** | 0 |
| **17 Memphis** | 0 |
| **2D** | 0 |
| **3JS** | 0 |
| **99 Percent** | 0 |
| **A Boogie Wit Da Hoodie** | 0 |
| **A Boogie Wit da Hoodie** | 1 |
| **A R I Z O N A** | 1 |
| **AGWA** | 0 |
| **ALMA** | 0 |
| **ALP** | 0 |
| **AV AV AV** | 0 |
| **AVVAH** | 0 |
| **AXSHN** | 1 |
| **Absofacto** | 1 |
| **Adam Sample** | 0 |
| **Adan Carmona** | 0 |
| **Adia Victoria** | 0 |
| **Alcatraz** | 0 |
| **Alessio Bernabei** | 0 |
| **Alex Hoyer** | 0 |
| **Alex Roy** | 0 |
| **Alexander Brown** | 0 |
| **Alexander Cardinale** | 0 |
| **Alexander Charles** | 0 |
| **Alice** | 0 |
| **Aliose** | 0 |
| **All Tvvins** | 1 |
| **Alma** | 0 |
| **Amaro Ferreiro** | 0 |
| **...** | ... |
| **Wild Youth** | 1 |
| **Wildling** | 0 |
| **Will Joseph Cook** | 1 |
| **Willy William** | 0 |
| **Witek Muzyk Ulicy** | 0 |
| **Wudstik** | 0 |
| **Xavier Cugat y su orquesta** | 0 |
| **Xavier Dunn** | 1 |
| **YFN Lucci** | 0 |
| **YONAKA** | 0 |
| **Yasutaka Nakata** | 0 |
| **Yellow Claw** | 1 |
| **Ylric Illians** | 0 |
| **Young F** | 0 |
| **Young Spray** | 0 |
| **YoungBoy Never Broke Again** | 0 |
| **Youngboy Never Broke Again** | 0 |
| **Yvng Swag** | 0 |
| **Zac Brown** | 0 |
| **Zak & Diego** | 0 |

| | success |
|---|---|
| **artist_name** | |
| **Zak Abel** | 1 |
| **Zakopower** | 0 |
| **Zarcort** | 0 |
| **Zbigniew Kurtycz** | 0 |
| **Zion & Lennox** | 1 |
| **birthday** | 0 |
| **dvsn** | 1 |
| **flor** | 1 |
| **gnash** | 1 |
| **livetune+** | 0 |

661 rows × 1 columns

In [25]:
```
sum(successful_artists['success'])
```

Out[25]:
```
83
```

From the dataframe above, there are total 661 artists in spotify. 83 of them can be predicted as successful.

Then we would like to analyze how many artists are successful before the year 2017 using the same method.

In [26]:
```
# def get_successful_before_2017(data):
df_before2017=data1.loc[data1['year']<2017]
df_before2017=df_before2017.groupby("artist_name").apply(get_successful_artists)
df_before2017=pd.DataFrame(df_before2017)
df_before2017=df_before2017.astype(int)
df_before2017.rename(columns={0:"success"},inplace=True)
df_before2017
```

Out[26]:

| artist_name | success |
|---|---|
| 3JS | 0 |
| 99 Percent | 0 |
| A Boogie Wit Da Hoodie | 0 |
| A R I Z O N A | 0 |
| AGWA | 0 |
| ALMA | 0 |
| AV AV AV | 0 |
| AVVAH | 0 |
| Adan Carmona | 0 |
| Adia Victoria | 0 |
| Alessio Bernabei | 0 |
| Alex Hoyer | 0 |
| Alex Roy | 0 |
| Alexander Brown | 0 |
| Alexander Cardinale | 0 |
| Alice | 0 |
| Aliose | 0 |
| All Tvvins | 1 |
| Alma | 0 |
| Amaro Ferreiro | 0 |
| Amir | 1 |
| Andy Bros | 0 |
| Angel | 0 |
| Angelica Garcia | 0 |
| Anna Puu | 0 |
| Annabel Jones | 0 |
| Anne-Marie | 1 |
| Anteros | 0 |
| Arco | 0 |
| Arsen | 0 |
| ... | ... |
| Ugly God | 0 |
| Urbano Prodigy | 0 |
| Usher | 0 |
| Utha Likumahuwa | 0 |
| VAN HOLZEN | 0 |
| VANT | 0 |
| VIMIC | 0 |
| Venus II | 0 |
| Vice | 1 |
| Vinyl on HBO | 1 |
| Virgul | 0 |
| WEDNESDAY CAMPANELLA | 0 |
| WSTRN | 1 |
| Waylon | 0 |
| We Are Messengers | 0 |
| Whethan | 0 |
| Will Joseph Cook | 1 |
| Willy William | 0 |
| Witek Muzyk Ulicy | 0 |
| Wudstik | 0 |

| artist_name | success |
|---|---|
| Xavier Dunn | 1 |
| Yellow Claw | 0 |
| Young Spray | 0 |
| Zac Brown | 0 |
| Zak & Diego | 0 |
| Zak Abel | 0 |
| Zion & Lennox | 1 |
| dvsn | 0 |
| gnash | 1 |
| livetune+ | 0 |

366 rows × 1 columns

In [27]:
```
sum(df_before2017['success'])
```

Out[27]: 28

It can be noticed that 28 of 366 artists were successful before the year 2017.

Now that we have created our dependent variable – whether an artist is successful or not, we can look at generating a set of features, based on the columns within our dataset, that we think might best explain the reasons for this success.

**FEATURE ENGINEERING**

There are a large number of factors that could have an impact on the success of an artist, such as the influence of a playlist, or the popularity of an artist in a certain geographical region.

To build a predictive model for this problem, we first need to turn these (largely qualitative) factors into measurable quantities. Characteristics like 'influence' and 'popularity' need to be quantified and standardized for all artists, to allow for a fair comparison.

The accurateness of these numerical estimates will be the fundamental driver of success for any model we build. There are many approaches one might take to generate features. Based on the data columns available to us, a sensible approach is to divide our feature set into three groups:

1. Artist Features
2. Playlist Features
3. User-base features

## Artist features

- Stream count
- Total Number of users
- Passion Score

The metric passion score is a metric suggested to us by Warner business analysts.

It is defined as the number of stream divided by the total number of users.

Warner analysts believe that repeated listens by a user is a far more indicative future success that simply total number of listens or total unique users. By including this in your model, we can evaluate whether this metric in fact might be of any significance.

**ACTION: Artist features**

Write useful functions to create these new features.

Firstly we calculate the stream count per artist for all the 661 artists into a dataframe 'stream_count'.

In [28]:
```
# Stream count per artist

stream_count=data1.groupby("artist_name")['artist_name'].count()
stream_count=pd.DataFrame(stream_count)
stream_count.rename(columns={"artist_name":"count"},inplace=True)
stream_count
#data1=pd.merge(data1,stream_count, on="artist_name")
#data1.head()
```

Out[28]:

| artist_name | count |
|---|---|
| #90s Update | 16 |
| 17 Memphis | 12 |
| 2D | 1 |
| 3JS | 5 |
| 99 Percent | 1291 |
| A Boogie Wit Da Hoodie | 9904 |
| A Boogie Wit da Hoodie | 13264 |
| A R I Z O N A | 68830 |
| AGWA | 3 |
| ALMA | 8 |
| ALP | 4 |
| AV AV AV | 57 |
| AVVAH | 20 |
| AXSHN | 112 |
| Absofacto | 138 |
| Adam Sample | 42 |
| Adan Carmona | 14 |
| Adia Victoria | 671 |
| Alcatraz | 7 |
| Alessio Bernabei | 191 |
| Alex Hoyer | 27 |
| Alex Roy | 3 |
| Alexander Brown | 147 |
| Alexander Cardinale | 584 |
| Alexander Charles | 53 |
| Alice | 45 |
| Aliose | 21 |
| All Tvvins | 10446 |
| Alma | 994 |
| Amaro Ferreiro | 8 |
| ... | ... |
| Wild Youth | 284 |
| Wildling | 17 |
| Will Joseph Cook | 15322 |
| Willy William | 1916 |
| Witek Muzyk Ulicy | 17 |
| Wudstik | 23 |
| Xavier Cugat y su orquesta | 1 |
| Xavier Dunn | 5824 |
| YFN Lucci | 1487 |
| YONAKA | 340 |
| Yasutaka Nakata | 90 |
| Yellow Claw | 6734 |
| Ylric Illians | 3 |
| Young F | 1 |
| Young Spray | 667 |
| YoungBoy Never Broke Again | 783 |
| Youngboy Never Broke Again | 104 |
| Yvng Swag | 238 |
| Zac Brown | 59 |
| Zak & Diego | 12 |

| artist_name | count |
|---|---|
| Zak Abel | 26966 |
| Zakopower | 1 |
| Zarcort | 25 |
| Zbigniew Kurtycz | 2 |
| Zion & Lennox | 10721 |
| birthday | 20 |
| dvsn | 25168 |
| flor | 109 |
| gnash | 165683 |
| livetune+ | 7 |

661 rows × 1 columns

Then we calculate the number of users per artist using 'customer_id' into a dataframe 'num_users'.

In [29]:
```
# Number of users per artist
num_users=data1.groupby("artist_name")['customer_id'].nunique()
num_users=pd.DataFrame(num_users)

#data1=pd.merge(data1,num_users,on="artist_name")
#data1.head()
num_users
```

Out[29]:

| artist_name | customer_id |
|---|---|
| #90s Update | 15 |
| 17 Memphis | 12 |
| 2D | 1 |
| 3JS | 4 |
| 99 Percent | 1189 |
| A Boogie Wit Da Hoodie | 7713 |
| A Boogie Wit da Hoodie | 11154 |
| A R I Z O N A | 58987 |
| AGWA | 3 |
| ALMA | 8 |
| ALP | 4 |
| AV AV AV | 54 |
| AVVAH | 20 |
| AXSHN | 109 |
| Absofacto | 136 |
| Adam Sample | 42 |
| Adan Carmona | 12 |
| Adia Victoria | 607 |
| Alcatraz | 7 |
| Alessio Bernabei | 156 |
| Alex Hoyer | 27 |
| Alex Roy | 3 |
| Alexander Brown | 141 |
| Alexander Cardinale | 552 |
| Alexander Charles | 51 |
| Alice | 39 |
| Aliose | 18 |
| All Tvvins | 9156 |
| Alma | 832 |
| Amaro Ferreiro | 4 |
| ... | ... |
| Wild Youth | 284 |
| Wildling | 17 |
| Will Joseph Cook | 12585 |
| Willy William | 1607 |
| Witek Muzyk Ulicy | 11 |
| Wudstik | 23 |
| Xavier Cugat y su orquesta | 1 |
| Xavier Dunn | 5406 |
| YFN Lucci | 1395 |
| YONAKA | 336 |
| Yasutaka Nakata | 77 |
| Yellow Claw | 5948 |
| Ylric Illians | 1 |
| Young F | 1 |
| Young Spray | 634 |
| YoungBoy Never Broke Again | 621 |
| Youngboy Never Broke Again | 75 |
| Yvng Swag | 226 |
| Zac Brown | 59 |
| Zak & Diego | 10 |

| | customer_id |
|---|---|
| **artist_name** | |
| **Zak Abel** | 23417 |
| **Zakopower** | 1 |
| **Zarcort** | 18 |
| **Zbigniew Kurtycz** | 2 |
| **Zion & Lennox** | 9303 |
| **birthday** | 20 |
| **dvsn** | 18712 |
| **flor** | 108 |
| **gnash** | 146108 |
| **livetune+** | 6 |

661 rows × 1 columns

And we calculate the passion score for each artist into a dataframe 'passionscore'. The number of stream is divided by the total number of users by using 'count' and 'customer_id' in the last two dataframe.

In [30]:
```
# Passion Score
passionscore=stream_count['count']/num_users['customer_id']
passionscore=pd.DataFrame(passionscore)
passionscore
passionscore.rename(columns={0:'passionscore'},inplace=True)
passionscore
#data1['passion_score']=data1['count']/data1['customer_id_y']
#data1.head()
```

Out[30]:

| | passionscore |
|---|---|
| **artist_name** | |
| **#90s Update** | 1.066667 |
| **17 Memphis** | 1.000000 |
| **2D** | 1.000000 |
| **3JS** | 1.250000 |
| **99 Percent** | 1.085786 |
| **A Boogie Wit Da Hoodie** | 1.284066 |
| **A Boogie Wit da Hoodie** | 1.189170 |
| **A R I Z O N A** | 1.166867 |
| **AGWA** | 1.000000 |
| **ALMA** | 1.000000 |
| **ALP** | 1.000000 |
| **AV AV AV** | 1.055556 |
| **AVVAH** | 1.000000 |
| **AXSHN** | 1.027523 |
| **Absofacto** | 1.014706 |
| **Adam Sample** | 1.000000 |
| **Adan Carmona** | 1.166667 |
| **Adia Victoria** | 1.105437 |
| **Alcatraz** | 1.000000 |
| **Alessio Bernabei** | 1.224359 |
| **Alex Hoyer** | 1.000000 |
| **Alex Roy** | 1.000000 |
| **Alexander Brown** | 1.042553 |
| **Alexander Cardinale** | 1.057971 |
| **Alexander Charles** | 1.039216 |
| **Alice** | 1.153846 |
| **Aliose** | 1.166667 |
| **All Tvvins** | 1.140891 |
| **Alma** | 1.194712 |
| **Amaro Ferreiro** | 2.000000 |
| **...** | ... |
| **Wild Youth** | 1.000000 |
| **Wildling** | 1.000000 |
| **Will Joseph Cook** | 1.217481 |
| **Willy William** | 1.192284 |
| **Witek Muzyk Ulicy** | 1.545455 |
| **Wudstik** | 1.000000 |
| **Xavier Cugat y su orquesta** | 1.000000 |
| **Xavier Dunn** | 1.077321 |
| **YFN Lucci** | 1.065950 |
| **YONAKA** | 1.011905 |
| **Yasutaka Nakata** | 1.168831 |
| **Yellow Claw** | 1.132145 |
| **Ylric Illians** | 3.000000 |
| **Young F** | 1.000000 |
| **Young Spray** | 1.052050 |
| **YoungBoy Never Broke Again** | 1.260870 |
| **Youngboy Never Broke Again** | 1.386667 |
| **Yvng Swag** | 1.053097 |
| **Zac Brown** | 1.000000 |
| **Zak & Diego** | 1.200000 |

| artist_name | passionscore |
|---|---|
| **Zak Abel** | 1.151557 |
| **Zakopower** | 1.000000 |
| **Zarcort** | 1.388889 |
| **Zbigniew Kurtycz** | 1.000000 |
| **Zion & Lennox** | 1.152424 |
| **birthday** | 1.000000 |
| **dvsn** | 1.345019 |
| **flor** | 1.009259 |
| **gnash** | 1.133976 |
| **livetune+** | 1.166667 |

661 rows × 1 columns

At last, we create a new dataframe contains 'success' and the three variables regarding to artist features for all artists which will be used later

In [31]:
```
# Artist Features DataFrame

df_artist_features=successful_artists
df_artist_features['count']=stream_count['count']
df_artist_features['number_users']=num_users['customer_id']
df_artist_features['passion_score']=passionscore['passionscore']

df_artist_features
```

Out[31]:

| | success | count | number_users | passion_score |
|---|---|---|---|---|
| **artist_name** | | | | |
| **#90s Update** | 0 | 16 | 15 | 1.066667 |
| **17 Memphis** | 0 | 12 | 12 | 1.000000 |
| **2D** | 0 | 1 | 1 | 1.000000 |
| **3JS** | 0 | 5 | 4 | 1.250000 |
| **99 Percent** | 0 | 1291 | 1189 | 1.085786 |
| **A Boogie Wit Da Hoodie** | 0 | 9904 | 7713 | 1.284066 |
| **A Boogie Wit da Hoodie** | 1 | 13264 | 11154 | 1.189170 |
| **A R I Z O N A** | 1 | 68830 | 58987 | 1.166867 |
| **AGWA** | 0 | 3 | 3 | 1.000000 |
| **ALMA** | 0 | 8 | 8 | 1.000000 |
| **ALP** | 0 | 4 | 4 | 1.000000 |
| **AV AV AV** | 0 | 57 | 54 | 1.055556 |
| **AVVAH** | 0 | 20 | 20 | 1.000000 |
| **AXSHN** | 1 | 112 | 109 | 1.027523 |
| **Absofacto** | 1 | 138 | 136 | 1.014706 |
| **Adam Sample** | 0 | 42 | 42 | 1.000000 |
| **Adan Carmona** | 0 | 14 | 12 | 1.166667 |
| **Adia Victoria** | 0 | 671 | 607 | 1.105437 |
| **Alcatraz** | 0 | 7 | 7 | 1.000000 |
| **Alessio Bernabei** | 0 | 191 | 156 | 1.224359 |
| **Alex Hoyer** | 0 | 27 | 27 | 1.000000 |
| **Alex Roy** | 0 | 3 | 3 | 1.000000 |
| **Alexander Brown** | 0 | 147 | 141 | 1.042553 |
| **Alexander Cardinale** | 0 | 584 | 552 | 1.057971 |
| **Alexander Charles** | 0 | 53 | 51 | 1.039216 |
| **Alice** | 0 | 45 | 39 | 1.153846 |
| **Aliose** | 0 | 21 | 18 | 1.166667 |
| **All Tvvins** | 1 | 10446 | 9156 | 1.140891 |
| **Alma** | 0 | 994 | 832 | 1.194712 |
| **Amaro Ferreiro** | 0 | 8 | 4 | 2.000000 |
| **...** | ... | ... | ... | ... |
| **Wild Youth** | 1 | 284 | 284 | 1.000000 |
| **Wildling** | 0 | 17 | 17 | 1.000000 |
| **Will Joseph Cook** | 1 | 15322 | 12585 | 1.217481 |
| **Willy William** | 0 | 1916 | 1607 | 1.192284 |
| **Witek Muzyk Ulicy** | 0 | 17 | 11 | 1.545455 |
| **Wudstik** | 0 | 23 | 23 | 1.000000 |
| **Xavier Cugat y su orquesta** | 0 | 1 | 1 | 1.000000 |
| **Xavier Dunn** | 1 | 5824 | 5406 | 1.077321 |
| **YFN Lucci** | 0 | 1487 | 1395 | 1.065950 |
| **YONAKA** | 0 | 340 | 336 | 1.011905 |
| **Yasutaka Nakata** | 0 | 90 | 77 | 1.168831 |
| **Yellow Claw** | 1 | 6734 | 5948 | 1.132145 |
| **Ylric Illians** | 0 | 3 | 1 | 3.000000 |
| **Young F** | 0 | 1 | 1 | 1.000000 |
| **Young Spray** | 0 | 667 | 634 | 1.052050 |
| **YoungBoy Never Broke Again** | 0 | 783 | 621 | 1.260870 |
| **Youngboy Never Broke Again** | 0 | 104 | 75 | 1.386667 |
| **Yvng Swag** | 0 | 238 | 226 | 1.053097 |
| **Zac Brown** | 0 | 59 | 59 | 1.000000 |
| **Zak & Diego** | 0 | 12 | 10 | 1.200000 |

| | success | count | number_users | passion_score |
|---|---|---|---|---|
| **artist_name** | | | | |
| **Zak Abel** | 1 | 26966 | 23417 | 1.151557 |
| **Zakopower** | 0 | 1 | 1 | 1.000000 |
| **Zarcort** | 0 | 25 | 18 | 1.388889 |
| **Zbigniew Kurtycz** | 0 | 2 | 2 | 1.000000 |
| **Zion & Lennox** | 1 | 10721 | 9303 | 1.152424 |
| **birthday** | 0 | 20 | 20 | 1.000000 |
| **dvsn** | 1 | 25168 | 18712 | 1.345019 |
| **flor** | 1 | 109 | 108 | 1.009259 |
| **gnash** | 1 | 165683 | 146108 | 1.133976 |
| **livetune+** | 0 | 7 | 6 | 1.166667 |

661 rows × 4 columns

In [32]:
```
df_artist_features.success.sum()
```

Out[32]: 83

## Playlist Features

Understanding an artist's growth as a function of his/her movement across different playlists is potentially key to understanding how to identify and breakout new artists on Spotify.

In turn, this could help us identify the most influential playlists and the reasons for their influence.

One way to model the effect of playlists on an artist's performance has been to include them as categorical features in our model, to note if there are any particular playlists or combinations of playlists that are responsible for propelling an artist to future success:

### ***Artist Feature 1 + Artist Feature 2 …. + Artist Feature N = Probability of Success***

**Success (1) = Artist Features on Key Playlist Failure (0) = Artist Not Featured on Key Playlist**

Where,

**⇒Artist Feature N = Prior Playlist 1 + Prior Playlist 2 +…Prior Playlist N**

Given that we have over 19,000 playlists in our dataset or 600 artists, using the playlists each artist has featured on, as categorical variables would lead to too many features and a very large, sparse matrix.

Instead, we need to think of ways to summarize the impact of these playlists. One way to do this would be to consider the top 20 playlists each artist has featured on.

Even better would be to come up with one metric that captures the net effect of all top 20 prior playlists, for each artist, rather including using all 20 playlists for each artists as binary variables. The intuition here is that if this metric as a whole has an influence on the performance of an artist, it would suggest that rather than the individual playlists themselves, it is a combination of their generalized features that affects the future performance of an artist.

Accordingly, different combinations of playlists could equate to having the same impact on an artist, thereby allowing us to identify undervalued playlists.

Some of the features such a metric could use is the number of unique users or 'reach', number of stream counts, and the passion score of each playlist

- Prior Playlist Stream Counts
- Prior Playlist Unique Users (Reach)
- Prior Playlist Passion Score

There are several other such features that you could generate to better capture the general characteristics of playlists, such as the average lift in stream counts and users they generate for artists that have featured on them.

The code to calculate these metrics is provided below:

**ACTION: Playlist features**

Write useful functions to create new playlist features, like those listed in the cell above.

Are there other sensible ones you could suggest, work in your group to think about what other features might be useful and whether you can calculate them with the data you have

We decide to create a dataframe contains streamcount, number of users and the passion score of each playlist and then calculate the weighted average of top 20 playlists' passion score for each artist. Thus, we work out the streamcount of each playlist and create the dataframe 'df_playlist' firstly.

In [33]:
```
# you could divide up the work in the group by getting different people to calculate different features

#def playlist_avg_stream_counts(data):

playlist_count=data1.groupby("playlist_name")['playlist_name'].count()
playlist_count=pd.DataFrame(playlist_count)
df_playlist=playlist_count
df_playlist.rename(columns={"playlist_name":"playlist_streamcount"},inplace=True)

df_playlist
```

Out[33]:

| | playlist_streamcount |
|---|---|
| **playlist_name** | |
| **SEPTEMBER 2016 TOP HITS** | 24 |
| **2015 Hits** | 2 |
| **2016 Rap ?** | 5 |
| **?Space ?** | 1 |
| **Avicii - Tiësto - Calvin Harris - Alesso - Swedish house mafia - Zedd - Nause - David Guetta - Har** | 1 |
| **Fall 2015 Hip Hop / R&B playlist** | 5 |
| **Hollister Vibe 2016** | 8 |
| **I took a pill in biza** | 1 |
| **Me & My Girls** | 2 |
| **Music Hits 2016 - Best Songs Playlist** | 2 |
| **Now 97 (Nick's Pediction)** | 17 |
| **PRESIDENT DAVO- I DONT WANNA BE A PLAYA** | 1 |
| **Perreos - Reggeaton - Pachangeo 2015/ Daddy Yankee Ft Omega El Fuerte - Estrellita de Madrugada** | 1 |
| **Throw back** | 1 |
| **iSpy** | 2 |
| **now96** | 2 |
| **top hit songs 2016** | 1 |
| **#1 Most Wanted Hits** | 10 |
| **#AbdiTVBdayMix** | 2 |
| **#AlDub: Happily Ever After** | 3 |
| **#BESTE INDIE** | 5 |
| **#Beste Urban** | 10 |
| **#CantoPop Hong Kong ?? ??? ????** | 2 |
| **#Covers** | 11 |
| **#Dance** | 14 |
| **#FlashbackFriday** | 1 |
| **#GIRLPOWER** | 3 |
| **#HDYNATION RADIO** | 1 |
| **#HairFlipPlaylist** | 1 |
| **#Hideout2017 First Artists Playlist** | 2 |
| **...** | ... |
| **you are beautiful??????** | 1 |
| **yup** | 4 |
| **zion y lenox motivan2** | 1 |
| **{ SEX SONGS }** | 1 |
| **{musica pra dançar pelado}** | 1 |
| **\|Solo Dance - Martin Jensen\|Setting Fire - The Chainsmokers\|Castle on the Hill - Ed Sheeran\|Shape of** | 1 |
| **~* 2016 Favs *~** | 2 |
| **~*cool finds*~** | 1 |
| **¡ADRENALINA!** | 2 |
| **¡Baila Sin Parar!** | 1 |
| **¡Hola 2016!** | 70 |
| **¡Por Fin Vacaciones!** | 1 |
| **¡Por Fin Viernes!** | 4 |
| **© Inrocks - Top 2016** | 1 |
| **® Summerfeeling 2017 [Kygo]** | 2 |
| **Årets bidrag & artister** | 20 |
| **ÉXITOS 2017 ABRIL - APRIL HITS - Shakira Me Enamoré Shakira - DNCE Kissing Strangers DNCE Europa F** | 116 |
| **Éxitos Argentina** | 7 |
| **Éxitos Chile** | 1 |
| **Éxitos Colombia** | 17 |

| playlist_name | playlist_streamcount |
|---|---|
| **Éxitos España** | 46 |
| **Éxitos MX** | 5 |
| **Éxitos México** | 13 |
| **Éxitos Party 2016** | 1 |
| **Éxitos Pop (los remixes)** | 2 |
| **Éxitos de Hoy - Chile** | 25 |
| **Éxitos en acústico** | 1 |
| **Ö3-Hörerplaylist** | 1 |
| **Örnis Playlist** | 1 |
| **écouter** | 2 |

7102 rows × 1 columns

We can see that there are 7102 playlists in total. Then we work out the number of users of each playlist and add it to the dataframe 'df_playlist'.

In [34]:
```
#def playlist_avg_number_of_users(data):

playlist_users=data1.groupby("playlist_name")['customer_id'].nunique()
playlist_users=pd.DataFrame(playlist_users)
playlist_users.rename(columns={"customer_id":"playlist_users"},inplace=True)
playlist_users
df_playlist['playlist_users']=playlist_users['playlist_users']
df_playlist
```

Out[34]:

| playlist_name | playlist_streamcount | playlist_users |
|---|---|---|
| SEPTEMBER 2016 TOP HITS | 24 | 14 |
| 2015 Hits | 2 | 2 |
| 2016 Rap ? | 5 | 5 |
| ?Space ? | 1 | 1 |
| Avicii - Tiësto - Calvin Harris - Alesso - Swedish house mafia - Zedd - Nause - David Guetta - Har | 1 | 1 |
| Fall 2015 Hip Hop / R&B playlist | 5 | 4 |
| Hollister Vibe 2016 | 8 | 8 |
| I took a pill in biza | 1 | 1 |
| Me & My Girls | 2 | 2 |
| Music Hits 2016 - Best Songs Playlist | 2 | 2 |
| Now 97 (Nick's Pediction) | 17 | 17 |
| PRESIDENT DAVO- I DONT WANNA BE A PLAYA | 1 | 1 |
| Perreos - Reggeaton - Pachangeo 2015/ Daddy Yankee Ft Omega El Fuerte - Estrellita de Madrugada | 1 | 1 |
| Throw back | 1 | 1 |
| iSpy | 2 | 2 |
| now96 | 2 | 2 |
| top hit songs 2016 | 1 | 1 |
| #1 Most Wanted Hits | 10 | 10 |
| #AbdiTVBdayMix | 2 | 2 |
| #AlDub: Happily Ever After | 3 | 3 |
| #BESTE INDIE | 5 | 3 |
| #Beste Urban | 10 | 9 |
| #CantoPop Hong Kong ?? ??? ???? | 2 | 2 |
| #Covers | 11 | 11 |
| #Dance | 14 | 13 |
| #FlashbackFriday | 1 | 1 |
| #GIRLPOWER | 3 | 3 |
| #HDYNATION RADIO | 1 | 1 |
| #HairFlipPlaylist | 1 | 1 |
| #Hideout2017 First Artists Playlist | 2 | 2 |
| ... | ... | ... |
| you are beautiful?????? | 1 | 1 |
| yup | 4 | 2 |
| zion y lenox motivan2 | 1 | 1 |
| { SEX SONGS } | 1 | 1 |
| {musica pra dançar pelado} | 1 | 1 |
| \|Solo Dance - Martin Jensen\|Setting Fire - The Chainsmokers\|Castle on the Hill - Ed Sheeran\|Shape of | 1 | 1 |
| ~* 2016 Favs *~ | 2 | 2 |
| ~*cool finds*~ | 1 | 1 |
| ¡ADRENALINA! | 2 | 2 |
| ¡Baila Sin Parar! | 1 | 1 |
| ¡Hola 2016! | 70 | 68 |
| ¡Por Fin Vacaciones! | 1 | 1 |
| ¡Por Fin Viernes! | 4 | 4 |
| © Inrocks - Top 2016 | 1 | 1 |
| ® Summerfeeling 2017 [Kygo] | 2 | 2 |
| Årets bidrag & artister | 20 | 15 |
| ÉXITOS 2017 ABRIL - APRIL HITS - Shakira Me Enamoré Shakira - DNCE Kissing Strangers DNCE Europa F | 116 | 109 |
| Éxitos Argentina | 7 | 6 |
| Éxitos Chile | 1 | 1 |
| Éxitos Colombia | 17 | 14 |

| | playlist_streamcount | playlist_users |
|---|---|---|
| **playlist_name** | | |
| **Éxitos España** | 46 | 43 |
| **Éxitos MX** | 5 | 5 |
| **Éxitos México** | 13 | 13 |
| **Éxitos Party 2016** | 1 | 1 |
| **Éxitos Pop (los remixes)** | 2 | 2 |
| **Éxitos de Hoy - Chile** | 25 | 14 |
| **Éxitos en acústico** | 1 | 1 |
| **Ö3-Hörerplaylist** | 1 | 1 |
| **Örnis Playlist** | 1 | 1 |
| **écouter** | 2 | 2 |

7102 rows × 2 columns

Similarly, we calculate the passion score of each playlist by dividing streamcount by number of users and add it to the dataframe 'df_playlist'.

In [35]:
```
#def playlist_avg_passion_score(data):
playlist_ps=playlist_count['playlist_streamcount']/playlist_users['playlist_users']
playlist_ps=pd.DataFrame(playlist_ps)
playlist_ps.rename(columns={0:'playlist_ps'},inplace=True)

df_playlist['playlist_ps']=playlist_ps['playlist_ps']
df_playlist
```

Out[35]:

| playlist_name | playlist_streamcount | playlist_users | playlist_ps |
|---|---|---|---|
| **SEPTEMBER 2016 TOP HITS** | 24 | 14 | 1.714286 |
| **2015 Hits** | 2 | 2 | 1.000000 |
| **2016 Rap ?** | 5 | 5 | 1.000000 |
| **?Space ?** | 1 | 1 | 1.000000 |
| **Avicii - Tiësto - Calvin Harris - Alesso - Swedish house mafia - Zedd - Nause - David Guetta - Har** | 1 | 1 | 1.000000 |
| **Fall 2015 Hip Hop / R&B playlist** | 5 | 4 | 1.250000 |
| **Hollister Vibe 2016** | 8 | 8 | 1.000000 |
| **I took a pill in biza** | 1 | 1 | 1.000000 |
| **Me & My Girls** | 2 | 2 | 1.000000 |
| **Music Hits 2016 - Best Songs Playlist** | 2 | 2 | 1.000000 |
| **Now 97 (Nick's Pediction)** | 17 | 17 | 1.000000 |
| **PRESIDENT DAVO- I DONT WANNA BE A PLAYA** | 1 | 1 | 1.000000 |
| **Perreos - Reggeaton - Pachangeo 2015/ Daddy Yankee Ft Omega El Fuerte - Estrellita de Madrugada** | 1 | 1 | 1.000000 |
| **Throw back** | 1 | 1 | 1.000000 |
| **iSpy** | 2 | 2 | 1.000000 |
| **now96** | 2 | 2 | 1.000000 |
| **top hit songs 2016** | 1 | 1 | 1.000000 |
| **#1 Most Wanted Hits** | 10 | 10 | 1.000000 |
| **#AbdiTVBdayMix** | 2 | 2 | 1.000000 |
| **#AlDub: Happily Ever After** | 3 | 3 | 1.000000 |
| **#BESTE INDIE** | 5 | 3 | 1.666667 |
| **#Beste Urban** | 10 | 9 | 1.111111 |
| **#CantoPop Hong Kong ?? ??? ????** | 2 | 2 | 1.000000 |
| **#Covers** | 11 | 11 | 1.000000 |
| **#Dance** | 14 | 13 | 1.076923 |
| **#FlashbackFriday** | 1 | 1 | 1.000000 |
| **#GIRLPOWER** | 3 | 3 | 1.000000 |
| **#HDYNATION RADIO** | 1 | 1 | 1.000000 |
| **#HairFlipPlaylist** | 1 | 1 | 1.000000 |
| **#Hideout2017 First Artists Playlist** | 2 | 2 | 1.000000 |
| **...** | ... | ... | ... |
| **you are beautiful??????** | 1 | 1 | 1.000000 |
| **yup** | 4 | 2 | 2.000000 |
| **zion y lenox motivan2** | 1 | 1 | 1.000000 |
| **{ SEX SONGS }** | 1 | 1 | 1.000000 |
| **{musica pra dançar pelado}** | 1 | 1 | 1.000000 |
| **\|Solo Dance - Martin Jensen\|Setting Fire - The Chainsmokers\|Castle on the Hill - Ed Sheeran\|Shape of** | 1 | 1 | 1.000000 |
| **~* 2016 Favs *~** | 2 | 2 | 1.000000 |
| **~*cool finds*~** | 1 | 1 | 1.000000 |
| **¡ADRENALINA!** | 2 | 2 | 1.000000 |
| **¡Baila Sin Parar!** | 1 | 1 | 1.000000 |
| **¡Hola 2016!** | 70 | 68 | 1.029412 |
| **¡Por Fin Vacaciones!** | 1 | 1 | 1.000000 |
| **¡Por Fin Viernes!** | 4 | 4 | 1.000000 |
| **© Inrocks - Top 2016** | 1 | 1 | 1.000000 |
| **® Summerfeeling 2017 [Kygo]** | 2 | 2 | 1.000000 |
| **Årets bidrag & artister** | 20 | 15 | 1.333333 |
| **ÉXITOS 2017 ABRIL - APRIL HITS - Shakira Me Enamoré Shakira - DNCE Kissing Strangers DNCE Europa F** | 116 | 109 | 1.064220 |
| **Éxitos Argentina** | 7 | 6 | 1.166667 |
| **Éxitos Chile** | 1 | 1 | 1.000000 |

| playlist_name | playlist_streamcount | playlist_users | playlist_ps |
|---|---|---|---|
| **Éxitos Colombia** | 17 | 14 | 1.214286 |
| **Éxitos España** | 46 | 43 | 1.069767 |
| **Éxitos MX** | 5 | 5 | 1.000000 |
| **Éxitos México** | 13 | 13 | 1.000000 |
| **Éxitos Party 2016** | 1 | 1 | 1.000000 |
| **Éxitos Pop (los remixes)** | 2 | 2 | 1.000000 |
| **Éxitos de Hoy - Chile** | 25 | 14 | 1.785714 |
| **Éxitos en acústico** | 1 | 1 | 1.000000 |
| **Ö3-Hörerplaylist** | 1 | 1 | 1.000000 |
| **Örnis Playlist** | 1 | 1 | 1.000000 |
| **écouter** | 2 | 2 | 1.000000 |

7102 rows × 3 columns

For calculating the weighted average of top 20 playlists' passion scores for each artist, we need to find the top 20 stream count of playlists for each artists firstly.

In [36]:
```
playlist_count=data1.groupby(['artist_name','playlist_name']).size()
playlist_count=playlist_count.groupby(level=0,group_keys=False).nlargest(20)
df_ap1=pd.DataFrame(playlist_count).reset_index()
df_ap1.rename(columns={0:'count'},inplace=True)
df_ap1
```

Out[36]:

| | artist_name | playlist_name | count |
|---|---|---|---|
| **0** | #90s Update | After Work House | 3 |
| **1** | #90s Update | ENERGY - HIT MUSIC ONLY! | 1 |
| **2** | 17 Memphis | Wild Country | 6 |
| **3** | 99 Percent | Musical.ly songs | 8 |
| **4** | 99 Percent | Party Bangers! | 8 |
| **5** | 99 Percent | jacob sartorius' musical.ly's. | 5 |
| **6** | 99 Percent | Topsify US Top 40 | 5 |
| **7** | 99 Percent | Tomorrow's Hits | 4 |
| **8** | 99 Percent | ROAD TRIP | 3 |
| **9** | 99 Percent | Top Songs of the Month - Sworkit Workout Playl... | 3 |
| **10** | 99 Percent | CLEAN MUSIC ONLY | 2 |
| **11** | 99 Percent | 99 Percent ? iTwerk (She Twerk)i | 1 |
| **12** | 99 Percent | iTunes UK TOP 40 - Continously updated | 1 |
| **13** | 99 Percent | Topsify US Dance Top 50 | 1 |
| **14** | 99 Percent | Tastemaker | 1 |
| **15** | 99 Percent | The Weeknd - Starboy | 1 |
| **16** | 99 Percent | BassBoosted??? | 1 |
| **17** | 99 Percent | NEW LOVE | 1 |
| **18** | 99 Percent | Music.ly songs | 1 |
| **19** | 99 Percent | Hits 2000-2016 | 1 |
| **20** | 99 Percent | slutty pre game | 1 |
| **21** | A Boogie Wit Da Hoodie | Rap Caviar | 1982 |
| **22** | A Boogie Wit Da Hoodie | Hip Hop Party | 213 |
| **23** | A Boogie Wit Da Hoodie | Most DefiLitly | 145 |
| **24** | A Boogie Wit Da Hoodie | RapCaviar | 143 |
| **25** | A Boogie Wit Da Hoodie | Hip Hop Club Bangers: 'LUDACRIS SPECIAL' Week | 74 |
| **26** | A Boogie Wit Da Hoodie | The 411 | 25 |
| **27** | A Boogie Wit Da Hoodie | Roots Picnic NYC | 22 |
| **28** | A Boogie Wit Da Hoodie | HipHop Hot 50 | 18 |
| **29** | A Boogie Wit Da Hoodie | Highbridge The Label | 12 |
| **...** | ... | ... | ... |
| **4175** | flor | Chill Vibes | 3 |
| **4176** | flor | Study Break | 2 |
| **4177** | flor | Electro Rock Top Tracks | 1 |
| **4178** | flor | Indie Rock Top Tracks | 1 |
| **4179** | flor | It's ALT Good! | 1 |
| **4180** | flor | License To Chill | 1 |
| **4181** | flor | New Music Tuesday | 1 |
| **4182** | flor | SAINTE - Winter Rotation | 1 |
| **4183** | flor | Tomorrow's Hits | 1 |
| **4184** | gnash | Hot Hits UK | 8947 |
| **4185** | gnash | Today's Top Hits | 8481 |
| **4186** | gnash | Heartbreaker | 2515 |
| **4187** | gnash | Topsify UK Top 40 | 2426 |
| **4188** | gnash | Revision Ballads | 2391 |
| **4189** | gnash | Mellow Pop | 1979 |
| **4190** | gnash | Top Tracks in The United Kingdom | 1845 |
| **4191** | gnash | Chilled Pop Hits | 1144 |
| **4192** | gnash | The Pop List | 1038 |
| **4193** | gnash | New Music Monday UK | 989 |
| **4194** | gnash | NEW CHILLOUT | 850 |
| **4195** | gnash | Top 100 tracks currently on Spotify | 684 |

| | artist_name | playlist_name | count |
|---|---|---|---|
| **4196** | gnash | Easy | 678 |
| **4197** | gnash | Chill Hits | 674 |
| **4198** | gnash | Official UK Top 40 Charts \| 7th June 2015 \| Up... | 528 |
| **4199** | gnash | Top Tracks of 2016 UK | 417 |
| **4200** | gnash | Acoustic & Chill | 388 |
| **4201** | gnash | Broken Heart | 369 |
| **4202** | gnash | Pop Right Now! | 369 |
| **4203** | gnash | UK Top 40 2016 | 367 |
| **4204** | livetune+ | J-Track Makunouchi | 1 |

4205 rows × 3 columns

Then we merge it with the relevant playlist passion score.

In [37]:
```
df_apmerge=df_apl.merge(df_playlist,on='playlist_name',how='inner')
df_apmerge=df_apmerge.sort_values(by=['artist_name','count'],ascending=[1,0])

df_apmerge.drop(columns=['playlist_streamcount','playlist_users'],inplace=True)
df_apmerge
```

Out[37]:

| | artist_name | playlist_name | count | playlist_ps |
|---|---|---|---|---|
| **0** | #90s Update | After Work House | 3 | 1.116279 |
| **4** | #90s Update | ENERGY - HIT MUSIC ONLY! | 1 | 1.387097 |
| **12** | 17 Memphis | Wild Country | 6 | 1.062500 |
| **18** | 99 Percent | Musical.ly songs | 8 | 1.000000 |
| **20** | 99 Percent | Party Bangers! | 8 | 1.410072 |
| **33** | 99 Percent | jacob sartorius' musical.ly's. | 5 | 1.000000 |
| **34** | 99 Percent | Topsify US Top 40 | 5 | 1.498516 |
| **45** | 99 Percent | Tomorrow's Hits | 4 | 1.662338 |
| **76** | 99 Percent | ROAD TRIP | 3 | 1.166667 |
| **77** | 99 Percent | Top Songs of the Month - Sworkit Workout Playl... | 3 | 1.055556 |
| **78** | 99 Percent | CLEAN MUSIC ONLY | 2 | 1.500000 |
| **79** | 99 Percent | 99 Percent ? iTwerk (She Twerk)i | 1 | 1.000000 |
| **80** | 99 Percent | iTunes UK TOP 40 - Continously updated | 1 | 1.416667 |
| **81** | 99 Percent | Topsify US Dance Top 50 | 1 | 1.730769 |
| **86** | 99 Percent | Tastemaker | 1 | 2.000000 |
| **87** | 99 Percent | The Weeknd - Starboy | 1 | 1.105263 |
| **88** | 99 Percent | BassBoosted??? | 1 | 1.000000 |
| **89** | 99 Percent | NEW LOVE | 1 | 2.000000 |
| **90** | 99 Percent | Music.ly songs | 1 | 1.000000 |
| **92** | 99 Percent | Hits 2000-2016 | 1 | 1.000000 |
| **93** | 99 Percent | slutty pre game | 1 | 1.000000 |
| **94** | A Boogie Wit Da Hoodie | Rap Caviar | 1982 | 1.229883 |
| **103** | A Boogie Wit Da Hoodie | Hip Hop Party | 213 | 1.147480 |
| **111** | A Boogie Wit Da Hoodie | Most DefiLitly | 145 | 1.190289 |
| **126** | A Boogie Wit Da Hoodie | RapCaviar | 143 | 1.252530 |
| **136** | A Boogie Wit Da Hoodie | Hip Hop Club Bangers: 'LUDACRIS SPECIAL' Week | 74 | 1.070755 |
| **146** | A Boogie Wit Da Hoodie | The 411 | 25 | 1.455760 |
| **167** | A Boogie Wit Da Hoodie | Roots Picnic NYC | 22 | 1.353698 |
| **177** | A Boogie Wit Da Hoodie | HipHop Hot 50 | 18 | 1.054054 |
| **180** | A Boogie Wit Da Hoodie | Highbridge The Label | 12 | 1.833333 |
| **...** | ... | ... | ... | ... |
| **1645** | flor | Chill Vibes | 3 | 1.149606 |
| **2875** | flor | Study Break | 2 | 1.204545 |
| **75** | flor | Tomorrow's Hits | 1 | 1.662338 |
| **1236** | flor | New Music Tuesday | 1 | 1.171184 |
| **1991** | flor | License To Chill | 1 | 1.000000 |
| **2002** | flor | Indie Rock Top Tracks | 1 | 1.000000 |
| **4201** | flor | Electro Rock Top Tracks | 1 | 1.000000 |
| **4202** | flor | It's ALT Good! | 1 | 1.333333 |
| **4203** | flor | SAINTE - Winter Rotation | 1 | 1.000000 |
| **1313** | gnash | Hot Hits UK | 8947 | 1.273152 |
| **366** | gnash | Today's Top Hits | 8481 | 1.292694 |
| **2517** | gnash | Heartbreaker | 2515 | 1.061522 |
| **470** | gnash | Topsify UK Top 40 | 2426 | 1.744574 |
| **2675** | gnash | Revision Ballads | 2391 | 1.076402 |
| **379** | gnash | Mellow Pop | 1979 | 1.193107 |
| **1353** | gnash | Top Tracks in The United Kingdom | 1845 | 1.331047 |
| **2558** | gnash | Chilled Pop Hits | 1144 | 1.170234 |
| **563** | gnash | The Pop List | 1038 | 1.188460 |
| **540** | gnash | New Music Monday UK | 989 | 1.184445 |
| **1427** | gnash | NEW CHILLOUT | 850 | 1.124306 |
| **312** | gnash | Top 100 tracks currently on Spotify | 684 | 1.288153 |

| | artist_name | playlist_name | count | playlist_ps |
|---|---|---|---|---|
| **1730** | gnash | Easy | 678 | 1.147103 |
| **391** | gnash | Chill Hits | 674 | 1.322059 |
| **1400** | gnash | Official UK Top 40 Charts \| 7th June 2015 \| Up... | 528 | 1.289677 |
| **3449** | gnash | Top Tracks of 2016 UK | 417 | 1.086371 |
| **1241** | gnash | Acoustic & Chill | 388 | 1.066171 |
| **480** | gnash | Pop Right Now! | 369 | 1.221854 |
| **4204** | gnash | Broken Heart | 369 | 1.049929 |
| **2202** | gnash | UK Top 40 2016 | 367 | 1.204271 |
| **4131** | livetune+ | J-Track Makunouchi | 1 | 1.000000 |

4205 rows × 4 columns

And calculate the sum of the different playlist's stream count for each artist.

In [38]:
```
artist_20_sum=df_apmerge.groupby(['artist_name']).agg({'count':sum})
df_apmerge=df_apmerge.merge(artist_20_sum,on='artist_name',how='inner',suffixes=('','_sum'))
df_apmerge
#ap_mg['wavg_passion']=ap_mg['count']*ap_mg['playlist_ps']/ap_mg['count_sum']
```

Out[38]:

| | artist_name | playlist_name | count | playlist_ps | count_sum |
|---|---|---|---|---|---|
| **0** | #90s Update | After Work House | 3 | 1.116279 | 4 |
| **1** | #90s Update | ENERGY - HIT MUSIC ONLY! | 1 | 1.387097 | 4 |
| **2** | 17 Memphis | Wild Country | 6 | 1.062500 | 6 |
| **3** | 99 Percent | Musical.ly songs | 8 | 1.000000 | 48 |
| **4** | 99 Percent | Party Bangers! | 8 | 1.410072 | 48 |
| **5** | 99 Percent | jacob sartorius' musical.ly's. | 5 | 1.000000 | 48 |
| **6** | 99 Percent | Topsify US Top 40 | 5 | 1.498516 | 48 |
| **7** | 99 Percent | Tomorrow's Hits | 4 | 1.662338 | 48 |
| **8** | 99 Percent | ROAD TRIP | 3 | 1.166667 | 48 |
| **9** | 99 Percent | Top Songs of the Month - Sworkit Workout Playl... | 3 | 1.055556 | 48 |
| **10** | 99 Percent | CLEAN MUSIC ONLY | 2 | 1.500000 | 48 |
| **11** | 99 Percent | 99 Percent ? iTwerk (She Twerk)i | 1 | 1.000000 | 48 |
| **12** | 99 Percent | iTunes UK TOP 40 - Continously updated | 1 | 1.416667 | 48 |
| **13** | 99 Percent | Topsify US Dance Top 50 | 1 | 1.730769 | 48 |
| **14** | 99 Percent | Tastemaker | 1 | 2.000000 | 48 |
| **15** | 99 Percent | The Weeknd - Starboy | 1 | 1.105263 | 48 |
| **16** | 99 Percent | BassBoosted??? | 1 | 1.000000 | 48 |
| **17** | 99 Percent | NEW LOVE | 1 | 2.000000 | 48 |
| **18** | 99 Percent | Music.ly songs | 1 | 1.000000 | 48 |
| **19** | 99 Percent | Hits 2000-2016 | 1 | 1.000000 | 48 |
| **20** | 99 Percent | slutty pre game | 1 | 1.000000 | 48 |
| **21** | A Boogie Wit Da Hoodie | Rap Caviar | 1982 | 1.229883 | 2697 |
| **22** | A Boogie Wit Da Hoodie | Hip Hop Party | 213 | 1.147480 | 2697 |
| **23** | A Boogie Wit Da Hoodie | Most DefiLitly | 145 | 1.190289 | 2697 |
| **24** | A Boogie Wit Da Hoodie | RapCaviar | 143 | 1.252530 | 2697 |
| **25** | A Boogie Wit Da Hoodie | Hip Hop Club Bangers: 'LUDACRIS SPECIAL' Week | 74 | 1.070755 | 2697 |
| **26** | A Boogie Wit Da Hoodie | The 411 | 25 | 1.455760 | 2697 |
| **27** | A Boogie Wit Da Hoodie | Roots Picnic NYC | 22 | 1.353698 | 2697 |
| **28** | A Boogie Wit Da Hoodie | HipHop Hot 50 | 18 | 1.054054 | 2697 |
| **29** | A Boogie Wit Da Hoodie | Highbridge The Label | 12 | 1.833333 | 2697 |
| **...** | ... | ... | ... | ... | ... |
| **4175** | flor | Chill Vibes | 3 | 1.149606 | 37 |
| **4176** | flor | Study Break | 2 | 1.204545 | 37 |
| **4177** | flor | Tomorrow's Hits | 1 | 1.662338 | 37 |
| **4178** | flor | New Music Tuesday | 1 | 1.171184 | 37 |
| **4179** | flor | License To Chill | 1 | 1.000000 | 37 |
| **4180** | flor | Indie Rock Top Tracks | 1 | 1.000000 | 37 |
| **4181** | flor | Electro Rock Top Tracks | 1 | 1.000000 | 37 |
| **4182** | flor | It's ALT Good! | 1 | 1.333333 | 37 |
| **4183** | flor | SAINTE - Winter Rotation | 1 | 1.000000 | 37 |
| **4184** | gnash | Hot Hits UK | 8947 | 1.273152 | 37079 |
| **4185** | gnash | Today's Top Hits | 8481 | 1.292694 | 37079 |
| **4186** | gnash | Heartbreaker | 2515 | 1.061522 | 37079 |
| **4187** | gnash | Topsify UK Top 40 | 2426 | 1.744574 | 37079 |
| **4188** | gnash | Revision Ballads | 2391 | 1.076402 | 37079 |
| **4189** | gnash | Mellow Pop | 1979 | 1.193107 | 37079 |
| **4190** | gnash | Top Tracks in The United Kingdom | 1845 | 1.331047 | 37079 |
| **4191** | gnash | Chilled Pop Hits | 1144 | 1.170234 | 37079 |
| **4192** | gnash | The Pop List | 1038 | 1.188460 | 37079 |
| **4193** | gnash | New Music Monday UK | 989 | 1.184445 | 37079 |
| **4194** | gnash | NEW CHILLOUT | 850 | 1.124306 | 37079 |
| **4195** | gnash | Top 100 tracks currently on Spotify | 684 | 1.288153 | 37079 |

| | artist_name | playlist_name | count | playlist_ps | count_sum |
|---|---|---|---|---|---|
| **4196** | gnash | Easy | 678 | 1.147103 | 37079 |
| **4197** | gnash | Chill Hits | 674 | 1.322059 | 37079 |
| **4198** | gnash | Official UK Top 40 Charts \| 7th June 2015 \| Up... | 528 | 1.289677 | 37079 |
| **4199** | gnash | Top Tracks of 2016 UK | 417 | 1.086371 | 37079 |
| **4200** | gnash | Acoustic & Chill | 388 | 1.066171 | 37079 |
| **4201** | gnash | Pop Right Now! | 369 | 1.221854 | 37079 |
| **4202** | gnash | Broken Heart | 369 | 1.049929 | 37079 |
| **4203** | gnash | UK Top 40 2016 | 367 | 1.204271 | 37079 |
| **4204** | livetune+ | J-Track Makunouchi | 1 | 1.000000 | 1 |

4205 rows × 5 columns

Here we calculate the weighted average for each playlist of each artist. For example, each artist might be included in many playlists, the stream count and the passion score of this artist in each playlist is different. We sum the total stream count for each artists in all their playlists and then use the weighted average method with multiplying the passion score to see the score of one artist in a particular playlist. The formula is: Stream count * playlist passion score / the sum of the stream count of one artist in different playlists.

In [39]:
```
df_apmerge
fly=df_apmerge['count']
fly
fly1=df_apmerge['playlist_ps']
fly1
fly2=df_apmerge['count_sum']
fly2
fly3=fly*fly1/fly2
fly3
```

Out[39]:
```
0       0.837209
1       0.346774
2       1.062500
3       0.166667
4       0.235012
5       0.104167
6       0.156095
7       0.138528
8       0.072917
9       0.065972
10      0.062500
11      0.020833
12      0.029514
13      0.036058
14      0.041667
15      0.023026
16      0.020833
17      0.041667
18      0.020833
19      0.020833
20      0.020833
21      0.903829
22      0.090624
23      0.063994
24      0.066411
25      0.029379
26      0.013494
27      0.011042
28      0.007035
29      0.008157
          ...
4175    0.093211
4176    0.065111
4177    0.044928
4178    0.031654
4179    0.027027
4180    0.027027
4181    0.027027
4182    0.036036
4183    0.027027
4184    0.307206
4185    0.295675
4186    0.072001
4187    0.114144
4188    0.069411
4189    0.063679
4190    0.066231
4191    0.036105
4192    0.033270
4193    0.031592
4194    0.025774
4195    0.023763
4196    0.020975
4197    0.024032
4198    0.018365
4199    0.012218
4200    0.011157
4201    0.012160
4202    0.010449
4203    0.011920
4204    1.000000
Length: 4205, dtype: float64
```

Here we add the weighted average score into the dataframe. 'wavg_x' is the score of each playlist and 'wavg_y' is the total weighted score of all the playlists of one artist.

In [40]:
```
df_apmerge['wavg']=fly3
playlistavg=df_apmerge.groupby('artist_name').agg({'wavg':sum})
playlistavg
df_apmerge=df_apmerge.merge(playlistavg,on='artist_name',how='inner')
df_apmerge
```

Out[40]:

| | artist_name | playlist_name | count | playlist_ps | count_sum | wavg_x | wavg_y |
|---|---|---|---|---|---|---|---|
| **0** | #90s Update | After Work House | 3 | 1.116279 | 4 | 0.837209 | 1.183983 |
| **1** | #90s Update | ENERGY - HIT MUSIC ONLY! | 1 | 1.387097 | 4 | 0.346774 | 1.183983 |
| **2** | 17 Memphis | Wild Country | 6 | 1.062500 | 6 | 1.062500 | 1.062500 |
| **3** | 99 Percent | Musical.ly songs | 8 | 1.000000 | 48 | 0.166667 | 1.277956 |
| **4** | 99 Percent | Party Bangers! | 8 | 1.410072 | 48 | 0.235012 | 1.277956 |
| **5** | 99 Percent | jacob sartorius' musical.ly's. | 5 | 1.000000 | 48 | 0.104167 | 1.277956 |
| **6** | 99 Percent | Topsify US Top 40 | 5 | 1.498516 | 48 | 0.156095 | 1.277956 |
| **7** | 99 Percent | Tomorrow's Hits | 4 | 1.662338 | 48 | 0.138528 | 1.277956 |
| **8** | 99 Percent | ROAD TRIP | 3 | 1.166667 | 48 | 0.072917 | 1.277956 |
| **9** | 99 Percent | Top Songs of the Month - Sworkit Workout Playl... | 3 | 1.055556 | 48 | 0.065972 | 1.277956 |
| **10** | 99 Percent | CLEAN MUSIC ONLY | 2 | 1.500000 | 48 | 0.062500 | 1.277956 |
| **11** | 99 Percent | 99 Percent ? iTwerk (She Twerk)i | 1 | 1.000000 | 48 | 0.020833 | 1.277956 |
| **12** | 99 Percent | iTunes UK TOP 40 - Continously updated | 1 | 1.416667 | 48 | 0.029514 | 1.277956 |
| **13** | 99 Percent | Topsify US Dance Top 50 | 1 | 1.730769 | 48 | 0.036058 | 1.277956 |
| **14** | 99 Percent | Tastemaker | 1 | 2.000000 | 48 | 0.041667 | 1.277956 |
| **15** | 99 Percent | The Weeknd - Starboy | 1 | 1.105263 | 48 | 0.023026 | 1.277956 |
| **16** | 99 Percent | BassBoosted??? | 1 | 1.000000 | 48 | 0.020833 | 1.277956 |
| **17** | 99 Percent | NEW LOVE | 1 | 2.000000 | 48 | 0.041667 | 1.277956 |
| **18** | 99 Percent | Music.ly songs | 1 | 1.000000 | 48 | 0.020833 | 1.277956 |
| **19** | 99 Percent | Hits 2000-2016 | 1 | 1.000000 | 48 | 0.020833 | 1.277956 |
| **20** | 99 Percent | slutty pre game | 1 | 1.000000 | 48 | 0.020833 | 1.277956 |
| **21** | A Boogie Wit Da Hoodie | Rap Caviar | 1982 | 1.229883 | 2697 | 0.903829 | 1.224615 |
| **22** | A Boogie Wit Da Hoodie | Hip Hop Party | 213 | 1.147480 | 2697 | 0.090624 | 1.224615 |
| **23** | A Boogie Wit Da Hoodie | Most DefiLitly | 145 | 1.190289 | 2697 | 0.063994 | 1.224615 |
| **24** | A Boogie Wit Da Hoodie | RapCaviar | 143 | 1.252530 | 2697 | 0.066411 | 1.224615 |
| **25** | A Boogie Wit Da Hoodie | Hip Hop Club Bangers: 'LUDACRIS SPECIAL' Week | 74 | 1.070755 | 2697 | 0.029379 | 1.224615 |
| **26** | A Boogie Wit Da Hoodie | The 411 | 25 | 1.455760 | 2697 | 0.013494 | 1.224615 |
| **27** | A Boogie Wit Da Hoodie | Roots Picnic NYC | 22 | 1.353698 | 2697 | 0.011042 | 1.224615 |
| **28** | A Boogie Wit Da Hoodie | HipHop Hot 50 | 18 | 1.054054 | 2697 | 0.007035 | 1.224615 |
| **29** | A Boogie Wit Da Hoodie | Highbridge The Label | 12 | 1.833333 | 2697 | 0.008157 | 1.224615 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **4175** | flor | Chill Vibes | 3 | 1.149606 | 37 | 0.093211 | 1.160215 |
| **4176** | flor | Study Break | 2 | 1.204545 | 37 | 0.065111 | 1.160215 |
| **4177** | flor | Tomorrow's Hits | 1 | 1.662338 | 37 | 0.044928 | 1.160215 |
| **4178** | flor | New Music Tuesday | 1 | 1.171184 | 37 | 0.031654 | 1.160215 |
| **4179** | flor | License To Chill | 1 | 1.000000 | 37 | 0.027027 | 1.160215 |
| **4180** | flor | Indie Rock Top Tracks | 1 | 1.000000 | 37 | 0.027027 | 1.160215 |
| **4181** | flor | Electro Rock Top Tracks | 1 | 1.000000 | 37 | 0.027027 | 1.160215 |
| **4182** | flor | It's ALT Good! | 1 | 1.333333 | 37 | 0.036036 | 1.160215 |
| **4183** | flor | SAINTE - Winter Rotation | 1 | 1.000000 | 37 | 0.027027 | 1.160215 |
| **4184** | gnash | Hot Hits UK | 8947 | 1.273152 | 37079 | 0.307206 | 1.260124 |
| **4185** | gnash | Today's Top Hits | 8481 | 1.292694 | 37079 | 0.295675 | 1.260124 |
| **4186** | gnash | Heartbreaker | 2515 | 1.061522 | 37079 | 0.072001 | 1.260124 |
| **4187** | gnash | Topsify UK Top 40 | 2426 | 1.744574 | 37079 | 0.114144 | 1.260124 |
| **4188** | gnash | Revision Ballads | 2391 | 1.076402 | 37079 | 0.069411 | 1.260124 |
| **4189** | gnash | Mellow Pop | 1979 | 1.193107 | 37079 | 0.063679 | 1.260124 |
| **4190** | gnash | Top Tracks in The United Kingdom | 1845 | 1.331047 | 37079 | 0.066231 | 1.260124 |
| **4191** | gnash | Chilled Pop Hits | 1144 | 1.170234 | 37079 | 0.036105 | 1.260124 |
| **4192** | gnash | The Pop List | 1038 | 1.188460 | 37079 | 0.033270 | 1.260124 |
| **4193** | gnash | New Music Monday UK | 989 | 1.184445 | 37079 | 0.031592 | 1.260124 |
| **4194** | gnash | NEW CHILLOUT | 850 | 1.124306 | 37079 | 0.025774 | 1.260124 |
| **4195** | gnash | Top 100 tracks currently on Spotify | 684 | 1.288153 | 37079 | 0.023763 | 1.260124 |

| | artist_name | playlist_name | count | playlist_ps | count_sum | wavg_x | wavg_y |
|---|---|---|---|---|---|---|---|
| **4196** | gnash | Easy | 678 | 1.147103 | 37079 | 0.020975 | 1.260124 |
| **4197** | gnash | Chill Hits | 674 | 1.322059 | 37079 | 0.024032 | 1.260124 |
| **4198** | gnash | Official UK Top 40 Charts \| 7th June 2015 \| Up... | 528 | 1.289677 | 37079 | 0.018365 | 1.260124 |
| **4199** | gnash | Top Tracks of 2016 UK | 417 | 1.086371 | 37079 | 0.012218 | 1.260124 |
| **4200** | gnash | Acoustic & Chill | 388 | 1.066171 | 37079 | 0.011157 | 1.260124 |
| **4201** | gnash | Pop Right Now! | 369 | 1.221854 | 37079 | 0.012160 | 1.260124 |
| **4202** | gnash | Broken Heart | 369 | 1.049929 | 37079 | 0.010449 | 1.260124 |
| **4203** | gnash | UK Top 40 2016 | 367 | 1.204271 | 37079 | 0.011920 | 1.260124 |
| **4204** | livetune+ | J-Track Makunouchi | 1 | 1.000000 | 1 | 1.000000 | 1.000000 |

4205 rows × 7 columns

After that, we create a dataframe regarding to the total weighted score of each artist and merge this to the dataframe 'df_artist_features' which contains all the useful features variables.

In [41]:
```
df_apmerge
df_apmerge.rename(columns={'wavg_x':'playlistavg'},inplace=True)
df_apmerge.rename(columns={'wavg_y':'weightedscore'},inplace=True)
df_apmerge
df_wght=df_apmerge[['artist_name','weightedscore']]
df_wght=df_wght.drop_duplicates()
df_wght=df_wght.set_index('artist_name')
df_wght
```

Out[41]:

| | weightedscore |
|---|---|
| **artist_name** | |
| **#90s Update** | 1.183983 |
| **17 Memphis** | 1.062500 |
| **99 Percent** | 1.277956 |
| **A Boogie Wit Da Hoodie** | 1.224615 |
| **A Boogie Wit da Hoodie** | 1.272593 |
| **A R I Z O N A** | 1.245686 |
| **AGWA** | 2.000000 |
| **ALP** | 1.000000 |
| **AVVAH** | 1.092715 |
| **AXSHN** | 1.323853 |
| **Absofacto** | 1.143588 |
| **Adam Sample** | 1.545097 |
| **Adan Carmona** | 2.767196 |
| **Adia Victoria** | 1.050996 |
| **Alessio Bernabei** | 1.114286 |
| **Alex Hoyer** | 1.153571 |
| **Alex Roy** | 2.767196 |
| **Alexander Brown** | 1.084948 |
| **Alexander Cardinale** | 1.100414 |
| **Alexander Charles** | 1.261209 |
| **Alice** | 1.210582 |
| **Aliose** | 1.333333 |
| **All Tvvins** | 1.115551 |
| **Alma** | 1.110234 |
| **Amir** | 1.099209 |
| **Andrew von Oeyen** | 1.000000 |
| **Andy Bros** | 1.038462 |
| **Angela Torres** | 1.000000 |
| **Angeles** | 1.312195 |
| **Angelica Garcia** | 1.075000 |
| **...** | ... |
| **Viktor Frisk** | 1.500000 |
| **Vince Pope** | 1.000000 |
| **Vinyl on HBO** | 1.190875 |
| **Virgul** | 1.272572 |
| **WEDNESDAY CAMPANELLA** | 1.833333 |
| **WSTRN** | 1.299935 |
| **Waylon** | 2.000000 |
| **We Are Messengers** | 1.051429 |
| **Whethan** | 1.429354 |
| **Wild Youth** | 1.122784 |
| **Will Joseph Cook** | 1.343154 |
| **Willy William** | 1.209903 |
| **Xavier Dunn** | 1.054221 |
| **YFN Lucci** | 1.200079 |
| **YONAKA** | 1.068039 |
| **Yasutaka Nakata** | 1.114286 |
| **Yellow Claw** | 1.173007 |
| **Young Spray** | 1.028498 |
| **YoungBoy Never Broke Again** | 1.378724 |
| **Youngboy Never Broke Again** | 1.335477 |

| | weightedscore |
|---|---|
| **artist_name** | |
| **Yvng Swag** | 1.497963 |
| **Zac Brown** | 1.229198 |
| **Zak Abel** | 1.222618 |
| **Zarcort** | 1.875000 |
| **Zion & Lennox** | 1.265757 |
| **birthday** | 1.000000 |
| **dvsn** | 1.128136 |
| **flor** | 1.160215 |
| **gnash** | 1.260124 |
| **livetune+** | 1.000000 |

471 rows × 1 columns

In [42]:
```
df_artist_features['weightedscore']=df_wght['weightedscore']
df_artist_features
```

Out[42]:

| artist_name | success | count | number_users | passion_score | weightedscore |
|---|---|---|---|---|---|
| #90s Update | 0 | 16 | 15 | 1.066667 | 1.183983 |
| 17 Memphis | 0 | 12 | 12 | 1.000000 | 1.062500 |
| 2D | 0 | 1 | 1 | 1.000000 | NaN |
| 3JS | 0 | 5 | 4 | 1.250000 | NaN |
| 99 Percent | 0 | 1291 | 1189 | 1.085786 | 1.277956 |
| A Boogie Wit Da Hoodie | 0 | 9904 | 7713 | 1.284066 | 1.224615 |
| A Boogie Wit da Hoodie | 1 | 13264 | 11154 | 1.189170 | 1.272593 |
| A R I Z O N A | 1 | 68830 | 58987 | 1.166867 | 1.245686 |
| AGWA | 0 | 3 | 3 | 1.000000 | 2.000000 |
| ALMA | 0 | 8 | 8 | 1.000000 | NaN |
| ALP | 0 | 4 | 4 | 1.000000 | 1.000000 |
| AV AV AV | 0 | 57 | 54 | 1.055556 | NaN |
| AVVAH | 0 | 20 | 20 | 1.000000 | 1.092715 |
| AXSHN | 1 | 112 | 109 | 1.027523 | 1.323853 |
| Absofacto | 1 | 138 | 136 | 1.014706 | 1.143588 |
| Adam Sample | 0 | 42 | 42 | 1.000000 | 1.545097 |
| Adan Carmona | 0 | 14 | 12 | 1.166667 | 2.767196 |
| Adia Victoria | 0 | 671 | 607 | 1.105437 | 1.050996 |
| Alcatraz | 0 | 7 | 7 | 1.000000 | NaN |
| Alessio Bernabei | 0 | 191 | 156 | 1.224359 | 1.114286 |
| Alex Hoyer | 0 | 27 | 27 | 1.000000 | 1.153571 |
| Alex Roy | 0 | 3 | 3 | 1.000000 | 2.767196 |
| Alexander Brown | 0 | 147 | 141 | 1.042553 | 1.084948 |
| Alexander Cardinale | 0 | 584 | 552 | 1.057971 | 1.100414 |
| Alexander Charles | 0 | 53 | 51 | 1.039216 | 1.261209 |
| Alice | 0 | 45 | 39 | 1.153846 | 1.210582 |
| Aliose | 0 | 21 | 18 | 1.166667 | 1.333333 |
| All Tvvins | 1 | 10446 | 9156 | 1.140891 | 1.115551 |
| Alma | 0 | 994 | 832 | 1.194712 | 1.110234 |
| Amaro Ferreiro | 0 | 8 | 4 | 2.000000 | NaN |
| ... | ... | ... | ... | ... | ... |
| Wild Youth | 1 | 284 | 284 | 1.000000 | 1.122784 |
| Wildling | 0 | 17 | 17 | 1.000000 | NaN |
| Will Joseph Cook | 1 | 15322 | 12585 | 1.217481 | 1.343154 |
| Willy William | 0 | 1916 | 1607 | 1.192284 | 1.209903 |
| Witek Muzyk Ulicy | 0 | 17 | 11 | 1.545455 | NaN |
| Wudstik | 0 | 23 | 23 | 1.000000 | NaN |
| Xavier Cugat y su orquesta | 0 | 1 | 1 | 1.000000 | NaN |
| Xavier Dunn | 1 | 5824 | 5406 | 1.077321 | 1.054221 |
| YFN Lucci | 0 | 1487 | 1395 | 1.065950 | 1.200079 |
| YONAKA | 0 | 340 | 336 | 1.011905 | 1.068039 |
| Yasutaka Nakata | 0 | 90 | 77 | 1.168831 | 1.114286 |
| Yellow Claw | 1 | 6734 | 5948 | 1.132145 | 1.173007 |
| Ylric Illians | 0 | 3 | 1 | 3.000000 | NaN |
| Young F | 0 | 1 | 1 | 1.000000 | NaN |
| Young Spray | 0 | 667 | 634 | 1.052050 | 1.028498 |
| YoungBoy Never Broke Again | 0 | 783 | 621 | 1.260870 | 1.378724 |
| Youngboy Never Broke Again | 0 | 104 | 75 | 1.386667 | 1.335477 |
| Yvng Swag | 0 | 238 | 226 | 1.053097 | 1.497963 |
| Zac Brown | 0 | 59 | 59 | 1.000000 | 1.229198 |
| Zak & Diego | 0 | 12 | 10 | 1.200000 | NaN |

| artist_name | success | count | number_users | passion_score | weightedscore |
|---|---|---|---|---|---|
| Zak Abel | 1 | 26966 | 23417 | 1.151557 | 1.222618 |
| Zakopower | 0 | 1 | 1 | 1.000000 | NaN |
| Zarcort | 0 | 25 | 18 | 1.388889 | 1.875000 |
| Zbigniew Kurtycz | 0 | 2 | 2 | 1.000000 | NaN |
| Zion & Lennox | 1 | 10721 | 9303 | 1.152424 | 1.265757 |
| birthday | 0 | 20 | 20 | 1.000000 | 1.000000 |
| dvsn | 1 | 25168 | 18712 | 1.345019 | 1.128136 |
| flor | 1 | 109 | 108 | 1.009259 | 1.160215 |
| gnash | 1 | 165683 | 146108 | 1.133976 | 1.260124 |
| livetune+ | 0 | 7 | 6 | 1.166667 | 1.000000 |

661 rows × 5 columns

In [43]:
```
artist_wavg=df_apmerge.groupby('artist_name').agg({'weightedscore':sum})
```

## User-base features

In this part, we will identify the user-base features by using the age and gender columns to create an audience profile per artist which includes gender percentage breakdown and age vector quantization.

**ACTION: User features**

Write useful functions to create new user features, like those listed in the cell above.

Are there other sensible ones you could suggest? Work in your group to think about what other features might be useful and whether you can calculate them with the data you have. Justify your reasoning.

In [44]:
```
# Gender breakdown

df_gen=data1.copy()

def gender_percentage(df_gen):
    if (len(df_gen)!=0):
        df_gen=df_gen.drop_duplicates(subset=['customer_id'])
        perc=len(df_gen[df_gen.gender=='male'])/len(df_gen)
        return perc
    else:
        return 0

gender_per=df_gen.groupby('artist_name').apply(gender_percentage)
gender_per=pd.DataFrame(gender_per)
gender_per.rename(columns={0:'malepercentage'},inplace=True)
gender_per
df_artist_features['malepercentage']=gender_per['malepercentage']
df_artist_features
```

Out[44]:

| artist_name | success | count | number_users | passion_score | weightedscore | malepercentage |
|---|---|---|---|---|---|---|
| #90s Update | 0 | 16 | 15 | 1.066667 | 1.183983 | 0.600000 |
| 17 Memphis | 0 | 12 | 12 | 1.000000 | 1.062500 | 0.333333 |
| 2D | 0 | 1 | 1 | 1.000000 | NaN | 1.000000 |
| 3JS | 0 | 5 | 4 | 1.250000 | NaN | 0.750000 |
| 99 Percent | 0 | 1291 | 1189 | 1.085786 | 1.277956 | 0.315391 |
| A Boogie Wit Da Hoodie | 0 | 9904 | 7713 | 1.284066 | 1.224615 | 0.712952 |
| A Boogie Wit da Hoodie | 1 | 13264 | 11154 | 1.189170 | 1.272593 | 0.670522 |
| A R I Z O N A | 1 | 68830 | 58987 | 1.166867 | 1.245686 | 0.473392 |
| AGWA | 0 | 3 | 3 | 1.000000 | 2.000000 | 1.000000 |
| ALMA | 0 | 8 | 8 | 1.000000 | NaN | 0.500000 |
| ALP | 0 | 4 | 4 | 1.000000 | 1.000000 | 0.250000 |
| AV AV AV | 0 | 57 | 54 | 1.055556 | NaN | 0.796296 |
| AVVAH | 0 | 20 | 20 | 1.000000 | 1.092715 | 0.350000 |
| AXSHN | 1 | 112 | 109 | 1.027523 | 1.323853 | 0.449541 |
| Absofacto | 1 | 138 | 136 | 1.014706 | 1.143588 | 0.588235 |
| Adam Sample | 0 | 42 | 42 | 1.000000 | 1.545097 | 0.404762 |
| Adan Carmona | 0 | 14 | 12 | 1.166667 | 2.767196 | 0.416667 |
| Adia Victoria | 0 | 671 | 607 | 1.105437 | 1.050996 | 0.635914 |
| Alcatraz | 0 | 7 | 7 | 1.000000 | NaN | 0.571429 |
| Alessio Bernabei | 0 | 191 | 156 | 1.224359 | 1.114286 | 0.506410 |
| Alex Hoyer | 0 | 27 | 27 | 1.000000 | 1.153571 | 0.407407 |
| Alex Roy | 0 | 3 | 3 | 1.000000 | 2.767196 | 0.000000 |
| Alexander Brown | 0 | 147 | 141 | 1.042553 | 1.084948 | 0.631206 |
| Alexander Cardinale | 0 | 584 | 552 | 1.057971 | 1.100414 | 0.394928 |
| Alexander Charles | 0 | 53 | 51 | 1.039216 | 1.261209 | 0.823529 |
| Alice | 0 | 45 | 39 | 1.153846 | 1.210582 | 0.692308 |
| Aliose | 0 | 21 | 18 | 1.166667 | 1.333333 | 0.500000 |
| All Tvvins | 1 | 10446 | 9156 | 1.140891 | 1.115551 | 0.666776 |
| Alma | 0 | 994 | 832 | 1.194712 | 1.110234 | 0.643029 |
| Amaro Ferreiro | 0 | 8 | 4 | 2.000000 | NaN | 0.750000 |
| ... | ... | ... | ... | ... | ... | ... |
| Wild Youth | 1 | 284 | 284 | 1.000000 | 1.122784 | 0.485915 |
| Wildling | 0 | 17 | 17 | 1.000000 | NaN | 0.764706 |
| Will Joseph Cook | 1 | 15322 | 12585 | 1.217481 | 1.343154 | 0.469130 |
| Willy William | 0 | 1916 | 1607 | 1.192284 | 1.209903 | 0.500311 |
| Witek Muzyk Ulicy | 0 | 17 | 11 | 1.545455 | NaN | 0.909091 |
| Wudstik | 0 | 23 | 23 | 1.000000 | NaN | 0.565217 |
| Xavier Cugat y su orquesta | 0 | 1 | 1 | 1.000000 | NaN | 0.000000 |
| Xavier Dunn | 1 | 5824 | 5406 | 1.077321 | 1.054221 | 0.497595 |
| YFN Lucci | 0 | 1487 | 1395 | 1.065950 | 1.200079 | 0.774194 |
| YONAKA | 0 | 340 | 336 | 1.011905 | 1.068039 | 0.642857 |
| Yasutaka Nakata | 0 | 90 | 77 | 1.168831 | 1.114286 | 0.688312 |
| Yellow Claw | 1 | 6734 | 5948 | 1.132145 | 1.173007 | 0.690484 |
| Ylric Illians | 0 | 3 | 1 | 3.000000 | NaN | 1.000000 |
| Young F | 0 | 1 | 1 | 1.000000 | NaN | 1.000000 |
| Young Spray | 0 | 667 | 634 | 1.052050 | 1.028498 | 0.799685 |
| YoungBoy Never Broke Again | 0 | 783 | 621 | 1.260870 | 1.378724 | 0.805153 |
| Youngboy Never Broke Again | 0 | 104 | 75 | 1.386667 | 1.335477 | 0.826667 |
| Yvng Swag | 0 | 238 | 226 | 1.053097 | 1.497963 | 0.539823 |
| Zac Brown | 0 | 59 | 59 | 1.000000 | 1.229198 | 0.559322 |
| Zak & Diego | 0 | 12 | 10 | 1.200000 | NaN | 0.500000 |

| | success | count | number_users | passion_score | weightedscore | malepercentage |
|---|---|---|---|---|---|---|
| **artist_name** | | | | | | |
| **Zak Abel** | 1 | 26966 | 23417 | 1.151557 | 1.222618 | 0.465559 |
| **Zakopower** | 0 | 1 | 1 | 1.000000 | NaN | 1.000000 |
| **Zarcort** | 0 | 25 | 18 | 1.388889 | 1.875000 | 0.777778 |
| **Zbigniew Kurtycz** | 0 | 2 | 2 | 1.000000 | NaN | 1.000000 |
| **Zion & Lennox** | 1 | 10721 | 9303 | 1.152424 | 1.265757 | 0.450715 |
| **birthday** | 0 | 20 | 20 | 1.000000 | 1.000000 | 0.400000 |
| **dvsn** | 1 | 25168 | 18712 | 1.345019 | 1.128136 | 0.452170 |
| **flor** | 1 | 109 | 108 | 1.009259 | 1.160215 | 0.527778 |
| **gnash** | 1 | 165683 | 146108 | 1.133976 | 1.260124 | 0.400450 |
| **livetune+** | 0 | 7 | 6 | 1.166667 | 1.000000 | 0.500000 |

661 rows × 6 columns

We add the percentage of male users of each artist into the dataframe firstly. For the age breakdown, we use the birth year of users in original dataset to calculate the age of users. We divided users into 4 groups where age between 0-18 belongs to Dependent, age between 18-25 belongs to youngadult, age between 25-50 belongs to adult and age between 50-100 belongs to senior. To avoid any collinearity, we drop the variable 'adult' later. And finally we add these user-base features into dataframe 'df_artist_features'.

In [45]:
```
# Age breakdown
import numpy as np
def age_percentages(df_age):
    df_age=df_age.dropna(subset=['birth_year'])

    df_age.birth_year=np.float64(df_age.birth_year)
    if 'DataTime' not in df_age.columns:
        df_age['DataTime']=pd.to_datetime(df_age.log_time)
    df_age['age']=df_age.year - df_age.birth_year
    df_age=df_age.drop_duplicates(subset=['customer_id'])
    bins=[0,18,25,50,100]
    group_names=['Dependent','YoungAdult','Adult','Senior']
    categories=pd.cut(df_age['age'],bins,labels=group_names)
    return pd.DataFrame([{y:x/categories.value_counts().sum() for x,y in zip(categories.value_counts(),categories.value_counts().key
s())}])
age_per=df_gen.groupby('artist_name').apply(age_percentages)
age_per=pd.DataFrame(age_per)
age_per=age_per.reset_index()
age_per=age_per.drop(columns=['level_1'])
#to avoid any collinearity
age_per=age_per.drop(columns=['Adult'])

age_per=age_per.set_index('artist_name')
age_per
df_artist_features[['Dependent','Senior','YoungAdult']]=age_per[['Dependent','Senior','YoungAdult']]
df_artist_features
```

```
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/ipykernel/__main__.py:14: RuntimeWarning: invalid value encountered in long
_scalars
```

Out[45]:

| artist_name | success | count | number_users | passion_score | weightedscore | malepercentage | Dependent | Senior | YoungAdult |
|---|---|---|---|---|---|---|---|---|---|
| **#90s Update** | 0 | 16 | 15 | 1.066667 | 1.183983 | 0.600000 | 0.066667 | 0.000000 | 0.200000 |
| **17 Memphis** | 0 | 12 | 12 | 1.000000 | 1.062500 | 0.333333 | 0.166667 | 0.083333 | 0.416667 |
| **2D** | 0 | 1 | 1 | 1.000000 | NaN | 1.000000 | 1.000000 | 0.000000 | 0.000000 |
| **3JS** | 0 | 5 | 4 | 1.250000 | NaN | 0.750000 | 0.000000 | 0.000000 | 0.250000 |
| **99 Percent** | 0 | 1291 | 1189 | 1.085786 | 1.277956 | 0.315391 | 0.379661 | 0.027966 | 0.347458 |
| **A Boogie Wit Da Hoodie** | 0 | 9904 | 7713 | 1.284066 | 1.224615 | 0.712952 | 0.222888 | 0.020725 | 0.498175 |
| **A Boogie Wit da Hoodie** | 1 | 13264 | 11154 | 1.189170 | 1.272593 | 0.670522 | 0.269546 | 0.025340 | 0.438182 |
| **A R I Z O N A** | 1 | 68830 | 58987 | 1.166867 | 1.245686 | 0.473392 | 0.140033 | 0.034987 | 0.406650 |
| **AGWA** | 0 | 3 | 3 | 1.000000 | 2.000000 | 1.000000 | 0.000000 | 0.666667 | 0.000000 |
| **ALMA** | 0 | 8 | 8 | 1.000000 | NaN | 0.500000 | 0.000000 | 0.125000 | 0.375000 |
| **ALP** | 0 | 4 | 4 | 1.000000 | 1.000000 | 0.250000 | 0.500000 | 0.000000 | 0.250000 |
| **AV AV AV** | 0 | 57 | 54 | 1.055556 | NaN | 0.796296 | 0.000000 | 0.000000 | 0.277778 |
| **AVVAH** | 0 | 20 | 20 | 1.000000 | 1.092715 | 0.350000 | 0.100000 | 0.000000 | 0.550000 |
| **AXSHN** | 1 | 112 | 109 | 1.027523 | 1.323853 | 0.449541 | 0.214953 | 0.046729 | 0.317757 |
| **Absofacto** | 1 | 138 | 136 | 1.014706 | 1.143588 | 0.588235 | 0.117647 | 0.022059 | 0.411765 |
| **Adam Sample** | 0 | 42 | 42 | 1.000000 | 1.545097 | 0.404762 | 0.119048 | 0.023810 | 0.642857 |
| **Adan Carmona** | 0 | 14 | 12 | 1.166667 | 2.767196 | 0.416667 | 0.000000 | 0.000000 | 0.272727 |
| **Adia Victoria** | 0 | 671 | 607 | 1.105437 | 1.050996 | 0.635914 | 0.044702 | 0.082781 | 0.241722 |
| **Alcatraz** | 0 | 7 | 7 | 1.000000 | NaN | 0.571429 | 0.000000 | 0.142857 | 0.142857 |
| **Alessio Bernabei** | 0 | 191 | 156 | 1.224359 | 1.114286 | 0.506410 | 0.089744 | 0.025641 | 0.384615 |
| **Alex Hoyer** | 0 | 27 | 27 | 1.000000 | 1.153571 | 0.407407 | 0.148148 | 0.037037 | 0.296296 |
| **Alex Roy** | 0 | 3 | 3 | 1.000000 | 2.767196 | 0.000000 | 0.000000 | 0.000000 | 0.666667 |
| **Alexander Brown** | 0 | 147 | 141 | 1.042553 | 1.084948 | 0.631206 | 0.028369 | 0.042553 | 0.290780 |
| **Alexander Cardinale** | 0 | 584 | 552 | 1.057971 | 1.100414 | 0.394928 | 0.105647 | 0.060109 | 0.355191 |
| **Alexander Charles** | 0 | 53 | 51 | 1.039216 | 1.261209 | 0.823529 | 0.137255 | 0.000000 | 0.529412 |
| **Alice** | 0 | 45 | 39 | 1.153846 | 1.210582 | 0.692308 | 0.102564 | 0.076923 | 0.307692 |
| **Aliose** | 0 | 21 | 18 | 1.166667 | 1.333333 | 0.500000 | 0.000000 | 0.055556 | 0.277778 |
| **All Tvvins** | 1 | 10446 | 9156 | 1.140891 | 1.115551 | 0.666776 | 0.115156 | 0.030136 | 0.396942 |
| **Alma** | 0 | 994 | 832 | 1.194712 | 1.110234 | 0.643029 | 0.153012 | 0.038554 | 0.346988 |
| **Amaro Ferreiro** | 0 | 8 | 4 | 2.000000 | NaN | 0.750000 | 0.000000 | 0.000000 | 0.333333 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **Wild Youth** | 1 | 284 | 284 | 1.000000 | 1.122784 | 0.485915 | 0.031690 | 0.140845 | 0.211268 |
| **Wildling** | 0 | 17 | 17 | 1.000000 | NaN | 0.764706 | 0.058824 | 0.117647 | 0.352941 |
| **Will Joseph Cook** | 1 | 15322 | 12585 | 1.217481 | 1.343154 | 0.469130 | 0.148583 | 0.035848 | 0.432974 |
| **Willy William** | 0 | 1916 | 1607 | 1.192284 | 1.209903 | 0.500311 | 0.130707 | 0.028143 | 0.472170 |
| **Witek Muzyk Ulicy** | 0 | 17 | 11 | 1.545455 | NaN | 0.909091 | 0.000000 | 0.000000 | 0.181818 |
| **Wudstik** | 0 | 23 | 23 | 1.000000 | NaN | 0.565217 | 0.000000 | 0.043478 | 0.434783 |
| **Xavier Cugat y su orquesta** | 0 | 1 | 1 | 1.000000 | NaN | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **Xavier Dunn** | 1 | 5824 | 5406 | 1.077321 | 1.054221 | 0.497595 | 0.070619 | 0.048318 | 0.366475 |
| **YFN Lucci** | 0 | 1487 | 1395 | 1.065950 | 1.200079 | 0.774194 | 0.205904 | 0.017279 | 0.493880 |
| **YONAKA** | 0 | 340 | 336 | 1.011905 | 1.068039 | 0.642857 | 0.089552 | 0.026866 | 0.367164 |
| **Yasutaka Nakata** | 0 | 90 | 77 | 1.168831 | 1.114286 | 0.688312 | 0.092105 | 0.013158 | 0.592105 |
| **Yellow Claw** | 1 | 6734 | 5948 | 1.132145 | 1.173007 | 0.690484 | 0.113360 | 0.023279 | 0.469130 |
| **Ylric Illians** | 0 | 3 | 1 | 3.000000 | NaN | 1.000000 | 0.000000 | 0.000000 | 0.000000 |
| **Young F** | 0 | 1 | 1 | 1.000000 | NaN | 1.000000 | 0.000000 | 0.000000 | 1.000000 |
| **Young Spray** | 0 | 667 | 634 | 1.052050 | 1.028498 | 0.799685 | 0.141046 | 0.020602 | 0.522979 |
| **YoungBoy Never Broke Again** | 0 | 783 | 621 | 1.260870 | 1.378724 | 0.805153 | 0.197411 | 0.019417 | 0.487055 |
| **Youngboy Never Broke Again** | 0 | 104 | 75 | 1.386667 | 1.335477 | 0.826667 | 0.246575 | 0.013699 | 0.493151 |
| **Yvng Swag** | 0 | 238 | 226 | 1.053097 | 1.497963 | 0.539823 | 0.351351 | 0.009009 | 0.427928 |
| **Zac Brown** | 0 | 59 | 59 | 1.000000 | 1.229198 | 0.559322 | 0.050847 | 0.084746 | 0.305085 |
| **Zak & Diego** | 0 | 12 | 10 | 1.200000 | NaN | 0.500000 | 0.000000 | 0.000000 | 0.222222 |

| artist_name | success | count | number_users | passion_score | weightedscore | malepercentage | Dependent | Senior | YoungAdult |
|---|---|---|---|---|---|---|---|---|---|
| Zak Abel | 1 | 26966 | 23417 | 1.151557 | 1.222618 | 0.465559 | 0.099047 | 0.039911 | 0.429319 |
| Zakopower | 0 | 1 | 1 | 1.000000 | NaN | 1.000000 | 0.000000 | 0.000000 | 0.000000 |
| Zarcort | 0 | 25 | 18 | 1.388889 | 1.875000 | 0.777778 | 0.166667 | 0.055556 | 0.388889 |
| Zbigniew Kurtycz | 0 | 2 | 2 | 1.000000 | NaN | 1.000000 | 0.000000 | 0.000000 | 0.000000 |
| Zion & Lennox | 1 | 10721 | 9303 | 1.152424 | 1.265757 | 0.450715 | 0.074226 | 0.024814 | 0.417413 |
| birthday | 0 | 20 | 20 | 1.000000 | 1.000000 | 0.400000 | 0.250000 | 0.000000 | 0.400000 |
| dvsn | 1 | 25168 | 18712 | 1.345019 | 1.128136 | 0.452170 | 0.137550 | 0.023677 | 0.493021 |
| flor | 1 | 109 | 108 | 1.009259 | 1.160215 | 0.527778 | 0.157407 | 0.037037 | 0.370370 |
| gnash | 1 | 165683 | 146108 | 1.133976 | 1.260124 | 0.400450 | 0.231351 | 0.029931 | 0.389555 |
| livetune+ | 0 | 7 | 6 | 1.166667 | 1.000000 | 0.500000 | 0.333333 | 0.000000 | 0.166667 |

661 rows × 9 columns

**Principle Component Analysis**

The data also contains a partial region code of the listener. We might want to consider including the regional breakdown of streams per artist as a feature of our model, to know if streams for certain regions are particularly influential on the future performance of an artist.

However, we have over 400 unique regions and like playlists, including them all would lead to too many features and a large sparse matrix. One way in which to extract relevant 'generalized' features of each region would be to incorporate census and demographic data, from publicly available datasets.

This is however beyond the scope of this courswork. Instead, a better way to summarize the impact of regional variation in streams is to use dimensionality reduction techniques. Here we will use Principle Component Analysis (PCA) to capture the regional variation in stream count.

PCA captures the majority of variation in the original feature set and represents it as a set of new orthogonal variables. Each 'component' of PCA is a linear combination of every feature, i.e. playlist in the dataset. Use `scikit-learn` 's PCA module (Pedregosa, et al., 2011) for generating PCA components.

For a comprehensive understanding of how sklearn's PCA module works, please refer to the sklearn documentation. We will using 10 components of PCA in our model.

*Note: We could also apply a similar method to condense variation in stream across the 19,600 different playlists in our dataset.*

Please refer to "L7 - Dimensionality Reduction" in Week 7 as necessary.

**ACTION: PCA features**

Write useful functions to create new user feature based on regions data.

Are there other sensible features you could suggest? Work in your group to think about what other features might be useful and whether you can calculate them with the data you have. Justify your reasoning.

**WARNING: PCA features**

If you struggle to complete this section successfully **please email me** and we will provide code to compute the new features. This will help with performance of the classifier in the next stage.

Firstly, we construct the streams region code breakdown for artist dataset by grouping data 1 by artist name and region code, and counting the nuber of region code rows of a particular region code. Name the data as artist_region_stream.

In [46]:
```
artist_region_stream=data1.groupby(['artist_name','region_code']).agg({'region_code':'count'})
```

We pivot the region code level of the index labels, return a DataFrame artist_region_stream_unstack having a new level of column labels whose inner-most level consists of the pivoted index labels and fill missing values by 0. And then nomralizing the artist_region_stream_unstack dataframe using standard Scaler.

In [47]:
```
artist_region_stream_unstack=artist_region_stream.unstack().fillna(0)
```

In [48]:
```
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

artist_region_scaled=scaler.fit_transform(artist_region_stream_unstack)
```

The linear dimensionality needed to be reduced, so we use the decomposition tool to reduce it. In our model, we will use 10 components of PCA and use Singular Value Decomposition of the data to project it to a lower dimensional space.Visualize region_pca dataframe.

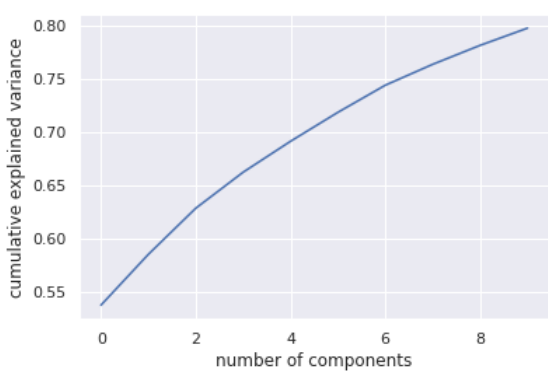
In [49]:
```
# Region Code PCA

from sklearn import decomposition

pca = decomposition.PCA(n_components=10)
region_pca=pca.fit_transform(artist_region_scaled)
```

**ACTION: PCA plot**

Use a figure to show which components of PCA explain the majority of variation in the data. Accordingly, use only those components in your further analysis.

The pca.explained_variance*ratio* parameter returns a vector of the variance explained by each dimension. Therefore the output will be the explained variance ratio by each dimension from the 10 dimensions.

In [50]:
```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```




Here the shape of region_pca dataframe is investigated.

In [51]:
```
pca.explained_variance_ratio_
```

Out[51]:
```
array([0.53696704, 0.04771371, 0.04332677, 0.03385378, 0.02905212,
       0.02722307, 0.02546508, 0.01961776, 0.01781458, 0.01618459])
```

In [52]:
```
region_pca.shape
```

Out[52]:
```
(650, 10)
```

Here we generate a name list for the principle components and create a Dataframe for the region_pca and use the name list created above. Get the artist name in the region_stream_unstack dataframe and Concatenate artist_pca and region_pca dataframe along the columns.

In [53]:
```
pc_name=[]
for i in range(1,11):
    pc_name.append('region_pc'+str(i))
```

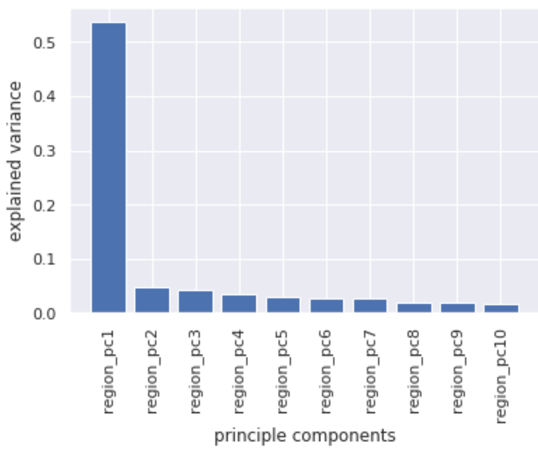
In [54]:
```
region_pcaDF=pd.DataFrame(data=region_pca,columns=pc_name)
```

In [55]:
```
artist_pca=pd.Series(artist_region_stream_unstack.index.values,name='artist_name')
```

In [56]:
```
region_pcaDF=pd.concat([artist_pca,region_pcaDF],axis=1)
```

At last, we can plot the cumunative pca explained variance ratio according to number of components.

In [57]:
```
plt.bar(pc_name,pca.explained_variance_ratio_)
plt.xticks(rotation=90)
plt.xlabel('principle components')
plt.ylabel('explained variance')
plt.show()
```




When we add up the explianed variance ratio of the 10 components, we can know that the total expleined variance ratio will be greater than 90% only when we include all of the 10 components. So we will keep all of the 10 pca components in our further analysis.

In [58]:
```
df_artist_features=pd.merge(df_artist_features,region_pcaDF, on="artist_name",how="outer")
df_artist_features.describe()
```

Out[58]:

| | success | count | number_users | passion_score | weightedscore | malepercentage | Dependent | Senior | YoungAdult | region_pc |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 661.000000 | 661.000000 | 661.000000 | 661.000000 | 471.000000 | 661.000000 | 660.000000 | 660.000000 | 660.000000 | 6.500000e+0 |
| **mean** | 0.125567 | 5757.184569 | 4797.928896 | 1.160953 | 1.238894 | 0.582713 | 0.089986 | 0.061952 | 0.357977 | -2.035978e-1 |
| **std** | 0.331612 | 32638.464044 | 27057.108261 | 0.826511 | 0.268372 | 0.231221 | 0.121536 | 0.137552 | 0.210957 | 1.662607e+0 |
| **min** | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -3.009433e+0 |
| **25%** | 0.000000 | 7.000000 | 6.000000 | 1.000000 | 1.088645 | 0.454545 | 0.000000 | 0.000000 | 0.247930 | -2.999401e+0 |
| **50%** | 0.000000 | 44.000000 | 39.000000 | 1.042735 | 1.185329 | 0.558764 | 0.065214 | 0.024550 | 0.386817 | -2.972168e+0 |
| **75%** | 0.000000 | 499.000000 | 468.000000 | 1.140891 | 1.290773 | 0.719298 | 0.138223 | 0.047982 | 0.479066 | -2.630680e+0 |
| **max** | 1.000000 | 447873.000000 | 367023.000000 | 19.000000 | 3.750000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 2.206560e+0 |

**Data transformation**

The final step is to decide whether or not to normalize/transform any of the features.

We should normalize data if we are more interested in the relative rather than absolute differences between variables. Given that all the numerical features in our dataset (centrality, lift, influence, gender breakdown, age breakdown) were meaningful, i.e. distances did make a difference.

**ACTION: Feature transformation**

Comment on whether transforming particular features (influence, gender breakdown, age breakdown) is useful. Calculate the transformation where necessary.

We have already combined all of our features that we generated above into the dataframe 'df_artist_features' that can be processed by a machine learning algorithm. Here we create a copy of it called 'df_af' and we drop the missing values of total weighted score to 0 here.

In [59]:
```
df_af=df_artist_features.copy()
df_af = df_af.reset_index()
df_af['weightedscore'].fillna(0,inplace=True)
df_af.head()
```

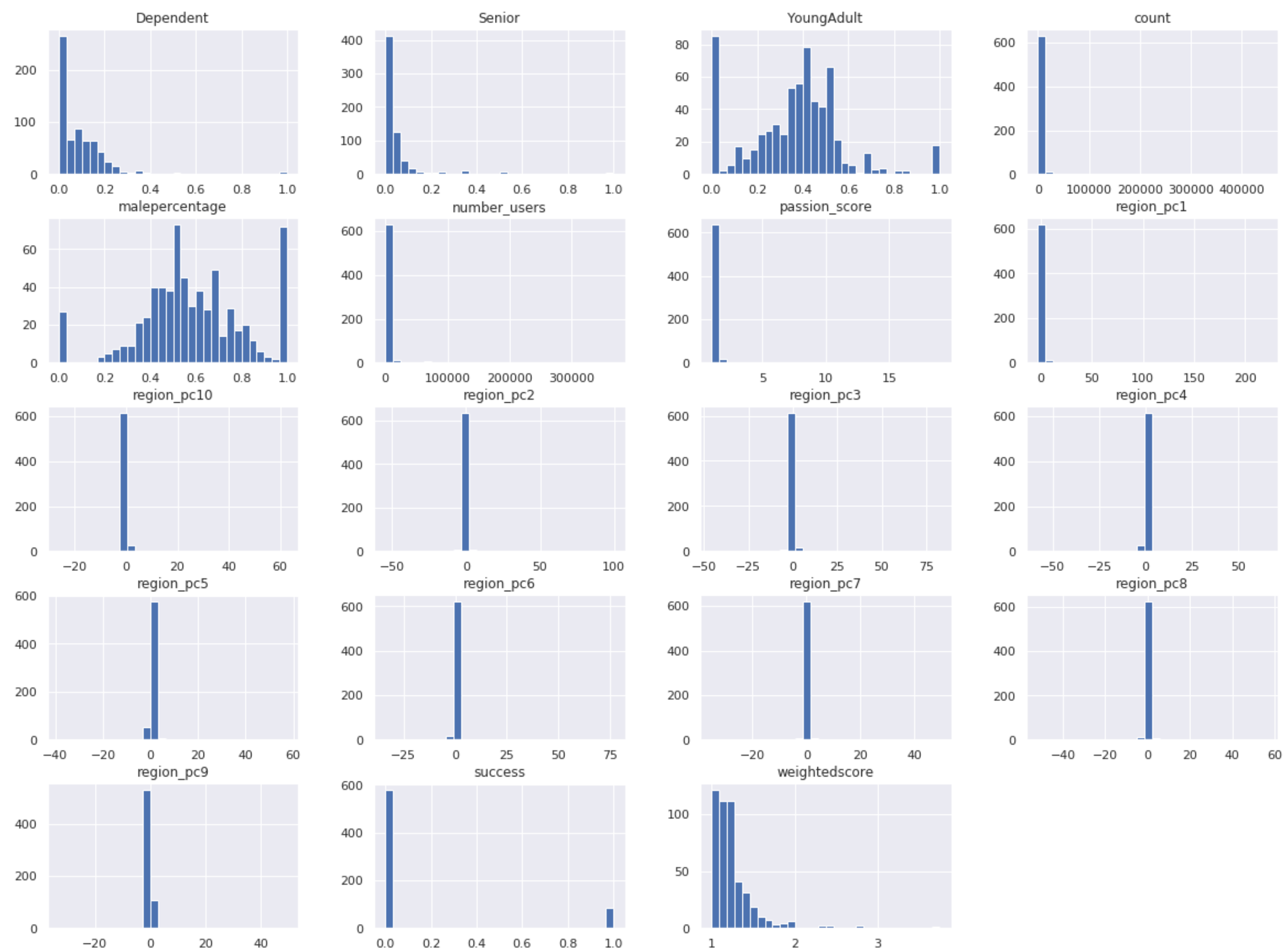
Out[59]:

| | index | artist_name | success | count | number_users | passion_score | weightedscore | malepercentage | Dependent | Senior | ... | region_pc1 | region |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | #90s Update | 0 | 16 | 15 | 1.066667 | 1.183983 | 0.600000 | 0.066667 | 0.000000 | ... | -2.996537 | 0.13 |
| **1** | 1 | 17 Memphis | 0 | 12 | 12 | 1.000000 | 1.062500 | 0.333333 | 0.166667 | 0.083333 | ... | -2.995259 | 0.13 |
| **2** | 2 | 2D | 0 | 1 | 1 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | ... | -3.001679 | 0.13 |
| **3** | 3 | 3JS | 0 | 5 | 4 | 1.250000 | 0.000000 | 0.750000 | 0.000000 | 0.000000 | ... | -2.999616 | 0.13 |
| **4** | 4 | 99 Percent | 0 | 1291 | 1189 | 1.085786 | 1.277956 | 0.315391 | 0.379661 | 0.027966 | ... | -2.421869 | 0.06 |

5 rows × 21 columns

We plot the bar chart of all the features here.

In [60]:
```
%matplotlib inline
import matplotlib.pyplot as plt
df_artist_features.hist(bins=30,figsize=(20,15))
plt.show()
```




By looking at the histogram of all variables, we decide to normalize "stream count", "total number of users" and all the pca region features. We create x and x1, where x is the "stream_count" column's values, x1 is the "total_number_of_users" column's values and x2-x10 are the values of pca 1 to pca 9 columns. We normalize them and create dataframes containing the value after normalized.

In [61]:
```
import numpy as np
from sklearn import preprocessing

x = df_af[["count"]].values
x_scaled=preprocessing.scale(x)
df_nor=pd.DataFrame(x_scaled)
df_nor
```

```
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dty
pe int64 was converted to float64 by the scale function.
  warnings.warn(msg, DataConversionWarning)
```

Out[61]:

| | 0 |
|---|---|
| **0** | -0.176036 |
| **1** | -0.176158 |
| **2** | -0.176496 |
| **3** | -0.176373 |
| **4** | -0.136942 |
| **5** | 0.127149 |
| **6** | 0.230173 |
| **7** | 1.933932 |
| **8** | -0.176434 |
| **9** | -0.176281 |
| **10** | -0.176404 |
| **11** | -0.174778 |
| **12** | -0.175913 |
| **13** | -0.173092 |
| **14** | -0.172295 |
| **15** | -0.175238 |
| **16** | -0.176097 |
| **17** | -0.155952 |
| **18** | -0.176312 |
| **19** | -0.170670 |
| **20** | -0.175698 |
| **21** | -0.176434 |
| **22** | -0.172019 |
| **23** | -0.158620 |
| **24** | -0.174901 |
| **25** | -0.175146 |
| **26** | -0.175882 |
| **27** | 0.143768 |
| **28** | -0.146048 |
| **29** | -0.176281 |
| **...** | ... |
| **631** | -0.167818 |
| **632** | -0.176005 |
| **633** | 0.293275 |
| **634** | -0.117778 |
| **635** | -0.176005 |
| **636** | -0.175821 |
| **637** | -0.176496 |
| **638** | 0.002049 |
| **639** | -0.130932 |
| **640** | -0.166101 |
| **641** | -0.173767 |
| **642** | 0.029951 |
| **643** | -0.176434 |
| **644** | -0.176496 |
| **645** | -0.156075 |
| **646** | -0.152518 |
| **647** | -0.173337 |
| **648** | -0.169229 |
| **649** | -0.174717 |
| **650** | -0.176158 |
| **651** | 0.650303 |

| | 0 |
|---|---|
| **652** | -0.176496 |
| **653** | -0.175760 |
| **654** | -0.176465 |
| **655** | 0.152200 |
| **656** | -0.175913 |
| **657** | 0.595172 |
| **658** | -0.173184 |
| **659** | 4.903629 |
| **660** | -0.176312 |

661 rows × 1 columns

In [62]:
```
x1 = df_af[["number_users"]].values
x1_scaled=preprocessing.scale(x1)
df_nor0=pd.DataFrame(x1_scaled)
df_nor0
```

```
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dty
pe int64 was converted to float64 by the scale function.
  warnings.warn(msg, DataConversionWarning)
```

Out[62]:

| | 0 |
|---|---|
| **0** | -0.176905 |
| **1** | -0.177016 |
| **2** | -0.177423 |
| **3** | -0.177312 |
| **4** | -0.133483 |
| **5** | 0.107819 |
| **6** | 0.235091 |
| **7** | 2.004283 |
| **8** | -0.177349 |
| **9** | -0.177164 |
| **10** | -0.177312 |
| **11** | -0.175463 |
| **12** | -0.176721 |
| **13** | -0.173429 |
| **14** | -0.172430 |
| **15** | -0.175907 |
| **16** | -0.177016 |
| **17** | -0.155009 |
| **18** | -0.177201 |
| **19** | -0.171690 |
| **20** | -0.176462 |
| **21** | -0.177349 |
| **22** | -0.172245 |
| **23** | -0.157044 |
| **24** | -0.175574 |
| **25** | -0.176018 |
| **26** | -0.176795 |
| **27** | 0.161191 |
| **28** | -0.146687 |
| **29** | -0.177312 |
| **...** | ... |
| **631** | -0.166956 |
| **632** | -0.176832 |
| **633** | 0.288019 |
| **634** | -0.118022 |
| **635** | -0.177053 |
| **636** | -0.176610 |
| **637** | -0.177423 |
| **638** | 0.022491 |
| **639** | -0.125864 |
| **640** | -0.165033 |
| **641** | -0.174612 |
| **642** | 0.042538 |
| **643** | -0.177423 |
| **644** | -0.177423 |
| **645** | -0.154011 |
| **646** | -0.154491 |
| **647** | -0.174686 |
| **648** | -0.169101 |
| **649** | -0.175278 |
| **650** | -0.177090 |
| **651** | 0.688661 |

| | 0 |
|---|---|
| 652 | -0.177423 |
| 653 | -0.176795 |
| 654 | -0.177386 |
| 655 | 0.166628 |
| 656 | -0.176721 |
| 657 | 0.514638 |
| 658 | -0.173466 |
| 659 | 5.226615 |
| 660 | -0.177238 |

661 rows × 1 columns

Before normalize the region pca variables, we decide to fill the missing values 'NA' by the mean of the variables.

In [63]:
```
pc=df_af[['region_pc1', 'region_pc2', 'region_pc3', 'region_pc4', 'region_pc5',
       'region_pc6', 'region_pc7', 'region_pc8', 'region_pc9', 'region_pc10']]
pc=pc.apply(lambda x: x.fillna(x.mean()),axis=0)
```

In [64]:
```
x2=pc[['region_pc1']].values
x3=pc[['region_pc2']].values
x4=pc[['region_pc3']].values
x5=pc[['region_pc4']].values
x6=pc[['region_pc5']].values
x7=pc[['region_pc6']].values
x8=pc[['region_pc7']].values
x9=pc[['region_pc8']].values
x10=pc[['region_pc9']].values
x11=pc[['region_pc10']].values

x2_scaled=preprocessing.scale(x2)
x3_scaled=preprocessing.scale(x3)
x4_scaled=preprocessing.scale(x4)
x5_scaled=preprocessing.scale(x5)
x6_scaled=preprocessing.scale(x6)
x7_scaled=preprocessing.scale(x7)
x8_scaled=preprocessing.scale(x8)
x9_scaled=preprocessing.scale(x9)
x10_scaled=preprocessing.scale(x10)
x11_scaled=preprocessing.scale(x11)

df_nor2=pd.DataFrame(x2_scaled)
df_nor3=pd.DataFrame(x3_scaled)
df_nor4=pd.DataFrame(x4_scaled)
df_nor5=pd.DataFrame(x5_scaled)
df_nor6=pd.DataFrame(x6_scaled)
df_nor7=pd.DataFrame(x7_scaled)
df_nor8=pd.DataFrame(x8_scaled)
df_nor9=pd.DataFrame(x9_scaled)
df_nor10=pd.DataFrame(x10_scaled)
df_nor11=pd.DataFrame(x11_scaled)
```

In [65]:
```
df_af['count']=df_nor
df_af['number_users']=df_nor0
df_af['region_pc1']=df_nor2
df_af['region_pc2']=df_nor3
df_af['region_pc3']=df_nor4
df_af['region_pc4']=df_nor5
df_af['region_pc5']=df_nor6
df_af['region_pc6']=df_nor7
df_af['region_pc7']=df_nor8
df_af['region_pc8']=df_nor9
df_af['region_pc9']=df_nor10
df_af['region_pc10']=df_nor11
df_af
```

Out[65]:

| | index | artist_name | success | count | number_users | passion_score | weightedscore | malepercentage | Dependent | Senior | ... | region_pc1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | #90s Update | 0 | -0.176036 | -0.176905 | 1.066667 | 1.183983 | 0.600000 | 0.066667 | 0.000000 | ... | -0.181890 |
| **1** | 1 | 17 Memphis | 0 | -0.176158 | -0.177016 | 1.000000 | 1.062500 | 0.333333 | 0.166667 | 0.083333 | ... | -0.181812 |
| **2** | 2 | 2D | 0 | -0.176496 | -0.177423 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | ... | -0.182202 |
| **3** | 3 | 3JS | 0 | -0.176373 | -0.177312 | 1.250000 | 0.000000 | 0.750000 | 0.000000 | 0.000000 | ... | -0.182077 |
| **4** | 4 | 99 Percent | 0 | -0.136942 | -0.133483 | 1.085786 | 1.277956 | 0.315391 | 0.379661 | 0.027966 | ... | -0.147007 |
| **5** | 5 | A Boogie Wit Da Hoodie | 0 | 0.127149 | 0.107819 | 1.284066 | 1.224615 | 0.712952 | 0.222888 | 0.020725 | ... | 0.019168 |
| **6** | 6 | A Boogie Wit da Hoodie | 1 | 0.230173 | 0.235091 | 1.189170 | 1.272593 | 0.670522 | 0.269546 | 0.025340 | ... | 0.394901 |
| **7** | 7 | A R I Z O N A | 1 | 1.933932 | 2.004283 | 1.166867 | 1.245686 | 0.473392 | 0.140033 | 0.034987 | ... | 2.879146 |
| **8** | 8 | AGWA | 0 | -0.176434 | -0.177349 | 1.000000 | 2.000000 | 1.000000 | 0.000000 | 0.666667 | ... | -0.182138 |
| **9** | 9 | ALMA | 0 | -0.176281 | -0.177164 | 1.000000 | 0.000000 | 0.500000 | 0.000000 | 0.125000 | ... | -0.181863 |
| **10** | 10 | ALP | 0 | -0.176404 | -0.177312 | 1.000000 | 1.000000 | 0.250000 | 0.500000 | 0.000000 | ... | -0.182207 |
| **11** | 11 | AV AV AV | 0 | -0.174778 | -0.175463 | 1.055556 | 0.000000 | 0.796296 | 0.000000 | 0.000000 | ... | -0.181647 |
| **12** | 12 | AVVAH | 0 | -0.175913 | -0.176721 | 1.000000 | 1.092715 | 0.350000 | 0.100000 | 0.000000 | ... | -0.181954 |
| **13** | 13 | AXSHN | 1 | -0.173092 | -0.173429 | 1.027523 | 1.323853 | 0.449541 | 0.214953 | 0.046729 | ... | -0.179053 |
| **14** | 14 | Absofacto | 1 | -0.172295 | -0.172430 | 1.014706 | 1.143588 | 0.588235 | 0.117647 | 0.022059 | ... | -0.176228 |
| **15** | 15 | Adam Sample | 0 | -0.175238 | -0.175907 | 1.000000 | 1.545097 | 0.404762 | 0.119048 | 0.023810 | ... | -0.175237 |
| **16** | 16 | Adan Carmona | 0 | -0.176097 | -0.177016 | 1.166667 | 2.767196 | 0.416667 | 0.000000 | 0.000000 | ... | -0.182151 |
| **17** | 17 | Adia Victoria | 0 | -0.155952 | -0.155009 | 1.105437 | 1.050996 | 0.635914 | 0.044702 | 0.082781 | ... | -0.168537 |
| **18** | 18 | Alcatraz | 0 | -0.176312 | -0.177201 | 1.000000 | 0.000000 | 0.571429 | 0.000000 | 0.142857 | ... | -0.181977 |
| **19** | 19 | Alessio Bernabei | 0 | -0.170670 | -0.171690 | 1.224359 | 1.114286 | 0.506410 | 0.089744 | 0.025641 | ... | -0.172524 |
| **20** | 20 | Alex Hoyer | 0 | -0.175698 | -0.176462 | 1.000000 | 1.153571 | 0.407407 | 0.148148 | 0.037037 | ... | -0.181371 |
| **21** | 21 | Alex Roy | 0 | -0.176434 | -0.177349 | 1.000000 | 2.767196 | 0.000000 | 0.000000 | 0.000000 | ... | -0.182184 |
| **22** | 22 | Alexander Brown | 0 | -0.172019 | -0.172245 | 1.042553 | 1.084948 | 0.631206 | 0.028369 | 0.042553 | ... | -0.180284 |
| **23** | 23 | Alexander Cardinale | 0 | -0.158620 | -0.157044 | 1.057971 | 1.100414 | 0.394928 | 0.105647 | 0.060109 | ... | -0.167502 |
| **24** | 24 | Alexander Charles | 0 | -0.174901 | -0.175574 | 1.039216 | 1.261209 | 0.823529 | 0.137255 | 0.000000 | ... | -0.181089 |
| **25** | 25 | Alice | 0 | -0.175146 | -0.176018 | 1.153846 | 1.210582 | 0.692308 | 0.102564 | 0.076923 | ... | -0.181159 |
| **26** | 26 | Aliose | 0 | -0.175882 | -0.176795 | 1.166667 | 1.333333 | 0.500000 | 0.000000 | 0.055556 | ... | -0.181810 |
| **27** | 27 | All Tvvins | 1 | 0.143768 | 0.161191 | 1.140891 | 1.115551 | 0.666776 | 0.115156 | 0.030136 | ... | 0.052643 |
| **28** | 28 | Alma | 0 | -0.146048 | -0.146687 | 1.194712 | 1.110234 | 0.643029 | 0.153012 | 0.038554 | ... | -0.079220 |
| **29** | 29 | Amaro Ferreiro | 0 | -0.176281 | -0.177312 | 2.000000 | 0.000000 | 0.750000 | 0.000000 | 0.000000 | ... | -0.182217 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **631** | 631 | Wild Youth | 1 | -0.167818 | -0.166956 | 1.000000 | 1.122784 | 0.485915 | 0.031690 | 0.140845 | ... | -0.156552 |
| **632** | 632 | Wildling | 0 | -0.176005 | -0.176832 | 1.000000 | 0.000000 | 0.764706 | 0.058824 | 0.117647 | ... | -0.182000 |
| **633** | 633 | Will Joseph Cook | 1 | 0.293275 | 0.288019 | 1.217481 | 1.343154 | 0.469130 | 0.148583 | 0.035848 | ... | 0.269432 |
| **634** | 634 | Willy William | 0 | -0.117778 | -0.118022 | 1.192284 | 1.209903 | 0.500311 | 0.130707 | 0.028143 | ... | -0.098638 |
| **635** | 635 | Witek Muzyk Ulicy | 0 | -0.176005 | -0.177053 | 1.545455 | 0.000000 | 0.909091 | 0.000000 | 0.000000 | ... | -0.181888 |
| **636** | 636 | Wudstik | 0 | -0.175821 | -0.176610 | 1.000000 | 0.000000 | 0.565217 | 0.000000 | 0.043478 | ... | -0.181758 |
| **637** | 637 | Xavier Cugat y su orquesta | 0 | -0.176496 | -0.177423 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | -0.182219 |
| **638** | 638 | Xavier Dunn | 1 | 0.002049 | 0.022491 | 1.077321 | 1.054221 | 0.497595 | 0.070619 | 0.048318 | ... | -0.047847 |
| **639** | 639 | YFN Lucci | 0 | -0.130932 | -0.125864 | 1.065950 | 1.200079 | 0.774194 | 0.205904 | 0.017279 | ... | -0.132275 |
| **640** | 640 | YONAKA | 0 | -0.166101 | -0.165033 | 1.011905 | 1.068039 | 0.642857 | 0.089552 | 0.026866 | ... | -0.163413 |
| **641** | 641 | Yasutaka Nakata | 0 | -0.173767 | -0.174612 | 1.168831 | 1.114286 | 0.688312 | 0.092105 | 0.013158 | ... | -0.180006 |

| | index | artist_name | success | count | number_users | passion_score | weightedscore | malepercentage | Dependent | Senior | ... | region_pc1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **642** | 642 | Yellow Claw | 1 | 0.029951 | 0.042538 | 1.132145 | 1.173007 | 0.690484 | 0.113360 | 0.023279 | ... | 0.151826 |
| **643** | 643 | Ylric Illians | 0 | -0.176434 | -0.177423 | 3.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | ... | -0.182150 |
| **644** | 644 | Young F | 0 | -0.176496 | -0.177423 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | ... | -0.182200 |
| **645** | 645 | Young Spray | 0 | -0.156075 | -0.154011 | 1.052050 | 1.028498 | 0.799685 | 0.141046 | 0.020602 | ... | -0.168229 |
| **646** | 646 | YoungBoy Never Broke Again | 0 | -0.152518 | -0.154491 | 1.260870 | 1.378724 | 0.805153 | 0.197411 | 0.019417 | ... | -0.158712 |
| **647** | 647 | Youngboy Never Broke Again | 0 | -0.173337 | -0.174686 | 1.386667 | 1.335477 | 0.826667 | 0.246575 | 0.013699 | ... | -0.178787 |
| **648** | 648 | Yvng Swag | 0 | -0.169229 | -0.169101 | 1.053097 | 1.497963 | 0.539823 | 0.351351 | 0.009009 | ... | -0.172383 |
| **649** | 649 | Zac Brown | 0 | -0.174717 | -0.175278 | 1.000000 | 1.229198 | 0.559322 | 0.050847 | 0.084746 | ... | -0.180501 |
| **650** | 650 | Zak & Diego | 0 | -0.176158 | -0.177090 | 1.200000 | 0.000000 | 0.500000 | 0.000000 | 0.000000 | ... | -0.182115 |
| **651** | 651 | Zak Abel | 1 | 0.650303 | 0.688661 | 1.151557 | 1.222618 | 0.465559 | 0.099047 | 0.039911 | ... | 0.550193 |
| **652** | 652 | Zakopower | 0 | -0.176496 | -0.177423 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | ... | -0.182219 |
| **653** | 653 | Zarcort | 0 | -0.175760 | -0.176795 | 1.388889 | 1.875000 | 0.777778 | 0.166667 | 0.055556 | ... | -0.173227 |
| **654** | 654 | Zbigniew Kurtycz | 0 | -0.176465 | -0.177386 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | ... | -0.182187 |
| **655** | 655 | Zion & Lennox | 1 | 0.152200 | 0.166628 | 1.152424 | 1.265757 | 0.450715 | 0.074226 | 0.024814 | ... | 0.578635 |
| **656** | 656 | birthday | 0 | -0.175913 | -0.176721 | 1.000000 | 1.000000 | 0.400000 | 0.250000 | 0.000000 | ... | -0.181827 |
| **657** | 657 | dvsn | 1 | 0.595172 | 0.514638 | 1.345019 | 1.128136 | 0.452170 | 0.137550 | 0.023677 | ... | 0.469397 |
| **658** | 658 | flor | 1 | -0.173184 | -0.173466 | 1.009259 | 1.160215 | 0.527778 | 0.157407 | 0.037037 | ... | -0.179915 |
| **659** | 659 | gnash | 1 | 4.903629 | 5.226615 | 1.133976 | 1.260124 | 0.400450 | 0.231351 | 0.029931 | ... | 4.020854 |
| **660** | 660 | livetune+ | 0 | -0.176312 | -0.177238 | 1.166667 | 1.000000 | 0.500000 | 0.333333 | 0.000000 | ... | -0.182148 |

661 rows × 21 columns

As the dataframe above presented, the values after normalized of stream count, number of users and all the pca variables have been merged into it.

In terms to improve accuracy of the model, we still decide to normalize all the other variables except the dependent variable 'success'.The new values will be added into the final dataframe 'df_af'.

In [66]:
```
df_af=df_af.dropna(subset=['Dependent','Senior','YoungAdult'])
df_af
xp=df_af[['passion_score']].values
xw=df_af[['weightedscore']].values
xm=df_af[['malepercentage']].values
xd=df_af[['Dependent']].values
xs=df_af[['Senior']].values
xy=df_af[['YoungAdult']].values

xp_scaled=preprocessing.scale(xp)
xw_scaled=preprocessing.scale(xw)
xm_scaled=preprocessing.scale(xm)
xd_scaled=preprocessing.scale(xd)
xs_scaled=preprocessing.scale(xs)
xy_scaled=preprocessing.scale(xy)
```

In [67]:
```
dfnormalp=pd.DataFrame(xp_scaled)
dfnormalw=pd.DataFrame(xw_scaled)
dfnormalm=pd.DataFrame(xm_scaled)
dfnormald=pd.DataFrame(xd_scaled)
dfnormals=pd.DataFrame(xs_scaled)
dfnormaly=pd.DataFrame(xy_scaled)

df_af['passion_score']=dfnormalp
df_af['weightedscore']=dfnormalw
df_af['malepercentage']=dfnormalm
df_af['Dependent']=dfnormald
df_af['Senior']=dfnormals
df_af['YoungAdult']=dfnormaly
```

```
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/ipykernel/__main__.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/ipykernel/__main__.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/ipykernel/__main__.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/ipykernel/__main__.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/ipykernel/__main__.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/ipykernel/__main__.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

In [68]:
```
df_af.head()
```

Out[68]:

| | index | artist_name | success | count | number_users | passion_score | weightedscore | malepercentage | Dependent | Senior | ... | region_pc1 | re |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | #90s Update | 0 | -0.176036 | -0.176905 | -0.114376 | 0.496370 | 0.077692 | -0.192020 | -0.450729 | ... | -0.181890 | |
| **1** | 1 | 17 Memphis | 0 | -0.176158 | -0.177016 | -0.195039 | 0.295276 | -1.078466 | 0.631405 | 0.155563 | ... | -0.181812 | |
| **2** | 2 | 2D | 0 | -0.176496 | -0.177423 | -0.195039 | -1.463501 | 1.811929 | 7.493283 | -0.450729 | ... | -0.182202 | |
| **3** | 3 | 3JS | 0 | -0.176373 | -0.177312 | 0.107446 | -1.463501 | 0.728031 | -0.740970 | -0.450729 | ... | -0.182077 | |
| **4** | 4 | 99 Percent | 0 | -0.136942 | -0.133483 | -0.091243 | 0.651924 | -1.156256 | 2.385255 | -0.247261 | ... | -0.147007 | |

5 rows × 21 columns

**Preprocessing**

Before we can run any models on our dataset, we must make sure it is prepared and cleaned to avoid errors in results. This stage is generally refered to as preprocessing.

To begin with, we need to deal with missing data in the dataframe - the ML algorithm will not be able to process NaN or missing values.

For this study, we will be imputing missing numerical values, and filling any one which we were not able to imput, with 0.

We have already fill the missing values of PCA variables by their mean. Here we just drop the rest missing values of other variables.

In [69]:
```
final_df = df_af
final_df=final_df.dropna()
final_df.describe()
```

Out[69]:

| | index | success | count | number_users | passion_score | weightedscore | malepercentage | Dependent | Senior | YoungAdult | reg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 65 |
| **mean** | 329.757208 | 0.125948 | 0.000535 | 0.000538 | -0.001999 | 0.002221 | 0.000540 | 0.001124 | -0.002076 | -0.001022 | ( |
| **std** | 190.700083 | 0.332043 | 1.002230 | 1.002229 | 1.000199 | 0.999890 | 1.001422 | 1.001101 | 1.000095 | 1.001174 | 1 |
| **min** | 0.000000 | 0.000000 | -0.176496 | -0.177423 | -0.195039 | -1.463501 | -2.523663 | -0.740970 | -0.450729 | -1.698210 | -( |
| **25%** | 165.500000 | 0.000000 | -0.176312 | -0.177238 | -0.195039 | -1.463501 | -0.552939 | -0.740970 | -0.450729 | -0.531870 | -( |
| **50%** | 330.000000 | 0.000000 | -0.175146 | -0.176018 | -0.143332 | 0.379213 | -0.101091 | -0.201019 | -0.272408 | 0.135481 | -( |
| **75%** | 494.500000 | 0.000000 | -0.161226 | -0.160095 | -0.024730 | 0.588245 | 0.594175 | 0.398058 | -0.103065 | 0.572346 | -( |
| **max** | 659.000000 | 1.000000 | 13.556110 | 13.397566 | 21.583878 | 4.743947 | 1.811929 | 7.493283 | 6.824777 | 3.045692 | 13 |

Next, we need to make sure that none of the variables going into the model are collinear, and if so, we need to remove those variables that are highly correlated.

**ACTION: Multi-collinearity**

Check and deal with multi-collinearity in your feature set.

We check for multicollinearity by using the corr() function and sort the value from highest to lowest. From the correlation,it can be noticed that there are high collinearity between stream count and number of users and region_pca1 since the values are higher than 90%.

In [70]:
```
# Check for multicollinearity
# Check for highly correlated variables (>90%)
features=['count',"number_users","passion_score","malepercentage","weightedscore","Dependent","Senior","YoungAdult",'region_pc1', 'region_pc2', 'region_pc3', 'region_pc4', 'region_pc5','region_pc6', 'region_pc7', 'region_pc8', 'region_pc9', 'region_pc10']
c = final_df[features].corr().abs()
s = c.unstack()
corr = sorted(s.items(),key = lambda x: x[1],  reverse=True)
corr = [corr[x] for x in range(len(corr)) if corr[x][1]!=1]
corr
```

Out[70]:
```
[(('count', 'number_users'), 0.996105303736572),
 (('number_users', 'count'), 0.996105303736572),
 (('count', 'region_pc1'), 0.9898737867659627),
 (('region_pc1', 'count'), 0.9898737867659627),
 (('number_users', 'region_pc1'), 0.9873308518917375),
 (('region_pc1', 'number_users'), 0.9873308518917375),
 (('Senior', 'YoungAdult'), 0.2900948562835749),
 (('YoungAdult', 'Senior'), 0.2900948562835749),
 (('malepercentage', 'weightedscore'), 0.26453641954549817),
 (('weightedscore', 'malepercentage'), 0.26453641954549817),
 (('weightedscore', 'YoungAdult'), 0.24551888138250416),
 (('YoungAdult', 'weightedscore'), 0.24551888138250416),
 (('malepercentage', 'Dependent'), 0.1940672876063316),
 (('Dependent', 'malepercentage'), 0.1940672876063316),
 (('malepercentage', 'Senior'), 0.19224178410800338),
 (('Senior', 'malepercentage'), 0.19224178410800338),
 (('Dependent', 'Senior'), 0.18306656522873146),
 (('Senior', 'Dependent'), 0.18306656522873146),
 (('weightedscore', 'Senior'), 0.17368201581352172),
 (('Senior', 'weightedscore'), 0.17368201581352172),
 (('weightedscore', 'Dependent'), 0.15621261849548154),
 (('Dependent', 'weightedscore'), 0.15621261849548154),
 (('number_users', 'Dependent'), 0.11213086543677231),
 (('Dependent', 'number_users'), 0.11213086543677231),
 (('malepercentage', 'YoungAdult'), 0.1114751232025845),
 (('YoungAdult', 'malepercentage'), 0.1114751232025845),
 (('number_users', 'region_pc3'), 0.10957352045088477),
 (('region_pc3', 'number_users'), 0.10957352045088477),
 (('count', 'region_pc3'), 0.10902012311526688),
 (('region_pc3', 'count'), 0.10902012311526688),
 (('count', 'Dependent'), 0.10895342847840789),
 (('Dependent', 'count'), 0.10895342847840789),
 (('passion_score', 'malepercentage'), 0.10360014603062476),
 (('malepercentage', 'passion_score'), 0.10360014603062476),
 (('Dependent', 'region_pc1'), 0.10028118899657193),
 (('region_pc1', 'Dependent'), 0.10028118899657193),
 (('passion_score', 'weightedscore'), 0.0975993741394282),
 (('weightedscore', 'passion_score'), 0.0975993741394282),
 (('passion_score', 'YoungAdult'), 0.09690902730132399),
 (('YoungAdult', 'passion_score'), 0.09690902730132399),
 (('weightedscore', 'region_pc10'), 0.0687990323717114),
 (('region_pc10', 'weightedscore'), 0.0687990323717114),
 (('malepercentage', 'region_pc1'), 0.06455253149903238),
 (('region_pc1', 'malepercentage'), 0.06455253149903238),
 (('Dependent', 'region_pc3'), 0.06408214210887761),
 (('region_pc3', 'Dependent'), 0.06408214210887761),
 (('number_users', 'malepercentage'), 0.0636047636563406),
 (('malepercentage', 'number_users'), 0.0636047636563406),
 (('count', 'malepercentage'), 0.060545756616547),
 (('malepercentage', 'count'), 0.060545756616547),
 (('number_users', 'weightedscore'), 0.06033980308806775),
 (('weightedscore', 'number_users'), 0.06033980308806775),
 (('malepercentage', 'region_pc7'), 0.05774476674353545),
 (('region_pc7', 'malepercentage'), 0.05774476674353545),
 (('count', 'weightedscore'), 0.056937283818603544),
 (('weightedscore', 'count'), 0.056937283818603544),
 (('weightedscore', 'region_pc1'), 0.056445049551071505),
 (('region_pc1', 'weightedscore'), 0.056445049551071505),
 (('count', 'region_pc5'), 0.04880734305689876),
 (('region_pc5', 'count'), 0.04880734305689876),
 (('passion_score', 'Dependent'), 0.045957677884510974),
 (('Dependent', 'passion_score'), 0.045957677884510974),
 (('YoungAdult', 'region_pc10'), 0.044651042285670905),
 (('region_pc10', 'YoungAdult'), 0.044651042285670905),
 (('YoungAdult', 'region_pc8'), 0.038532889700918935),
 (('region_pc8', 'YoungAdult'), 0.038532889700918935),
 (('malepercentage', 'region_pc8'), 0.038409406275909504),
 (('region_pc8', 'malepercentage'), 0.038409406275909504),
 (('Dependent', 'region_pc4'), 0.038174282112028994),
 (('region_pc4', 'Dependent'), 0.038174282112028994),
 (('Dependent', 'region_pc8'), 0.03329161019137374),
 (('region_pc8', 'Dependent'), 0.03329161019137374),
 (('count', 'region_pc7'), 0.03134405782914076),
 (('region_pc7', 'count'), 0.03134405782914076),
 (('number_users', 'region_pc4'), 0.03071636858597516),
 (('region_pc4', 'number_users'), 0.03071636858597516),
 (('Dependent', 'region_pc9'), 0.029701234307331917),
 (('region_pc9', 'Dependent'), 0.029701234307331917),
 (('malepercentage', 'region_pc4'), 0.02951811129583706),
 (('region_pc4', 'malepercentage'), 0.02951811129583706),
 (('number_users', 'region_pc10'), 0.02868690893762176),
```

```
(('region_pc10', 'number_users'), 0.02868690893762176),
(('malepercentage', 'region_pc10'), 0.028305190598450932),
(('region_pc10', 'malepercentage'), 0.028305190598450932),
(('number_users', 'region_pc5'), 0.02814543419312314),
(('region_pc5', 'number_users'), 0.02814543419312314),
(('Dependent', 'YoungAdult'), 0.027891528463898602),
(('YoungAdult', 'Dependent'), 0.027891528463898602),
(('malepercentage', 'region_pc5'), 0.02747630067346027),
(('region_pc5', 'malepercentage'), 0.02747630067346027),
(('weightedscore', 'region_pc8'), 0.02521810640738936),
(('region_pc8', 'weightedscore'), 0.02521810640738936),
(('weightedscore', 'region_pc6'), 0.024558204965050092),
(('region_pc6', 'weightedscore'), 0.024558204965050092),
(('YoungAdult', 'region_pc7'), 0.02381356791507353),
(('region_pc7', 'YoungAdult'), 0.02381356791507353),
(('YoungAdult', 'region_pc9'), 0.02274344833197705),
(('region_pc9', 'YoungAdult'), 0.02274344833197705),
(('passion_score', 'Senior'), 0.02242663112590295),
(('Senior', 'passion_score'), 0.02242663112590295),
(('Dependent', 'region_pc5'), 0.020459814513901803),
(('region_pc5', 'Dependent'), 0.020459814513901803),
(('weightedscore', 'region_pc9'), 0.019850383202387415),
(('region_pc9', 'weightedscore'), 0.019850383202387415),
(('count', 'region_pc10'), 0.019321588509432823),
(('region_pc10', 'count'), 0.019321588509432823),
(('count', 'region_pc4'), 0.01889190112438198),
(('region_pc4', 'count'), 0.01889190112438198),
(('YoungAdult', 'region_pc5'), 0.018058259382282872),
(('region_pc5', 'YoungAdult'), 0.018058259382282872),
(('Senior', 'region_pc1'), 0.017931846936521525),
(('region_pc1', 'Senior'), 0.017931846936521525),
(('Senior', 'region_pc7'), 0.017088249454834),
(('region_pc7', 'Senior'), 0.017088249454834),
(('Dependent', 'region_pc2'), 0.01704282704383673),
(('region_pc2', 'Dependent'), 0.01704282704383673),
(('count', 'region_pc2'), 0.01690725942879011),
(('region_pc2', 'count'), 0.01690725942879011),
(('count', 'region_pc6'), 0.01656477417907123),
(('region_pc6', 'count'), 0.01656477417907123),
(('weightedscore', 'region_pc3'), 0.016402473657499292),
(('region_pc3', 'weightedscore'), 0.016402473657499292),
(('Senior', 'region_pc3'), 0.01605160141415977),
(('region_pc3', 'Senior'), 0.01605160141415977),
(('count', 'region_pc9'), 0.01586854393557983),
(('region_pc9', 'count'), 0.01586854393557983),
(('Senior', 'region_pc8'), 0.01424578298731101),
(('region_pc8', 'Senior'), 0.01424578298731101),
(('YoungAdult', 'region_pc6'), 0.013900305193216557),
(('region_pc6', 'YoungAdult'), 0.013900305193216557),
(('count', 'region_pc8'), 0.013391644760323685),
(('region_pc8', 'count'), 0.013391644760323685),
(('malepercentage', 'region_pc9'), 0.012967810359164327),
(('region_pc9', 'malepercentage'), 0.012967810359164327),
(('Dependent', 'region_pc10'), 0.011259353320483731),
(('region_pc10', 'Dependent'), 0.011259353320483731),
(('count', 'Senior'), 0.010930029543369604),
(('Senior', 'count'), 0.010930029543369604),
(('malepercentage', 'region_pc2'), 0.010319525933010616),
(('region_pc2', 'malepercentage'), 0.010319525933010616),
(('number_users', 'passion_score'), 0.00937122537555112),
(('passion_score', 'number_users'), 0.00937122537555112),
(('YoungAdult', 'region_pc1'), 0.009309475329385098),
(('region_pc1', 'YoungAdult'), 0.009309475329385098),
(('count', 'passion_score'), 0.009178702589767978),
(('passion_score', 'count'), 0.009178702589767978),
(('Senior', 'region_pc2'), 0.008894234785797261),
(('region_pc2', 'Senior'), 0.008894234785797261),
(('passion_score', 'region_pc1'), 0.00874202870956091),
(('region_pc1', 'passion_score'), 0.00874202870956091),
(('number_users', 'region_pc7'), 0.0086121536703273),
(('region_pc7', 'number_users'), 0.0086121536703273),
(('number_users', 'region_pc8'), 0.008096146719919197),
(('region_pc8', 'number_users'), 0.008096146719919197),
(('number_users', 'Senior'), 0.007944822873582794),
(('Senior', 'number_users'), 0.007944822873582794),
(('YoungAdult', 'region_pc2'), 0.007873931165980676),
(('region_pc2', 'YoungAdult'), 0.007873931165980676),
(('count', 'YoungAdult'), 0.007827231763698867),
(('YoungAdult', 'count'), 0.007827231763698867),
(('number_users', 'region_pc2'), 0.007291587144133633),
(('region_pc2', 'number_users'), 0.007291587144133633),
(('weightedscore', 'region_pc5'), 0.0068329641631396055),
```

```
(('region_pc5', 'weightedscore'), 0.0068329641631396055),
(('weightedscore', 'region_pc7'), 0.00667622174568678),
(('region_pc7', 'weightedscore'), 0.00667622174568678),
(('weightedscore', 'region_pc4'), 0.006511622812787667),
(('region_pc4', 'weightedscore'), 0.006511622812787667),
(('Senior', 'region_pc9'), 0.006378462772257145),
(('region_pc9', 'Senior'), 0.006378462772257145),
(('passion_score', 'region_pc2'), 0.006045862770993209),
(('region_pc2', 'passion_score'), 0.006045862770993209),
(('passion_score', 'region_pc5'), 0.005795341585632269),
(('region_pc5', 'passion_score'), 0.005795341585632269),
(('number_users', 'YoungAdult'), 0.00569591000908643),
(('YoungAdult', 'number_users'), 0.00569591000908643),
(('YoungAdult', 'region_pc3'), 0.005549245668271518),
(('region_pc3', 'YoungAdult'), 0.005549245668271518),
(('number_users', 'region_pc6'), 0.005536353103217127),
(('region_pc6', 'number_users'), 0.005536353103217127),
(('YoungAdult', 'region_pc4'), 0.005344963334627365),
(('region_pc4', 'YoungAdult'), 0.005344963334627365),
(('passion_score', 'region_pc9'), 0.005054481114466202),
(('region_pc9', 'passion_score'), 0.005054481114466202),
(('number_users', 'region_pc9'), 0.004805791046997966),
(('region_pc9', 'number_users'), 0.004805791046997966),
(('Senior', 'region_pc5'), 0.004206478571370755),
(('region_pc5', 'Senior'), 0.004206478571370755),
(('malepercentage', 'region_pc6'), 0.003539637872845347),
(('region_pc6', 'malepercentage'), 0.003539637872845347),
(('passion_score', 'region_pc10'), 0.003513640314324124),
(('region_pc10', 'passion_score'), 0.003513640314324124),
(('passion_score', 'region_pc8'), 0.0034151041986756874),
(('region_pc8', 'passion_score'), 0.0034151041986756874),
(('malepercentage', 'region_pc3'), 0.0027958404078685917),
(('region_pc3', 'malepercentage'), 0.0027958404078685917),
(('Senior', 'region_pc6'), 0.0025571594416695063),
(('region_pc6', 'Senior'), 0.0025571594416695063),
(('Senior', 'region_pc10'), 0.002306930790496077),
(('region_pc10', 'Senior'), 0.002306930790496077),
(('passion_score', 'region_pc3'), 0.0019001827793416709),
(('region_pc3', 'passion_score'), 0.0019001827793416709),
(('weightedscore', 'region_pc2'), 0.001726979078403338),
(('region_pc2', 'weightedscore'), 0.001726979078403338),
(('passion_score', 'region_pc4'), 0.0015538734608392811),
(('region_pc4', 'passion_score'), 0.0015538734608392811),
(('passion_score', 'region_pc7'), 0.0013583796983024363),
(('region_pc7', 'passion_score'), 0.0013583796983024363),
(('Senior', 'region_pc4'), 0.0013290005529663302),
(('region_pc4', 'Senior'), 0.0013290005529663302),
(('Dependent', 'region_pc6'), 0.0013227951724981608),
(('region_pc6', 'Dependent'), 0.0013227951724981608),
(('passion_score', 'region_pc6'), 0.0010049147065156823),
(('region_pc6', 'passion_score'), 0.0010049147065156823),
(('Dependent', 'region_pc7'), 0.00017310709530611208),
(('region_pc7', 'Dependent'), 0.00017310709530611208),
(('region_pc1', 'region_pc7'), 3.193684451216419e-05),
(('region_pc7', 'region_pc1'), 3.193684451216419e-05),
(('region_pc1', 'region_pc3'), 2.60468428692369e-05),
(('region_pc3', 'region_pc1'), 2.60468428692369e-05),
(('region_pc1', 'region_pc5'), 1.956024466359322e-05),
(('region_pc5', 'region_pc1'), 1.956024466359322e-05),
(('region_pc1', 'region_pc2'), 1.5056051943031464e-05),
(('region_pc2', 'region_pc1'), 1.5056051943031464e-05),
(('region_pc1', 'region_pc6'), 1.1023181009390485e-05),
(('region_pc6', 'region_pc1'), 1.1023181009390485e-05),
(('region_pc3', 'region_pc7'), 8.258409637082782e-06),
(('region_pc7', 'region_pc3'), 8.258409637082782e-06),
(('region_pc5', 'region_pc7'), 6.201768652620103e-06),
(('region_pc7', 'region_pc5'), 6.201768652620103e-06),
(('region_pc3', 'region_pc5'), 5.057998462895076e-06),
(('region_pc5', 'region_pc3'), 5.057998462895076e-06),
(('region_pc2', 'region_pc7'), 4.773669883373147e-06),
(('region_pc7', 'region_pc2'), 4.773669883373147e-06),
(('region_pc1', 'region_pc9'), 4.065181064247601e-06),
(('region_pc9', 'region_pc1'), 4.065181064247601e-06),
(('region_pc2', 'region_pc3'), 3.8932788351207454e-06),
(('region_pc3', 'region_pc2'), 3.8932788351207454e-06),
(('region_pc6', 'region_pc7'), 3.495009127193439e-06),
(('region_pc7', 'region_pc6'), 3.495009127193439e-06),
(('region_pc2', 'region_pc5'), 2.9237124767641364e-06),
(('region_pc5', 'region_pc2'), 2.9237124767641364e-06),
(('region_pc3', 'region_pc6'), 2.8504433510898885e-06),
(('region_pc6', 'region_pc3'), 2.8504433510898885e-06),
(('region_pc5', 'region_pc6'), 2.1405756499019757e-06),
```

```
 (('region_pc6', 'region_pc5'), 2.1405756499019757e-06),
 (('region_pc2', 'region_pc6'), 1.6476592164781537e-06),
 (('region_pc6', 'region_pc2'), 1.6476592164781537e-06),
 (('region_pc7', 'region_pc9'), 1.2889050413090252e-06),
 (('region_pc9', 'region_pc7'), 1.2889050413090252e-06),
 (('region_pc1', 'region_pc10'), 1.1697373762716279e-06),
 (('region_pc10', 'region_pc1'), 1.1697373762716279e-06),
 (('region_pc3', 'region_pc9'), 1.051190766592958e-06),
 (('region_pc9', 'region_pc3'), 1.051190766592958e-06),
 (('region_pc5', 'region_pc9'), 7.894111039687379e-07),
 (('region_pc9', 'region_pc5'), 7.894111039687379e-07),
 (('region_pc1', 'region_pc8'), 6.257317598721048e-07),
 (('region_pc8', 'region_pc1'), 6.257317598721048e-07),
 (('region_pc2', 'region_pc9'), 6.076312177482547e-07),
 (('region_pc9', 'region_pc2'), 6.076312177482547e-07),
 (('region_pc6', 'region_pc9'), 4.448680977911559e-07),
 (('region_pc9', 'region_pc6'), 4.448680977911559e-07),
 (('region_pc7', 'region_pc10'), 3.708769665948633e-07),
 (('region_pc10', 'region_pc7'), 3.708769665948633e-07),
 (('region_pc3', 'region_pc10'), 3.0247895578159924e-07),
 (('region_pc10', 'region_pc3'), 3.0247895578159924e-07),
 (('region_pc5', 'region_pc10'), 2.2714967191255054e-07),
 (('region_pc10', 'region_pc5'), 2.2714967191255054e-07),
 (('region_pc7', 'region_pc8'), 1.9839423055449377e-07),
 (('region_pc8', 'region_pc7'), 1.9839423055449377e-07),
 (('region_pc2', 'region_pc10'), 1.748432670450361e-07),
 (('region_pc10', 'region_pc2'), 1.748432670450361e-07),
 (('region_pc3', 'region_pc8'), 1.6180344827617363e-07),
 (('region_pc8', 'region_pc3'), 1.6180344827617363e-07),
 (('region_pc6', 'region_pc10'), 1.2801148020448216e-07),
 (('region_pc10', 'region_pc6'), 1.2801148020448216e-07),
 (('region_pc5', 'region_pc8'), 1.2150981859132372e-07),
 (('region_pc8', 'region_pc5'), 1.2150981859132372e-07),
 (('region_pc2', 'region_pc8'), 9.352941913166267e-08),
 (('region_pc8', 'region_pc2'), 9.352941913166267e-08),
 (('region_pc6', 'region_pc8'), 6.847560590492236e-08),
 (('region_pc8', 'region_pc6'), 6.847560590492236e-08),
 (('region_pc9', 'region_pc10'), 4.720705957879839e-08),
 (('region_pc10', 'region_pc9'), 4.720705957879839e-08),
 (('region_pc8', 'region_pc9'), 2.52544627783188e-08),
 (('region_pc9', 'region_pc8'), 2.52544627783188e-08),
 (('region_pc1', 'region_pc4'), 1.873900781895363e-08),
 (('region_pc4', 'region_pc1'), 1.873900781895363e-08),
 (('region_pc8', 'region_pc10'), 7.266197752399767e-09),
 (('region_pc10', 'region_pc8'), 7.266197752399767e-09),
 (('region_pc4', 'region_pc7'), 5.942120958912061e-09),
 (('region_pc7', 'region_pc4'), 5.942120958912061e-09),
 (('region_pc3', 'region_pc4'), 4.8522421476040205e-09),
 (('region_pc4', 'region_pc3'), 4.8522421476040205e-09),
 (('region_pc4', 'region_pc5'), 3.6392944584245098e-09),
 (('region_pc5', 'region_pc4'), 3.6392944584245098e-09),
 (('region_pc2', 'region_pc4'), 2.801228203917875e-09),
 (('region_pc4', 'region_pc2'), 2.801228203917875e-09),
 (('region_pc4', 'region_pc6'), 2.0556818239044856e-09),
 (('region_pc6', 'region_pc4'), 2.0556818239044856e-09),
 (('region_pc4', 'region_pc9'), 7.517937617539371e-10),
 (('region_pc9', 'region_pc4'), 7.517937617539371e-10),
 (('region_pc4', 'region_pc10'), 2.187676956468748e-10),
 (('region_pc10', 'region_pc4'), 2.187676956468748e-10),
 (('region_pc4', 'region_pc8'), 1.1519515839384683e-10),
 (('region_pc8', 'region_pc4'), 1.1519515839384683e-10)]
```

We need to remove one of the highly correlated varibles. Firstly we remove the variable 'number of userscount'and check the collinearity again.

In [71]:
```
final_df=final_df.drop(columns=['index'])
final_df=final_df.drop(columns=['number_users'])
final_df
```

Out[71]:

| | artist_name | success | count | passion_score | weightedscore | malepercentage | Dependent | Senior | YoungAdult | region_pc1 | region_pc2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | #90s Update | 0 | -0.176036 | -0.114376 | 0.496370 | 0.077692 | -0.192020 | -0.450729 | -0.749429 | -0.181890 | 0.027127 |
| **1** | 17 Memphis | 0 | -0.176158 | -0.195039 | 0.295276 | -1.078466 | 0.631405 | 0.155563 | 0.278416 | -0.181812 | 0.027157 |
| **2** | 2D | 0 | -0.176496 | -0.195039 | -1.463501 | 1.811929 | 7.493283 | -0.450729 | -1.698210 | -0.182202 | 0.027240 |
| **3** | 3JS | 0 | -0.176373 | 0.107446 | -1.463501 | 0.728031 | -0.740970 | -0.450729 | -0.512234 | -0.182077 | 0.027143 |
| **4** | 99 Percent | 0 | -0.136942 | -0.091243 | 0.651924 | -1.156256 | 2.385255 | -0.247261 | -0.049905 | -0.147007 | 0.014216 |
| **5** | A Boogie Wit Da Hoodie | 0 | 0.127149 | 0.148664 | 0.563629 | 0.567407 | 1.094350 | -0.299946 | 0.665084 | 0.019168 | -0.023976 |
| **6** | A Boogie Wit da Hoodie | 1 | 0.230173 | 0.033845 | 0.643047 | 0.383446 | 1.478543 | -0.266364 | 0.380483 | 0.394901 | 0.341414 |
| **7** | A R I Z O N A | 1 | 1.933932 | 0.006860 | 0.598507 | -0.471227 | 0.412100 | -0.196180 | 0.230898 | 2.879146 | -0.022420 |
| **8** | AGWA | 0 | -0.176434 | -0.195039 | 1.847138 | 1.811929 | -0.740970 | 4.399609 | -1.698210 | -0.182138 | 0.027221 |
| **9** | ALMA | 0 | -0.176281 | -0.195039 | -1.463501 | -0.355867 | -0.740970 | 0.458709 | 0.080753 | -0.181863 | 0.027089 |
| **10** | ALP | 0 | -0.176404 | -0.195039 | 0.191819 | -1.439765 | 3.376156 | -0.450729 | -0.512234 | -0.182207 | 0.027249 |
| **11** | AV AV AV | 0 | -0.174778 | -0.127820 | -1.463501 | 0.928753 | -0.740970 | -0.450729 | -0.380459 | -0.181647 | 0.027098 |
| **12** | AVVAH | 0 | -0.175913 | -0.195039 | 0.345292 | -1.006206 | 0.082455 | -0.450729 | 0.910936 | -0.181954 | 0.027176 |
| **13** | AXSHN | 1 | -0.173092 | -0.161738 | 0.727898 | -0.574635 | 1.029010 | -0.110752 | -0.190802 | -0.179053 | 0.026326 |
| **14** | Absofacto | 1 | -0.172295 | -0.177246 | 0.429502 | 0.026685 | 0.227766 | -0.290240 | 0.255162 | -0.176228 | 0.025227 |
| **15** | Adam Sample | 0 | -0.175238 | -0.195039 | 1.094128 | -0.768781 | 0.239298 | -0.277503 | 1.351442 | -0.175237 | 0.031843 |
| **16** | Adan Carmona | 0 | -0.176097 | 0.006618 | 3.117092 | -0.717166 | -0.740970 | -0.450729 | -0.404418 | -0.182151 | 0.027261 |
| **17** | Adia Victoria | 0 | -0.155952 | -0.067467 | 0.276233 | 0.233402 | -0.372883 | 0.151548 | -0.551505 | -0.168537 | 0.022534 |
| **18** | Alcatraz | 0 | -0.176312 | -0.195039 | -1.463501 | -0.046182 | -0.740970 | 0.588629 | -1.020510 | -0.181977 | 0.027176 |
| **19** | Alessio Bernabei | 0 | -0.170670 | 0.076422 | 0.380998 | -0.328075 | -0.001999 | -0.264177 | 0.126368 | -0.172524 | 0.020383 |
| **20** | Alex Hoyer | 0 | -0.175698 | -0.195039 | 0.446028 | -0.757311 | 0.478919 | -0.181266 | -0.292609 | -0.181371 | 0.027117 |
| **21** | Alex Roy | 0 | -0.176434 | -0.195039 | 3.117092 | -2.523663 | -0.740970 | -0.450729 | 1.464392 | -0.182184 | 0.027229 |
| **22** | Alexander Brown | 0 | -0.172019 | -0.143552 | 0.332434 | 0.212987 | -0.507374 | -0.141133 | -0.318777 | -0.180284 | 0.026580 |
| **23** | Alexander Cardinale | 0 | -0.158620 | -0.124898 | 0.358036 | -0.811418 | 0.128951 | -0.013403 | -0.013217 | -0.167502 | 0.020393 |
| **24** | Alexander Charles | 0 | -0.174901 | -0.147590 | 0.624202 | 1.046825 | 0.389222 | -0.450729 | 0.813268 | -0.181089 | 0.026740 |
| **25** | Alice | 0 | -0.175146 | -0.008894 | 0.540400 | 0.477901 | 0.103569 | 0.108925 | -0.238548 | -0.181159 | 0.026626 |
| **26** | Aliose | 0 | -0.175882 | 0.006618 | 0.743592 | -0.355867 | -0.740970 | -0.046534 | -0.380459 | -0.181810 | 0.027139 |
| **27** | All Tvvins | 1 | 0.143768 | -0.024569 | 0.383093 | 0.367205 | 0.207255 | -0.231471 | 0.184846 | 0.052643 | -0.060070 |
| **28** | Alma | 0 | -0.146048 | 0.040550 | 0.374291 | 0.264248 | 0.518970 | -0.170227 | -0.052133 | -0.079220 | -0.188054 |
| **29** | Amaro Ferreiro | 0 | -0.176281 | 1.014901 | -1.463501 | 0.728031 | -0.740970 | -0.450729 | -0.116909 | -0.182217 | 0.027250 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **630** | Whethan | 0 | -0.041675 | -0.195039 | 0.395066 | -0.416932 | -0.480025 | 0.573990 | -0.695977 | -0.038634 | 0.023829 |
| **631** | Wild Youth | 1 | -0.167818 | -0.195039 | -1.463501 | 0.791790 | -0.256602 | 0.405213 | -0.023891 | -0.156552 | 0.077764 |
| **632** | Wildling | 0 | -0.176005 | 0.068100 | 0.759848 | -0.489707 | 0.482499 | -0.189914 | 0.355777 | -0.182000 | 0.027202 |
| **633** | Will Joseph Cook | 1 | 0.293275 | 0.037613 | 0.539275 | -0.354518 | 0.335302 | -0.245977 | 0.541719 | 0.269432 | -0.252760 |
| **634** | Willy William | 0 | -0.117778 | 0.464928 | -1.463501 | 1.417784 | -0.740970 | -0.450729 | -0.835682 | -0.098638 | -0.156896 |
| **635** | Witek Muzyk Ulicy | 0 | -0.176005 | -0.195039 | -1.463501 | -0.073111 | -0.740970 | -0.134403 | 0.364356 | -0.181888 | 0.027241 |
| **636** | Wudstik | 0 | -0.175821 | -0.195039 | -1.463501 | -2.523663 | -0.740970 | -0.450729 | -1.698210 | -0.181758 | 0.027013 |
| **637** | Xavier Cugat y su orquesta | 0 | -0.176496 | -0.101485 | 0.281572 | -0.366293 | -0.159477 | -0.099190 | 0.040310 | -0.182219 | 0.027251 |
| **638** | Xavier Dunn | 1 | 0.002049 | -0.115244 | 0.523013 | 0.832924 | 0.954492 | -0.325018 | 0.644711 | -0.047847 | -0.024444 |
| **639** | YFN Lucci | 0 | -0.130932 | -0.180635 | 0.304444 | 0.263503 | -0.003574 | -0.255268 | 0.043581 | -0.132275 | 0.020076 |
| **640** | YONAKA | 0 | -0.166101 | 0.009237 | 0.380998 | 0.460576 | 0.017448 | -0.354999 | 1.110680 | -0.163413 | -0.083496 |

| | artist_name | success | count | passion_score | weightedscore | malepercentage | Dependent | Senior | YoungAdult | region_pc1 | region_pc2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **641** | Yasutaka Nakata | 0 | -0.173767 | -0.035151 | 0.478201 | 0.469995 | 0.192468 | -0.281360 | 0.527295 | -0.180006 | 0.026421 |
| **642** | Yellow Claw | 1 | 0.029951 | 2.224841 | -1.463501 | 1.811929 | -0.740970 | -0.450729 | -1.698210 | 0.151826 | -0.265502 |
| **643** | Ylric Illians | 0 | -0.176434 | -0.195039 | -1.463501 | 1.811929 | -0.740970 | -0.450729 | 3.045692 | -0.182150 | 0.027246 |
| **644** | Young F | 0 | -0.176496 | -0.132061 | 0.238992 | 0.943443 | 0.420438 | -0.300837 | 0.782753 | -0.182200 | 0.027241 |
| **645** | Young Spray | 0 | -0.156075 | 0.120597 | 0.818727 | 0.967152 | 0.884562 | -0.309457 | 0.612332 | -0.168229 | 0.023239 |
| **646** | YoungBoy Never Broke Again | 0 | -0.152518 | 0.272804 | 0.747140 | 1.060427 | 1.289394 | -0.351064 | 0.641249 | -0.158712 | 0.024421 |
| **647** | Youngboy Never Broke Again | 0 | -0.173337 | -0.130794 | 1.016106 | -0.183211 | 2.152146 | -0.385184 | 0.331838 | -0.178787 | 0.026837 |
| **648** | Yvng Swag | 0 | -0.169229 | -0.195039 | 0.571214 | -0.098671 | -0.322279 | 0.165839 | -0.250918 | -0.172383 | 0.025746 |
| **649** | Zac Brown | 0 | -0.174717 | 0.046949 | -1.463501 | -0.355867 | -0.740970 | -0.450729 | -0.644009 | -0.180501 | 0.026656 |
| **650** | Zak & Diego | 0 | -0.176158 | -0.011665 | 0.560322 | -0.505188 | 0.074610 | -0.160358 | 0.338439 | -0.182115 | 0.027224 |
| **651** | Zak Abel | 1 | 0.650303 | -0.195039 | -1.463501 | 1.811929 | -0.740970 | -0.450729 | -1.698210 | 0.550193 | 0.001384 |
| **652** | Zakopower | 0 | -0.176496 | 0.275493 | 1.640223 | 0.848464 | 0.631405 | -0.046534 | 0.146641 | -0.182219 | 0.027251 |
| **653** | Zarcort | 0 | -0.175760 | -0.195039 | -1.463501 | 1.811929 | -0.740970 | -0.450729 | -1.698210 | -0.173227 | -0.014794 |
| **654** | Zbigniew Kurtycz | 0 | -0.176465 | -0.010615 | 0.631732 | -0.569547 | -0.129775 | -0.270195 | 0.281956 | -0.182187 | 0.027241 |
| **655** | Zion & Lennox | 1 | 0.152200 | -0.195039 | 0.191819 | -0.789426 | 1.317593 | -0.450729 | 0.199351 | 0.578635 | 0.750254 |
| **656** | birthday | 0 | -0.175913 | 0.222413 | 0.403924 | -0.563240 | 0.391649 | -0.278470 | 0.640631 | -0.181827 | 0.027139 |
| **657** | dvsn | 1 | 0.595172 | -0.183836 | 0.457025 | -0.235434 | 0.555162 | -0.181266 | 0.058791 | 0.469397 | -0.294230 |
| **658** | flor | 1 | -0.173184 | -0.032936 | 0.622407 | -0.787474 | 1.164031 | -0.232967 | 0.149800 | -0.179915 | 0.026429 |
| **659** | gnash | 1 | 4.903629 | 0.006618 | 0.191819 | -0.355867 | 2.003781 | -0.450729 | -0.907560 | 4.020854 | -1.664809 |

659 rows × 19 columns

In [72]:
```
features=['count',"passion_score","malepercentage","weightedscore","Dependent","Senior","YoungAdult",'region_pc1', 'region_pc2', 're
gion_pc3', 'region_pc4', 'region_pc5','region_pc6', 'region_pc7', 'region_pc8', 'region_pc9', 'region_pc10']
c = final_df[features].corr().abs()
s = c.unstack()
corr = sorted(s.items(),key = lambda x: x[1],  reverse=True)
corr = [corr[x] for x in range(len(corr)) if corr[x][1]!=1]
corr
```

Out[72]:
```
[(('count', 'region_pc1'), 0.9898737867659627),
 (('region_pc1', 'count'), 0.9898737867659627),
 (('Senior', 'YoungAdult'), 0.2900948562835749),
 (('YoungAdult', 'Senior'), 0.2900948562835749),
 (('malepercentage', 'weightedscore'), 0.26453641954549817),
 (('weightedscore', 'malepercentage'), 0.26453641954549817),
 (('weightedscore', 'YoungAdult'), 0.24551888138250416),
 (('YoungAdult', 'weightedscore'), 0.24551888138250416),
 (('malepercentage', 'Dependent'), 0.1940672876063316),
 (('Dependent', 'malepercentage'), 0.1940672876063316),
 (('malepercentage', 'Senior'), 0.19224178410800338),
 (('Senior', 'malepercentage'), 0.19224178410800338),
 (('Dependent', 'Senior'), 0.18306656522873146),
 (('Senior', 'Dependent'), 0.18306656522873146),
 (('weightedscore', 'Senior'), 0.17368201581352172),
 (('Senior', 'weightedscore'), 0.17368201581352172),
 (('weightedscore', 'Dependent'), 0.15621261849548154),
 (('Dependent', 'weightedscore'), 0.15621261849548154),
 (('malepercentage', 'YoungAdult'), 0.1114751232025845),
 (('YoungAdult', 'malepercentage'), 0.1114751232025845),
 (('count', 'region_pc3'), 0.10902012311526688),
 (('region_pc3', 'count'), 0.10902012311526688),
 (('count', 'Dependent'), 0.10895342847840789),
 (('Dependent', 'count'), 0.10895342847840789),
 (('passion_score', 'malepercentage'), 0.10360014603062476),
 (('malepercentage', 'passion_score'), 0.10360014603062476),
 (('Dependent', 'region_pc1'), 0.10028118899657193),
 (('region_pc1', 'Dependent'), 0.10028118899657193),
 (('passion_score', 'weightedscore'), 0.0975993741394282),
 (('weightedscore', 'passion_score'), 0.0975993741394282),
 (('passion_score', 'YoungAdult'), 0.09690902730132399),
 (('YoungAdult', 'passion_score'), 0.09690902730132399),
 (('weightedscore', 'region_pc10'), 0.0687990323717114),
 (('region_pc10', 'weightedscore'), 0.0687990323717114),
 (('malepercentage', 'region_pc1'), 0.06455253149903238),
 (('region_pc1', 'malepercentage'), 0.06455253149903238),
 (('Dependent', 'region_pc3'), 0.06408214210887761),
 (('region_pc3', 'Dependent'), 0.06408214210887761),
 (('count', 'malepercentage'), 0.060545756616547),
 (('malepercentage', 'count'), 0.060545756616547),
 (('malepercentage', 'region_pc7'), 0.05774476674353545),
 (('region_pc7', 'malepercentage'), 0.05774476674353545),
 (('count', 'weightedscore'), 0.056937283818603544),
 (('weightedscore', 'count'), 0.056937283818603544),
 (('weightedscore', 'region_pc1'), 0.056445049551071505),
 (('region_pc1', 'weightedscore'), 0.056445049551071505),
 (('count', 'region_pc5'), 0.04880734305689876),
 (('region_pc5', 'count'), 0.04880734305689876),
 (('passion_score', 'Dependent'), 0.045957677884510974),
 (('Dependent', 'passion_score'), 0.045957677884510974),
 (('YoungAdult', 'region_pc10'), 0.044651042285670905),
 (('region_pc10', 'YoungAdult'), 0.044651042285670905),
 (('YoungAdult', 'region_pc8'), 0.038532889700918935),
 (('region_pc8', 'YoungAdult'), 0.038532889700918935),
 (('malepercentage', 'region_pc8'), 0.038409406275909504),
 (('region_pc8', 'malepercentage'), 0.038409406275909504),
 (('Dependent', 'region_pc4'), 0.038174282112028994),
 (('region_pc4', 'Dependent'), 0.038174282112028994),
 (('Dependent', 'region_pc8'), 0.03329161019137374),
 (('region_pc8', 'Dependent'), 0.03329161019137374),
 (('count', 'region_pc7'), 0.03134405782914076),
 (('region_pc7', 'count'), 0.03134405782914076),
 (('Dependent', 'region_pc9'), 0.029701234307331917),
 (('region_pc9', 'Dependent'), 0.029701234307331917),
 (('malepercentage', 'region_pc4'), 0.02951811129583706),
 (('region_pc4', 'malepercentage'), 0.02951811129583706),
 (('malepercentage', 'region_pc10'), 0.028305190598450932),
 (('region_pc10', 'malepercentage'), 0.028305190598450932),
 (('Dependent', 'YoungAdult'), 0.027891528463898602),
 (('YoungAdult', 'Dependent'), 0.027891528463898602),
 (('malepercentage', 'region_pc5'), 0.02747630067346027),
 (('region_pc5', 'malepercentage'), 0.02747630067346027),
 (('weightedscore', 'region_pc8'), 0.02521810640738936),
 (('region_pc8', 'weightedscore'), 0.02521810640738936),
 (('weightedscore', 'region_pc6'), 0.024558204965050092),
 (('region_pc6', 'weightedscore'), 0.024558204965050092),
 (('YoungAdult', 'region_pc7'), 0.02381356791507353),
 (('region_pc7', 'YoungAdult'), 0.02381356791507353),
 (('YoungAdult', 'region_pc9'), 0.02274344833197705),
 (('region_pc9', 'YoungAdult'), 0.02274344833197705),
 (('passion_score', 'Senior'), 0.02242663112590295),
```

```
(('Senior', 'passion_score'), 0.02242663112590295),
(('Dependent', 'region_pc5'), 0.020459814513901803),
(('region_pc5', 'Dependent'), 0.020459814513901803),
(('weightedscore', 'region_pc9'), 0.019850383202387415),
(('region_pc9', 'weightedscore'), 0.019850383202387415),
(('count', 'region_pc10'), 0.019321588509432823),
(('region_pc10', 'count'), 0.019321588509432823),
(('count', 'region_pc4'), 0.01889190112438198),
(('region_pc4', 'count'), 0.01889190112438198),
(('YoungAdult', 'region_pc5'), 0.018058259382282872),
(('region_pc5', 'YoungAdult'), 0.018058259382282872),
(('Senior', 'region_pc1'), 0.017931846936521525),
(('region_pc1', 'Senior'), 0.017931846936521525),
(('Senior', 'region_pc7'), 0.017088249454834),
(('region_pc7', 'Senior'), 0.017088249454834),
(('Dependent', 'region_pc2'), 0.01704282704383673),
(('region_pc2', 'Dependent'), 0.01704282704383673),
(('count', 'region_pc2'), 0.01690725942879011),
(('region_pc2', 'count'), 0.01690725942879011),
(('count', 'region_pc6'), 0.01656477417907123),
(('region_pc6', 'count'), 0.01656477417907123),
(('weightedscore', 'region_pc3'), 0.016402473657499292),
(('region_pc3', 'weightedscore'), 0.016402473657499292),
(('Senior', 'region_pc3'), 0.01605160141415977),
(('region_pc3', 'Senior'), 0.01605160141415977),
(('count', 'region_pc9'), 0.01586854393557983),
(('region_pc9', 'count'), 0.01586854393557983),
(('Senior', 'region_pc8'), 0.01424578298731101),
(('region_pc8', 'Senior'), 0.01424578298731101),
(('YoungAdult', 'region_pc6'), 0.013900305193216557),
(('region_pc6', 'YoungAdult'), 0.013900305193216557),
(('count', 'region_pc8'), 0.013391644760323685),
(('region_pc8', 'count'), 0.013391644760323685),
(('malepercentage', 'region_pc9'), 0.012967810359164327),
(('region_pc9', 'malepercentage'), 0.012967810359164327),
(('Dependent', 'region_pc10'), 0.011259353320483731),
(('region_pc10', 'Dependent'), 0.011259353320483731),
(('count', 'Senior'), 0.010930029543369604),
(('Senior', 'count'), 0.010930029543369604),
(('malepercentage', 'region_pc2'), 0.010319525933010616),
(('region_pc2', 'malepercentage'), 0.010319525933010616),
(('YoungAdult', 'region_pc1'), 0.009309475329385098),
(('region_pc1', 'YoungAdult'), 0.009309475329385098),
(('count', 'passion_score'), 0.009178702589767978),
(('passion_score', 'count'), 0.009178702589767978),
(('Senior', 'region_pc2'), 0.008894234785797261),
(('region_pc2', 'Senior'), 0.008894234785797261),
(('passion_score', 'region_pc1'), 0.00874202870956091),
(('region_pc1', 'passion_score'), 0.00874202870956091),
(('YoungAdult', 'region_pc2'), 0.007873931165980676),
(('region_pc2', 'YoungAdult'), 0.007873931165980676),
(('count', 'YoungAdult'), 0.007827231763698867),
(('YoungAdult', 'count'), 0.007827231763698867),
(('weightedscore', 'region_pc5'), 0.0068329641631396055),
(('region_pc5', 'weightedscore'), 0.0068329641631396055),
(('weightedscore', 'region_pc7'), 0.00667622174568678),
(('region_pc7', 'weightedscore'), 0.00667622174568678),
(('weightedscore', 'region_pc4'), 0.006511622812787667),
(('region_pc4', 'weightedscore'), 0.006511622812787667),
(('Senior', 'region_pc9'), 0.006378462772257145),
(('region_pc9', 'Senior'), 0.006378462772257145),
(('passion_score', 'region_pc2'), 0.006045862770993209),
(('region_pc2', 'passion_score'), 0.006045862770993209),
(('passion_score', 'region_pc5'), 0.005795341585632269),
(('region_pc5', 'passion_score'), 0.005795341585632269),
(('YoungAdult', 'region_pc3'), 0.005549245668271518),
(('region_pc3', 'YoungAdult'), 0.005549245668271518),
(('YoungAdult', 'region_pc4'), 0.005344963334627365),
(('region_pc4', 'YoungAdult'), 0.005344963334627365),
(('passion_score', 'region_pc9'), 0.005054481114466202),
(('region_pc9', 'passion_score'), 0.005054481114466202),
(('Senior', 'region_pc5'), 0.004206478571370755),
(('region_pc5', 'Senior'), 0.004206478571370755),
(('malepercentage', 'region_pc6'), 0.003539637872845347),
(('region_pc6', 'malepercentage'), 0.003539637872845347),
(('passion_score', 'region_pc10'), 0.003513640314324124),
(('region_pc10', 'passion_score'), 0.003513640314324124),
(('passion_score', 'region_pc8'), 0.0034151041986756874),
(('region_pc8', 'passion_score'), 0.0034151041986756874),
(('malepercentage', 'region_pc3'), 0.0027958404078685917),
(('region_pc3', 'malepercentage'), 0.0027958404078685917),
(('Senior', 'region_pc6'), 0.0025571594416695063),
```

```
(('region_pc6', 'Senior'), 0.0025571594416695063),
(('Senior', 'region_pc10'), 0.002306930790496077),
(('region_pc10', 'Senior'), 0.002306930790496077),
(('passion_score', 'region_pc3'), 0.0019001827793416709),
(('region_pc3', 'passion_score'), 0.0019001827793416709),
(('weightedscore', 'region_pc2'), 0.001726979078403338),
(('region_pc2', 'weightedscore'), 0.001726979078403338),
(('passion_score', 'region_pc4'), 0.0015538734608392811),
(('region_pc4', 'passion_score'), 0.0015538734608392811),
(('passion_score', 'region_pc7'), 0.0013583796983024363),
(('region_pc7', 'passion_score'), 0.0013583796983024363),
(('Senior', 'region_pc4'), 0.0013290005529663302),
(('region_pc4', 'Senior'), 0.0013290005529663302),
(('Dependent', 'region_pc6'), 0.0013227951724981608),
(('region_pc6', 'Dependent'), 0.0013227951724981608),
(('passion_score', 'region_pc6'), 0.0010049147065156823),
(('region_pc6', 'passion_score'), 0.0010049147065156823),
(('Dependent', 'region_pc7'), 0.00017310709530611208),
(('region_pc7', 'Dependent'), 0.00017310709530611208),
(('region_pc1', 'region_pc7'), 3.193684451216419e-05),
(('region_pc7', 'region_pc1'), 3.193684451216419e-05),
(('region_pc1', 'region_pc3'), 2.60468428692369e-05),
(('region_pc3', 'region_pc1'), 2.60468428692369e-05),
(('region_pc1', 'region_pc5'), 1.956024466359322e-05),
(('region_pc5', 'region_pc1'), 1.956024466359322e-05),
(('region_pc1', 'region_pc2'), 1.5056051943031464e-05),
(('region_pc2', 'region_pc1'), 1.5056051943031464e-05),
(('region_pc1', 'region_pc6'), 1.1023181009390485e-05),
(('region_pc6', 'region_pc1'), 1.1023181009390485e-05),
(('region_pc3', 'region_pc7'), 8.258409637082782e-06),
(('region_pc7', 'region_pc3'), 8.258409637082782e-06),
(('region_pc5', 'region_pc7'), 6.201768652620103e-06),
(('region_pc7', 'region_pc5'), 6.201768652620103e-06),
(('region_pc3', 'region_pc5'), 5.057998462895076e-06),
(('region_pc5', 'region_pc3'), 5.057998462895076e-06),
(('region_pc2', 'region_pc7'), 4.773669883373147e-06),
(('region_pc7', 'region_pc2'), 4.773669883373147e-06),
(('region_pc1', 'region_pc9'), 4.065181064247601e-06),
(('region_pc9', 'region_pc1'), 4.065181064247601e-06),
(('region_pc2', 'region_pc3'), 3.8932788351207454e-06),
(('region_pc3', 'region_pc2'), 3.8932788351207454e-06),
(('region_pc6', 'region_pc7'), 3.495009127193439e-06),
(('region_pc7', 'region_pc6'), 3.495009127193439e-06),
(('region_pc2', 'region_pc5'), 2.9237124767641364e-06),
(('region_pc5', 'region_pc2'), 2.9237124767641364e-06),
(('region_pc3', 'region_pc6'), 2.8504433510898885e-06),
(('region_pc6', 'region_pc3'), 2.8504433510898885e-06),
(('region_pc5', 'region_pc6'), 2.1405756499019757e-06),
(('region_pc6', 'region_pc5'), 2.1405756499019757e-06),
(('region_pc2', 'region_pc6'), 1.6476592164781537e-06),
(('region_pc6', 'region_pc2'), 1.6476592164781537e-06),
(('region_pc7', 'region_pc9'), 1.2889050413090252e-06),
(('region_pc9', 'region_pc7'), 1.2889050413090252e-06),
(('region_pc1', 'region_pc10'), 1.1697373762716279e-06),
(('region_pc10', 'region_pc1'), 1.1697373762716279e-06),
(('region_pc3', 'region_pc9'), 1.051190766592958e-06),
(('region_pc9', 'region_pc3'), 1.051190766592958e-06),
(('region_pc5', 'region_pc9'), 7.894111039687379e-07),
(('region_pc9', 'region_pc5'), 7.894111039687379e-07),
(('region_pc1', 'region_pc8'), 6.257317598721048e-07),
(('region_pc8', 'region_pc1'), 6.257317598721048e-07),
(('region_pc2', 'region_pc9'), 6.076312177482547e-07),
(('region_pc9', 'region_pc2'), 6.076312177482547e-07),
(('region_pc6', 'region_pc9'), 4.448680977911559e-07),
(('region_pc9', 'region_pc6'), 4.448680977911559e-07),
(('region_pc7', 'region_pc10'), 3.708769665948633e-07),
(('region_pc10', 'region_pc7'), 3.708769665948633e-07),
(('region_pc3', 'region_pc10'), 3.0247895578159924e-07),
(('region_pc10', 'region_pc3'), 3.0247895578159924e-07),
(('region_pc5', 'region_pc10'), 2.2714967191255054e-07),
(('region_pc10', 'region_pc5'), 2.2714967191255054e-07),
(('region_pc7', 'region_pc8'), 1.9839423055449377e-07),
(('region_pc8', 'region_pc7'), 1.9839423055449377e-07),
(('region_pc2', 'region_pc10'), 1.748432670450361e-07),
(('region_pc10', 'region_pc2'), 1.748432670450361e-07),
(('region_pc3', 'region_pc8'), 1.6180344827617363e-07),
(('region_pc8', 'region_pc3'), 1.6180344827617363e-07),
(('region_pc6', 'region_pc10'), 1.2801148020448216e-07),
(('region_pc10', 'region_pc6'), 1.2801148020448216e-07),
(('region_pc5', 'region_pc8'), 1.2150981859132372e-07),
(('region_pc8', 'region_pc5'), 1.2150981859132372e-07),
(('region_pc2', 'region_pc8'), 9.352941913166267e-08),
```

```
(('region_pc8', 'region_pc2'), 9.352941913166267e-08),
(('region_pc6', 'region_pc8'), 6.847560590492236e-08),
(('region_pc8', 'region_pc6'), 6.847560590492236e-08),
(('region_pc9', 'region_pc10'), 4.720705957879839e-08),
(('region_pc10', 'region_pc9'), 4.720705957879839e-08),
(('region_pc8', 'region_pc9'), 2.52544627783188e-08),
(('region_pc9', 'region_pc8'), 2.52544627783188e-08),
(('region_pc1', 'region_pc4'), 1.873900781895363e-08),
(('region_pc4', 'region_pc1'), 1.873900781895363e-08),
(('region_pc8', 'region_pc10'), 7.266197752399767e-09),
(('region_pc10', 'region_pc8'), 7.266197752399767e-09),
(('region_pc4', 'region_pc7'), 5.942120958912061e-09),
(('region_pc7', 'region_pc4'), 5.942120958912061e-09),
(('region_pc3', 'region_pc4'), 4.8522421476040205e-09),
(('region_pc4', 'region_pc3'), 4.8522421476040205e-09),
(('region_pc4', 'region_pc5'), 3.6392944584245098e-09),
(('region_pc5', 'region_pc4'), 3.6392944584245098e-09),
(('region_pc2', 'region_pc4'), 2.801228203917875e-09),
(('region_pc4', 'region_pc2'), 2.801228203917875e-09),
(('region_pc4', 'region_pc6'), 2.0556818239044856e-09),
(('region_pc6', 'region_pc4'), 2.0556818239044856e-09),
(('region_pc4', 'region_pc9'), 7.517937617539371e-10),
(('region_pc9', 'region_pc4'), 7.517937617539371e-10),
(('region_pc4', 'region_pc10'), 2.187676956468748e-10),
(('region_pc10', 'region_pc4'), 2.187676956468748e-10),
(('region_pc4', 'region_pc8'), 1.1519515839384683e-10),
(('region_pc8', 'region_pc4'), 1.1519515839384683e-10)]
```

Then we remove the stream count.

In [73]:
```
final_df=final_df.drop(columns=['count'])
final_df
```

Out[73]:

| | artist_name | success | passion_score | weightedscore | malepercentage | Dependent | Senior | YoungAdult | region_pc1 | region_pc2 | region_pc3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | #90s Update | 0 | -0.114376 | 0.496370 | 0.077692 | -0.192020 | -0.450729 | -0.749429 | -0.181890 | 0.027127 | -0.047444 |
| **1** | 17 Memphis | 0 | -0.195039 | 0.295276 | -1.078466 | 0.631405 | 0.155563 | 0.278416 | -0.181812 | 0.027157 | -0.047693 |
| **2** | 2D | 0 | -0.195039 | -1.463501 | 1.811929 | 7.493283 | -0.450729 | -1.698210 | -0.182202 | 0.027240 | -0.047163 |
| **3** | 3JS | 0 | 0.107446 | -1.463501 | 0.728031 | -0.740970 | -0.450729 | -0.512234 | -0.182077 | 0.027143 | -0.047344 |
| **4** | 99 Percent | 0 | -0.091243 | 0.651924 | -1.156256 | 2.385255 | -0.247261 | -0.049905 | -0.147007 | 0.014216 | -0.096025 |
| **5** | A Boogie Wit Da Hoodie | 0 | 0.148664 | 0.563629 | 0.567407 | 1.094350 | -0.299946 | 0.665084 | 0.019168 | -0.023976 | -0.157641 |
| **6** | A Boogie Wit da Hoodie | 1 | 0.033845 | 0.643047 | 0.383446 | 1.478543 | -0.266364 | 0.380483 | 0.394901 | 0.341414 | 2.427304 |
| **7** | A R I Z O N A | 1 | 0.006860 | 0.598507 | -0.471227 | 0.412100 | -0.196180 | 0.230898 | 2.879146 | -0.022420 | 6.251410 |
| **8** | AGWA | 0 | -0.195039 | 1.847138 | 1.811929 | -0.740970 | 4.399609 | -1.698210 | -0.182138 | 0.027221 | -0.047156 |
| **9** | ALMA | 0 | -0.195039 | -1.463501 | -0.355867 | -0.740970 | 0.458709 | 0.080753 | -0.181863 | 0.027089 | -0.047690 |
| **10** | ALP | 0 | -0.195039 | 0.191819 | -1.439765 | 3.376156 | -0.450729 | -0.512234 | -0.182207 | 0.027249 | -0.047151 |
| **11** | AV AV AV | 0 | -0.127820 | -1.463501 | 0.928753 | -0.740970 | -0.450729 | -0.380459 | -0.181647 | 0.027098 | -0.047443 |
| **12** | AVVAH | 0 | -0.195039 | 0.345292 | -1.006206 | 0.082455 | -0.450729 | 0.910936 | -0.181954 | 0.027176 | -0.046786 |
| **13** | AXSHN | 1 | -0.161738 | 0.727898 | -0.574635 | 1.029010 | -0.110752 | -0.190802 | -0.179053 | 0.026326 | -0.047692 |
| **14** | Absofacto | 1 | -0.177246 | 0.429502 | 0.026685 | 0.227766 | -0.290240 | 0.255162 | -0.176228 | 0.025227 | -0.029911 |
| **15** | Adam Sample | 0 | -0.195039 | 1.094128 | -0.768781 | 0.239298 | -0.277503 | 1.351442 | -0.175237 | 0.031843 | -0.025801 |
| **16** | Adan Carmona | 0 | 0.006618 | 3.117092 | -0.717166 | -0.740970 | -0.450729 | -0.404418 | -0.182151 | 0.027261 | -0.047170 |
| **17** | Adia Victoria | 0 | -0.067467 | 0.276233 | 0.233402 | -0.372883 | 0.151548 | -0.551505 | -0.168537 | 0.022534 | -0.059181 |
| **18** | Alcatraz | 0 | -0.195039 | -1.463501 | -0.046182 | -0.740970 | 0.588629 | -1.020510 | -0.181977 | 0.027176 | -0.047120 |
| **19** | Alessio Bernabei | 0 | 0.076422 | 0.380998 | -0.328075 | -0.001999 | -0.264177 | 0.126368 | -0.172524 | 0.020383 | 0.026792 |
| **20** | Alex Hoyer | 0 | -0.195039 | 0.446028 | -0.757311 | 0.478919 | -0.181266 | -0.292609 | -0.181371 | 0.027117 | -0.046930 |
| **21** | Alex Roy | 0 | -0.195039 | 3.117092 | -2.523663 | -0.740970 | -0.450729 | 1.464392 | -0.182184 | 0.027229 | -0.047196 |
| **22** | Alexander Brown | 0 | -0.143552 | 0.332434 | 0.212987 | -0.507374 | -0.141133 | -0.318777 | -0.180284 | 0.026580 | -0.049804 |
| **23** | Alexander Cardinale | 0 | -0.124898 | 0.358036 | -0.811418 | 0.128951 | -0.013403 | -0.013217 | -0.167502 | 0.020393 | -0.065333 |
| **24** | Alexander Charles | 0 | -0.147590 | 0.624202 | 1.046825 | 0.389222 | -0.450729 | 0.813268 | -0.181089 | 0.026740 | -0.048547 |
| **25** | Alice | 0 | -0.008894 | 0.540400 | 0.477901 | 0.103569 | 0.108925 | -0.238548 | -0.181159 | 0.026626 | -0.048573 |
| **26** | Aliose | 0 | 0.006618 | 0.743592 | -0.355867 | -0.740970 | -0.046534 | -0.380459 | -0.181810 | 0.027139 | -0.046714 |
| **27** | All Tvvins | 1 | -0.024569 | 0.383093 | 0.367205 | 0.207255 | -0.231471 | 0.184846 | 0.052643 | -0.060070 | -0.256201 |
| **28** | Alma | 0 | 0.040550 | 0.374291 | 0.264248 | 0.518970 | -0.170227 | -0.052133 | -0.079220 | -0.188054 | 0.176932 |
| **29** | Amaro Ferreiro | 0 | 1.014901 | -1.463501 | 0.728031 | -0.740970 | -0.450729 | -0.116909 | -0.182217 | 0.027250 | -0.047136 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **630** | Whethan | 0 | -0.195039 | 0.395066 | -0.416932 | -0.480025 | 0.573990 | -0.695977 | -0.038634 | 0.023829 | -0.086762 |
| **631** | Wild Youth | 1 | -0.195039 | -1.463501 | 0.791790 | -0.256602 | 0.405213 | -0.023891 | -0.156552 | 0.077764 | 0.002793 |
| **632** | Wildling | 0 | 0.068100 | 0.759848 | -0.489707 | 0.482499 | -0.189914 | 0.355777 | -0.182000 | 0.027202 | -0.047004 |
| **633** | Will Joseph Cook | 1 | 0.037613 | 0.539275 | -0.354518 | 0.335302 | -0.245977 | 0.541719 | 0.269432 | -0.252760 | -0.049741 |
| **634** | Willy William | 0 | 0.464928 | -1.463501 | 1.417784 | -0.740970 | -0.450729 | -0.835682 | -0.098638 | -0.156896 | -0.010792 |
| **635** | Witek Muzyk Ulicy | 0 | -0.195039 | -1.463501 | -0.073111 | -0.740970 | -0.134403 | 0.364356 | -0.181888 | 0.027241 | -0.046777 |
| **636** | Wudstik | 0 | -0.195039 | -1.463501 | -2.523663 | -0.740970 | -0.450729 | -1.698210 | -0.181758 | 0.027013 | -0.047703 |
| **637** | Xavier Cugat y su orquesta | 0 | -0.101485 | 0.281572 | -0.366293 | -0.159477 | -0.099190 | 0.040310 | -0.182219 | 0.027251 | -0.047132 |
| **638** | Xavier Dunn | 1 | -0.115244 | 0.523013 | 0.832924 | 0.954492 | -0.325018 | 0.644711 | -0.047847 | -0.024444 | -0.123017 |
| **639** | YFN Lucci | 0 | -0.180635 | 0.304444 | 0.263503 | -0.003574 | -0.255268 | 0.043581 | -0.132275 | 0.020076 | 0.278806 |
| **640** | YONAKA | 0 | 0.009237 | 0.380998 | 0.460576 | 0.017448 | -0.354999 | 1.110680 | -0.163413 | -0.083496 | -0.000707 |

| | artist_name | success | passion_score | weightedscore | malepercentage | Dependent | Senior | YoungAdult | region_pc1 | region_pc2 | region_pc3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **641** | Yasutaka Nakata | 0 | -0.035151 | 0.478201 | 0.469995 | 0.192468 | -0.281360 | 0.527295 | -0.180006 | 0.026421 | -0.047073 |
| **642** | Yellow Claw | 1 | 2.224841 | -1.463501 | 1.811929 | -0.740970 | -0.450729 | -1.698210 | 0.151826 | -0.265502 | 0.364808 |
| **643** | Ylric Illians | 0 | -0.195039 | -1.463501 | 1.811929 | -0.740970 | -0.450729 | 3.045692 | -0.182150 | 0.027246 | -0.047180 |
| **644** | Young F | 0 | -0.132061 | 0.238992 | 0.943443 | 0.420438 | -0.300837 | 0.782753 | -0.182200 | 0.027241 | -0.047171 |
| **645** | Young Spray | 0 | 0.120597 | 0.818727 | 0.967152 | 0.884562 | -0.309457 | 0.612332 | -0.168229 | 0.023239 | -0.060957 |
| **646** | YoungBoy Never Broke Again | 0 | 0.272804 | 0.747140 | 1.060427 | 1.289394 | -0.351064 | 0.641249 | -0.158712 | 0.024421 | -0.035003 |
| **647** | Youngboy Never Broke Again | 0 | -0.130794 | 1.016106 | -0.183211 | 2.152146 | -0.385184 | 0.331838 | -0.178787 | 0.026837 | -0.048167 |
| **648** | Yvng Swag | 0 | -0.195039 | 0.571214 | -0.098671 | -0.322279 | 0.165839 | -0.250918 | -0.172383 | 0.025746 | -0.025219 |
| **649** | Zac Brown | 0 | 0.046949 | -1.463501 | -0.355867 | -0.740970 | -0.450729 | -0.644009 | -0.180501 | 0.026656 | -0.048918 |
| **650** | Zak & Diego | 0 | -0.011665 | 0.560322 | -0.505188 | 0.074610 | -0.160358 | 0.338439 | -0.182115 | 0.027224 | -0.047210 |
| **651** | Zak Abel | 1 | -0.195039 | -1.463501 | 1.811929 | -0.740970 | -0.450729 | -1.698210 | 0.550193 | 0.001384 | 0.417429 |
| **652** | Zakopower | 0 | 0.275493 | 1.640223 | 0.848464 | 0.631405 | -0.046534 | 0.146641 | -0.182219 | 0.027251 | -0.047132 |
| **653** | Zarcort | 0 | -0.195039 | -1.463501 | 1.811929 | -0.740970 | -0.450729 | -1.698210 | -0.173227 | -0.014794 | -0.027215 |
| **654** | Zbigniew Kurtycz | 0 | -0.010615 | 0.631732 | -0.569547 | -0.129775 | -0.270195 | 0.281956 | -0.182187 | 0.027241 | -0.047041 |
| **655** | Zion & Lennox | 1 | -0.195039 | 0.191819 | -0.789426 | 1.317593 | -0.450729 | 0.199351 | 0.578635 | 0.750254 | 2.771638 |
| **656** | birthday | 0 | 0.222413 | 0.403924 | -0.563240 | 0.391649 | -0.278470 | 0.640631 | -0.181827 | 0.027139 | -0.046582 |
| **657** | dvsn | 1 | -0.183836 | 0.457025 | -0.235434 | 0.555162 | -0.181266 | 0.058791 | 0.469397 | -0.294230 | 0.637802 |
| **658** | flor | 1 | -0.032936 | 0.622407 | -0.787474 | 1.164031 | -0.232967 | 0.149800 | -0.179915 | 0.026429 | -0.049804 |
| **659** | gnash | 1 | 0.006618 | 0.191819 | -0.355867 | 2.003781 | -0.450729 | -0.907560 | 4.020854 | -1.664809 | -4.657319 |

659 rows × 18 columns

Finally, we want to take a look out the class balance in our dependent variable.

Given the natural bias in our data, i.e. there are more cases of failure than of success in the training and test sets; there is a strong bias toward predicting ‘failure’. Based on our complete (unbalanced classes) training sample, if the model only predicted ‘failure’, we would achieve an accuracy of 88.8%.

To give us a more even class balance, without losing too much data, we will sample data from the bigger class to achive a class balance closer to 60-40.

There is another way to determine the accuracy of our predictions using a confusion matrix and ROC curve, but more on that later. For now, we will go ahead with sampling the bigger class:

**ACTION: Class balance**

Calculate and comment on class balance.

We will calculate and comment on class balance in the next part.

In [74]:
```
final_df.describe()
```

Out[74]:

| | success | passion_score | weightedscore | malepercentage | Dependent | Senior | YoungAdult | region_pc1 | region_pc2 | region_pc3 | reg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 659.000000 | 6.590 |
| **mean** | 0.125948 | -0.001999 | 0.002221 | 0.000540 | 0.001124 | -0.002076 | -0.001022 | 0.000553 | -0.000083 | 0.000143 | 1.028 |
| **std** | 0.332043 | 1.000199 | 0.999890 | 1.001422 | 1.001101 | 1.000095 | 1.001174 | 1.002227 | 1.002276 | 1.002274 | 1.002 |
| **min** | 0.000000 | -0.195039 | -1.463501 | -2.523663 | -0.740970 | -0.450729 | -1.698210 | -0.182673 | -11.115732 | -9.772089 | -1.404 |
| **25%** | 0.000000 | -0.195039 | -1.463501 | -0.552939 | -0.740970 | -0.450729 | -0.531870 | -0.182056 | 0.024155 | -0.047922 | -6.79( |
| **50%** | 0.000000 | -0.143332 | 0.379213 | -0.101091 | -0.201019 | -0.272408 | 0.135481 | -0.180331 | 0.027143 | -0.047149 | -7.64: |
| **75%** | 0.000000 | -0.024730 | 0.588245 | 0.594175 | 0.398058 | -0.103065 | 0.572346 | -0.156454 | 0.027247 | -0.039213 | 3.31: |
| **max** | 1.000000 | 21.583878 | 4.743947 | 1.811929 | 7.493283 | 6.824777 | 3.045692 | 13.393821 | 20.287415 | 17.666122 | 1.577 |

# 4. Evaluate algorithms

**Model Selection**

There are number of classification models available to us via the `scikit-learn` package, and we can rapidly experiment using each of them to find the optimal model.

Below is an outline of the steps we will take to arrive at the best model:

- Split data into training and validation (hold-out) set
- Use cross-validation to fit different models to training set
- Select model with the highest cross-validation score as model of choice
- Tune hyper parameters of chosen model.
- Test the model on hold-out set

Make reference to "P6 - Pipelines and parameter tuning" from week 6 as necessary.

**ACTION: Spot-check algorithms**

Try a mixture of algorithm representations (e.g. instances and trees).

Try a mixture of learning algorithms (e.g. different algorithms for learning the same type of representation).

Try a mixture of modeling types (e.g. linear and nonlinear functions or parametric and nonparametric).

Divide this work up among the different members of your team and then compare and comment on the performance of various approaches.

In this part, we firstly split the data into train and validation set, using test size ratio equals to 0.2 and keep random state equals to 42.

In [75]:
```
from sklearn.model_selection import train_test_split
features2=['passion_score','malepercentage','weightedscore','Dependent','Senior','YoungAdult','region_pc1', 'region_pc2', 'region_pc3', 'region_pc4', 'region_pc5','region_pc6', 'region_pc7', 'region_pc8', 'region_pc9', 'region_pc10']
X=final_df[features2]
Y=final_df['success']

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

In [76]:
```
print(len(x_train))
```

```
527
```

Then we create a dataframe named train_set which contains x train and y train.

In [77]:
```
train_set=x_train.copy()
train_set["success"]=y_train
train_set
```

Out[77]:

| | passion_score | malepercentage | weightedscore | Dependent | Senior | YoungAdult | region_pc1 | region_pc2 | region_pc3 | region_pc4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **18** | -0.195039 | -0.046182 | -1.463501 | -0.740970 | 0.588629 | -1.020510 | -1.819767e-01 | 2.717605e-02 | -4.712030e-02 | -5.833343e-05 | 3 |
| **363** | -0.145987 | -0.678107 | 0.807414 | 0.547382 | -0.302249 | 0.076720 | -1.821755e-01 | 2.723835e-02 | -4.721750e-02 | -9.376095e-05 | 3 |
| **389** | 0.118649 | -0.917888 | 0.310056 | -0.740970 | 0.088197 | 0.410191 | -1.821835e-01 | 2.724189e-02 | -4.719324e-02 | -8.943787e-05 | 3 |
| **452** | -0.195039 | -0.355867 | -1.463501 | -0.740970 | -0.450729 | 0.673741 | -1.822058e-01 | 2.724584e-02 | -4.714971e-02 | -7.956949e-05 | 3 |
| **61** | -0.195039 | 0.366732 | 0.832587 | 0.288312 | -0.450729 | 1.266729 | -1.765837e-01 | 3.148539e-02 | -3.706627e-02 | -4.156105e-03 | 3 |
| **265** | -0.195039 | 1.811929 | 0.595555 | -0.740970 | 1.974440 | -0.116909 | -1.809136e-01 | 2.678007e-02 | -4.770876e-02 | 3.928689e-05 | 3 |
| **418** | -0.195039 | 1.811929 | -1.463501 | -0.740970 | -0.450729 | -1.698210 | -1.819862e-01 | 2.719817e-02 | -4.711783e-02 | 3.684685e-05 | 3 |
| **29** | 1.014901 | 0.728031 | -1.463501 | -0.740970 | -0.450729 | -0.116909 | -1.822170e-01 | 2.724997e-02 | -4.713639e-02 | -7.020079e-05 | 3 |
| **158** | -0.195039 | -0.789426 | -1.463501 | -0.740970 | -0.450729 | -0.749429 | -1.821515e-01 | 2.723987e-02 | -4.719322e-02 | -6.479092e-05 | 3 |
| **287** | 0.400834 | 0.100511 | 0.473121 | -0.431412 | 0.151005 | -0.645991 | -1.821051e-01 | 2.725181e-02 | -4.671435e-02 | 2.052338e-04 | 3 |
| **211** | -0.195039 | -0.552939 | 0.191819 | 0.007598 | -0.450729 | -0.404418 | -1.820883e-01 | 2.714923e-02 | -4.727706e-02 | -1.513760e-04 | 3 |
| **303** | -0.195039 | 1.811929 | -1.463501 | -0.740970 | -0.450729 | 3.045692 | -1.808722e-01 | 2.685999e-02 | -4.883770e-02 | -5.523059e-04 | 3 |
| **394** | -0.180970 | -0.759178 | 0.507679 | -0.549476 | 0.141463 | -0.594977 | -1.764518e-01 | 2.549962e-02 | -3.099126e-02 | 1.308614e-02 | 4 |
| **183** | -0.168447 | -0.117648 | 0.753021 | 0.578330 | -0.169356 | 0.346124 | -1.791806e-01 | 2.667086e-02 | -4.902779e-02 | -5.256255e-04 | 3 |
| **383** | -0.195039 | -2.523663 | -1.463501 | -0.740970 | -0.450729 | -1.698210 | -1.770931e-01 | 2.544677e-02 | -5.135797e-02 | -1.233894e-03 | 3 |
| **590** | -0.195039 | 0.077692 | 0.555989 | -0.740970 | -0.450729 | -0.749429 | -1.818181e-01 | 2.728438e-02 | -4.789282e-02 | -4.838841e-04 | 3 |
| **336** | -0.195039 | 1.811929 | -1.463501 | -0.740970 | 3.187024 | 0.673741 | -1.822140e-01 | 2.724862e-02 | -4.713878e-02 | -7.161555e-05 | 3 |
| **333** | -0.195039 | 1.811929 | -1.463501 | -0.740970 | -0.450729 | -1.698210 | -1.822036e-01 | 2.724284e-02 | -4.717100e-02 | -7.312064e-05 | 3 |
| **182** | -0.186973 | 0.829195 | 0.454428 | 0.796091 | -0.353722 | 0.357481 | -1.766622e-01 | 2.915473e-02 | 5.044172e-02 | -5.369756e-02 | |
| **205** | -0.149163 | -0.028852 | 0.480750 | -0.088470 | -0.207669 | 0.234527 | -1.378678e-01 | 1.740332e-02 | -6.528351e-02 | 2.060853e-03 | |
| **619** | -0.195039 | -0.789426 | 1.019478 | -0.740970 | -0.450729 | 0.199351 | -1.818933e-01 | 2.713172e-02 | -4.740327e-02 | -9.680981e-05 | 3 |
| **624** | -0.195039 | 0.366732 | -1.463501 | 2.003781 | -0.450729 | -0.116909 | -1.801225e-01 | 2.924881e-02 | -4.254675e-02 | 2.002322e-03 | 3 |
| **453** | -0.195039 | 1.811929 | -1.463501 | -0.740970 | -0.450729 | -1.698210 | -1.822117e-01 | 2.724841e-02 | -4.713931e-02 | -7.214275e-05 | 3 |
| **353** | -0.195039 | -1.078466 | 2.398911 | -0.740970 | -0.450729 | 1.464392 | -1.821565e-01 | 2.725226e-02 | -4.718472e-02 | -7.436708e-05 | 3 |
| **257** | -0.195039 | 1.811929 | -1.463501 | -0.740970 | 6.824777 | -1.698210 | -1.788919e-01 | 2.652303e-02 | -4.671080e-02 | 3.648985e-04 | 3 |
| **516** | -0.195039 | 1.811929 | -1.463501 | -0.740970 | 6.824777 | -1.698210 | -1.822067e-01 | 2.724622e-02 | -4.714949e-02 | -7.543941e-05 | 3 |
| **104** | 0.194679 | -1.357762 | 0.445337 | 2.191863 | -0.302808 | 0.334381 | -1.194811e-01 | 3.660933e-02 | -6.730388e-02 | -2.166522e-02 | 3 |
| **114** | -0.168671 | -0.157984 | 0.505425 | 0.266683 | -0.213908 | 0.315065 | -5.963937e-02 | 3.107895e-02 | 1.532542e-01 | 1.868665e-01 | 3 |
| **165** | -0.013548 | -0.355867 | 0.926033 | 1.111737 | -0.450729 | 0.673741 | -1.819640e-01 | 2.707665e-02 | -4.756597e-02 | -9.169617e-05 | 3 |
| **339** | -0.195039 | -2.523663 | -1.463501 | -0.740970 | -0.450729 | -1.698210 | -1.773345e-01 | 4.086087e-02 | 9.380572e-02 | 1.051013e-01 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **475** | -0.030979 | 0.121783 | 0.414700 | 0.096412 | -0.327415 | 0.311918 | -1.747560e-01 | 3.196033e-02 | -2.130046e-05 | -1.698441e-02 | 2 |
| **561** | 0.409931 | -0.139087 | 1.295365 | -0.535114 | -0.268841 | -1.105222 | -1.820628e-01 | 2.721775e-02 | -4.659597e-02 | 2.174508e-04 | 3 |
| **253** | -0.137423 | 0.779645 | 0.494353 | 0.992557 | -0.450729 | 0.548902 | -1.822053e-01 | 2.724647e-02 | -4.714741e-02 | -7.124630e-05 | 3 |

| | passion_score | malepercentage | weightedscore | Dependent | Senior | YoungAdult | region_pc1 | region_pc2 | region_pc3 | region_pc4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **21** | -0.195039 | -2.523663 | 3.117092 | -0.740970 | -0.450729 | 1.464392 | -1.821841e-01 | 2.722863e-02 | -4.719617e-02 | -8.276441e-05 | 3 |
| **314** | -0.091060 | 0.321569 | 0.258197 | 0.030991 | -0.337049 | 0.525494 | -1.820912e-01 | 2.722448e-02 | -4.720843e-02 | -7.393355e-05 | 3 |
| **460** | -0.110498 | -0.679978 | 0.554435 | 0.190507 | -0.236743 | 0.401843 | -1.818261e-01 | 2.707615e-02 | -4.699044e-02 | 1.354145e-04 | 3 |
| **161** | 0.059613 | -0.786138 | 0.610521 | 0.592496 | -0.199339 | 0.147812 | 8.521464e-18 | 3.769045e-18 | -1.281014e-17 | 4.936926e-19 | 6 |
| **277** | -0.195039 | -1.439765 | -1.463501 | -0.740970 | -0.450729 | -0.512234 | -1.821436e-01 | 2.723244e-02 | -4.688446e-02 | 4.385339e-05 | 3 |
| **192** | 0.168083 | -0.297926 | 0.347863 | -0.414030 | 0.121590 | -0.290847 | -1.818001e-01 | 2.711326e-02 | -4.676865e-02 | 2.209048e-04 | 3 |
| **386** | 0.024950 | -0.651476 | 0.469291 | 1.879020 | -0.120024 | -0.296602 | -1.821917e-01 | 2.723551e-02 | -4.717437e-02 | -8.658890e-05 | 3 |
| **414** | -0.195039 | -2.523663 | -1.463501 | 7.493283 | -0.450729 | -1.698210 | -1.789456e-01 | 3.191332e-02 | -2.218329e-02 | -8.362956e-03 | 4 |
| **492** | -0.115366 | -0.584487 | 0.638061 | 0.828242 | -0.239463 | 0.430397 | -4.624907e-02 | 2.241126e-01 | 3.155800e-01 | 3.459787e-02 | |
| **344** | -0.195039 | -0.572647 | -1.463501 | 0.559175 | -0.450729 | 1.048260 | -1.815693e-01 | 2.682776e-02 | -4.804028e-02 | -3.702010e-04 | 3 |
| **309** | -0.098244 | 0.858099 | 0.695300 | 0.411825 | -0.450729 | 1.148131 | -1.822140e-01 | 2.724862e-02 | -4.713878e-02 | -7.161555e-05 | 3 |
| **130** | -0.144988 | 0.345938 | 0.455478 | 1.447077 | -0.201003 | 0.444298 | 4.684330e-02 | -1.266712e-01 | 1.494517e-01 | 1.338378e-01 | |
| **99** | -0.170931 | -0.312675 | 0.527389 | -0.513849 | 0.127692 | -0.453866 | -1.289844e-01 | 2.583232e-02 | 1.044968e-02 | 1.462316e-02 | 1 |
| **373** | 0.002157 | -0.338202 | 0.464116 | 0.502846 | -0.144033 | 0.190463 | -1.806601e-01 | 2.667409e-02 | -4.896062e-02 | -6.655521e-04 | 3 |
| **87** | -0.104066 | -0.046182 | 0.841685 | -0.431412 | -0.122511 | -0.235804 | -1.740907e-01 | 3.949288e-02 | -3.329101e-02 | -1.987233e-03 | 2 |
| **459** | -0.195039 | 0.077692 | 0.191819 | -0.740970 | -0.086954 | -0.986625 | 3.331612e-01 | -1.436613e-01 | -6.254941e-01 | -2.049828e-01 | |
| **331** | 11.904359 | 1.811929 | -1.463501 | -0.740970 | -0.450729 | -1.698210 | -1.822187e-01 | 2.725086e-02 | -4.713206e-02 | -6.983581e-05 | 3 |
| **215** | -0.085045 | 1.417784 | -1.463501 | 0.007598 | -0.450729 | 1.751901 | -1.743961e-01 | 2.843560e-02 | 2.582719e-01 | -2.061555e-01 | |
| **467** | -0.088941 | -0.675039 | 0.731198 | 0.197370 | -0.218249 | 0.153693 | -1.822155e-01 | 2.725042e-02 | -4.713176e-02 | -7.047764e-05 | 3 |
| **121** | -0.195039 | -0.355867 | -1.463501 | -0.740970 | 3.187024 | -1.698210 | -1.821731e-01 | 2.723985e-02 | -4.722449e-02 | -1.183901e-04 | 3 |
| **615** | -0.195039 | -1.078466 | -1.463501 | 0.631405 | -0.450729 | -0.380459 | -1.690151e-01 | 2.251880e-02 | -6.226047e-02 | -5.074609e-03 | 2 |
| **20** | -0.195039 | -0.757311 | 0.446028 | 0.478919 | -0.181266 | -0.292609 | -1.813707e-01 | 2.711700e-02 | -4.692955e-02 | -1.905033e-05 | 3 |
| **71** | -0.195039 | 1.811929 | -1.463501 | -0.740970 | -0.450729 | -1.698210 | -1.822187e-01 | 2.725086e-02 | -4.713206e-02 | -6.983581e-05 | 3 |
| **106** | -0.155369 | 1.101176 | 0.386562 | 0.338932 | -0.450729 | 0.868163 | -1.806680e-01 | 2.703206e-02 | -4.715721e-02 | 2.170092e-06 | 3 |
| **271** | 0.036923 | 0.311210 | 0.351285 | 0.707187 | -0.277372 | 0.675535 | -7.232600e-02 | -7.368285e-02 | 2.670108e-01 | 2.415764e-01 | 1 |
| **436** | 0.025343 | 0.882874 | 0.251696 | 0.144434 | -0.424652 | 0.614230 | -1.755706e-01 | 2.559214e-02 | -4.714047e-02 | 9.512779e-04 | 2 |
| **102** | 0.071494 | -0.729089 | 0.701651 | 0.954735 | -0.163196 | 0.088704 | 1.339382e+01 | 2.028742e+01 | -1.442709e+00 | 6.766193e-01 | 6. |

527 rows × 17 columns

## Class Balance

In our analysis before, we know that most value of the dependent variable 'success' is 0, so there will be a strong bias toward predicting to artist failure. Here we use resample method to increase the size of 'success' and generate a new dataframe containing the increased training set which is called x_train_final. After the calculation, we set the value of n_samples as 450 and we can see that the ratio between success and failure is close to 1 after checking for the success value count.

In [78]:
```
from sklearn.utils import resample


train_set_fail = train_set[train_set.success==0]
train_set_success = train_set[train_set.success==1]

df_rs = resample(train_set_success, replace=True, n_samples=450, random_state=42)
x_train_final = pd.concat([train_set, df_rs])
x_train_final
x_train_final.success.value_counts()
```

Out[78]:
```
1    516
0    461
Name: success, dtype: int64
```

Here we split the x_train_final dataset into dependent variable and independent variables named x_final and y_final.

In [79]:
```
x_final=x_train_final[features2]
y_final=x_train_final['success']
```

We need to calculate the cross validation scores of different models, including decision tree, SGD, random forest, KNN and SVR. Comparing the scores of each model, we can see that the score of random forest is the highest. Thus, we select random forest to improve our result.

In [80]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVR
from sklearn.neighbors import KNeighborsClassifier
```

In [81]:
```
from sklearn.model_selection import cross_val_score
X =x_final.values
y = y_final.values

tree_clf = DecisionTreeClassifier()
FOLDS = 5
s_secision_tree = cross_val_score(tree_clf, X, y, cv=FOLDS, verbose=0)
print("tree_clf: " + str(s_secision_tree.mean()))

#bag_clf = BaggingClassifier()
#FOLDS = 5
#s_secision_tree = cross_val_score(estimator_decision_tree, X, y, cv=FOLDS, verbose=0)
#print("bag_clf: " + str(s_secision_tree.mean()))

SGD_clf=SGDClassifier()
FOLDS = 5
s_SGD = cross_val_score(SGD_clf, X, y, cv=FOLDS, verbose=0)
print("SGD_clf: " + str(s_SGD.mean()))

forest = RandomForestClassifier()
FOLDS = 5
s_forest = cross_val_score(forest, X, y, cv=FOLDS, verbose=0)
print("forest: " + str(s_forest.mean()))

KNN_classifier = KNeighborsClassifier()
FOLDS = 5
s_KNN = cross_val_score(KNN_classifier, X, y, cv=FOLDS, verbose=0)
print("KNN_classifier: " + str(s_KNN.mean()))

svm_reg_clf=LinearSVR()
FOLDS = 5
s_SVR = cross_val_score(svm_reg_clf, X, y, cv=FOLDS, verbose=0)
print("svm_reg_clf: " + str(s_SVR.mean()))
```

```
tree_clf: 0.9610985292203565
SGD_clf: 0.8157178185604581
forest: 0.9785031888585187
KNN_classifier: 0.9263100351425224
```

```
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:166: FutureWarning: max_iter an
d tol parameters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol
is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:166: FutureWarning: max_iter an
d tol parameters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol
is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:166: FutureWarning: max_iter an
d tol parameters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol
is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:166: FutureWarning: max_iter an
d tol parameters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol
is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:166: FutureWarning: max_iter an
d tol parameters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol
is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The default value of n_estim
ators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The default value of n_estim
ators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The default value of n_estim
ators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The default value of n_estim
ators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The default value of n_estim
ators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/svm/base.py:922: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/svm/base.py:922: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/svm/base.py:922: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/svm/base.py:922: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
svm_reg_clf: -1.9128310111905324
```

```
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/svm/base.py:922: ConvergenceWarning: Liblinear failed to converge,
increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

## 5. Improve Results

#### Hyper Parameter Tuning

It should be clear from your analysis above that you have selected a particular class of classifier as it gives the best result.

We can now tune the classifier (Hyper Parameter Tuning using Grid Search) to determine the optimal parameters.

**ACTION: Hyper Parameter Tuning**

Perform hyperparameter turing and demonstrate improved performance. Comment on any specific behaviour of your chosen classifier and set out the final structure and parameter settings.

We use grid search to find the best parameters of the random forest classifier here. From the output, we can see that the best parameters to get the best score include the max features 'log2' and when number of estimators equals to 200.

In [82]:
```
# SELECTED MODEL:
# Random forest

# GRIDSEARCH
model = RandomForestClassifier(n_jobs=-1,max_features= 'sqrt' ,n_estimators=50, oob_score = True)
from sklearn.model_selection import GridSearchCV, cross_val_score
param_grid = {
        'n_estimators': [100,125,150,200,250],
        'max_features': ['auto', 'sqrt', 'log2']
    }

CV_rfc = GridSearchCV(estimator=model, param_grid=param_grid, cv= 5)
CV_rfc.fit(x_final, y_final)

print('GRIDSEARCH')
print('-'*10)
print('Best Parameters: ',CV_rfc.best_params_)
print('Best Score: ',CV_rfc.best_score_)
```

```
GRIDSEARCH
----------
Best Parameters:  {'max_features': 'sqrt', 'n_estimators': 100}
Best Score:  0.9856704196519959
```

Finally, we can use the final model(random forest) to predict the test set and it can be noticed that the accuracy score of the predction is 87.12%.

In [83]:
```
# PRINT FINAL MODEL RESULT
finalmodel = RandomForestClassifier(n_jobs=-1,max_features= CV_rfc.best_params_['max_features'] ,n_estimators=CV_rfc.best_params_['n_estimators'], oob_score = True)
finalmodel.fit(x_final, y_final)
```

Out[83]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='sqrt', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
            oob_score=True, random_state=None, verbose=0, warm_start=False)
```

In [84]:
```
# VALIDATION SET SCORE
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

y_pred = finalmodel.predict(x_test)
train_score = finalmodel.score(x_final, y_final)
validation_score = accuracy_score(y_test, y_pred)
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)

print(validation_score)
```

```
[[106   9]
 [  8   9]]
0.8712121212121212
```

**Ensemble model**

Now that we have a good classifier working, explore building an ensemble model.

This might include Boostrapping, Bagging, Boosting and Stacking.

Refer to "L6 - Trees and Forests, Ensemble Learning 1" from Week 6 as necessary.

**ACTION: Ensemble modeling**

Build an ensemble model and demonstrate improved performance. Comment on any specific behaviour of your chosen classifier and set out the final structure and parameter settings.

Divide this work up among the different members of your team and then compare and comment on the performance of various approaches.

In the part of ensemble modeling, gradient boosting classifier, bagging classifier, adaboosting classifier, and voting classifier are applied. Fitting them with X_final and Y_final and printing out the accuracy score of each model. We can see that accuracy score of each models have been improved and voting classifier is the highest now.

In [85]:

```
# Ensemble models

#Bagging Classifier
from sklearn.ensemble import BaggingClassifier
bagging_clf = BaggingClassifier(DecisionTreeClassifier(random_state=42), n_estimators=500,
                                              max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
bagging_clf.fit(x_final, y_final)
y_pred_bagging = bagging_clf.predict(x_test)
print(accuracy_score(y_test, y_pred_bagging))
```

```
0.8560606060606061
```

In [86]:

```
from sklearn.ensemble import AdaBoostClassifier
ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=2,random_state=42), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(x_final, y_final)
y_pred_ada=ada_clf.predict(x_test)
print(accuracy_score(y_test, y_pred_ada))
```

```
0.8939393939393939
```

In [87]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier


log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier(n_jobs=-1,max_features= CV_rfc.best_params_['max_features'] ,n_estimators=CV_rfc.best_params_['n_es
timators'], oob_score = True)

svm_clf = SVC()
KNN_clf = KNeighborsClassifier(n_neighbors=5)
SGD_clf=SGDClassifier(random_state=42)
#
tree_clf=DecisionTreeClassifier(max_depth=2, random_state=42)
#

voting_clf = VotingClassifier(estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf),('knn',KNN_clf),('sgd',SGD_clf),('tf',t
ree_clf)],
voting='hard')
voting_clf.fit(x_final, y_final)

from sklearn.metrics import accuracy_score
log_clf.fit(x_final, y_final)
y_predlog=log_clf.predict(x_test)
print('logisticregression', accuracy_score(y_test, y_predlog))

rnd_clf .fit(x_final, y_final)
y_predrnd=rnd_clf.predict(x_test)
print('randomforest', accuracy_score(y_test, y_predrnd))

svm_clf.fit(x_final, y_final)
y_predsvm=svm_clf.predict(x_test)
print('SVM', accuracy_score(y_test, y_predsvm))

KNN_clf.fit(x_final, y_final)
y_predknn=KNN_clf.predict(x_test)
print('KNN', accuracy_score(y_test, y_predknn))

SGD_clf.fit(x_final, y_final)
y_predsgd=SGD_clf.predict(x_test)
print('SGD', accuracy_score(y_test, y_predsgd))

tree_clf.fit(x_final, y_final)
y_predtree=tree_clf.predict(x_test)
print('Tree', accuracy_score(y_test, y_predtree))

voting_clf.fit(x_final, y_final)
y_predvoting=voting_clf.predict(x_test)
print('Voting', accuracy_score(y_test, y_predvoting))
```

```
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:166: FutureWarning: max_iter and tol parameters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

logisticregression 0.8409090909090909
randomforest 0.8863636363636364
SVM 0.7651515151515151
KNN 0.75
SGD 0.7575757575757576
Tree 0.7803030303030303

/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:166: FutureWarning: max_iter and tol parameters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

Voting 0.8939393939393939

/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
/opt/anaconda/envs/Python3/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:166: FutureWarning: max_iter and tol parameters have been added in SGDClassifier in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  FutureWarning)
```

## 6. Present Results

### Confusion Matrix

To get a better idea of the quality of our predictions, we can plot a confusion matrix and ROC curve.

A confusion matrix is a technique for summarizing the performance of a classification algorithm that allows visualization of the performance of an algorithm.

Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).

The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives you insight not only into the errors being made by your classifier but more importantly the types of errors that are being made.

**ACTION: Confusion matrix**

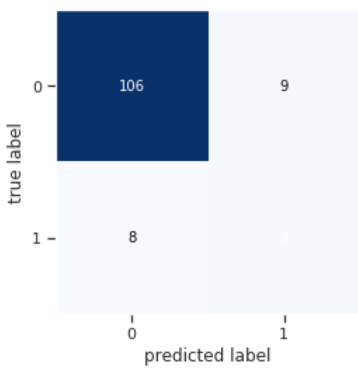
Comment on the performance of your final algorithm. Repeat analysis from earlier in the Notebook if necessary.

Explain confusion matrix results, calculate accuracy and precision etc.

In [88]:
```
# Confusion Matrix
# install the plot package
!pip install mlxtend
```

```
Collecting mlxtend
  Using cached https://files.pythonhosted.org/packages/16/e6/30e50ed9c053a1530c83149090e1f5fd9fccc8503dca2ecce1bb52f34de0/mlxtend-0.15.0.0-py2.py3-none-any.whl
Requirement already satisfied: pandas>=0.17.1 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from mlxtend) (0.23.4)
Requirement already satisfied: scipy>=0.17 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from mlxtend) (1.1.0)
Requirement already satisfied: numpy>=1.10.4 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from mlxtend) (1.15.4)
Requirement already satisfied: scikit-learn>=0.18 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from mlxtend) (0.20.1)
Requirement already satisfied: matplotlib>=1.5.1 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from mlxtend) (3.0.2)
Requirement already satisfied: setuptools in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from mlxtend) (40.6.3)
Requirement already satisfied: python-dateutil>=2.5.0 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from pandas>=0.17.1->mlxtend) (2.7.5)
Requirement already satisfied: pytz>=2011k in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from pandas>=0.17.1->mlxtend) (2018.7)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (1.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (2.3.0)
Requirement already satisfied: six>=1.5 in /opt/anaconda/envs/Python3/lib/python3.6/site-packages (from python-dateutil>=2.5.0->pandas>=0.17.1->mlxtend) (1.12.0)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.15.0.0
```

In [89]:
```
# Plot Confusion Matrix
from mlxtend.plotting import plot_confusion_matrix

fig, ax = plot_confusion_matrix(conf_mat=confusion_matrix)
plt.show()
```




## ROC Curve

Receiver Operating Characteristic (ROC) curves show the ability of the model to classify subjects correctly across a range of decision thresholds, i.e. it plots the True Positive Rate vs. False Positive Rate at every probability threshold.

The AUC summarizes the results of an ROC – it is the probability that a randomly chosen ‘success’ example has a higher probability of being a success than a randomly chosen ‘failure’ example. A random classification would yield an AUC of 0.5, and a perfectly accurate one would yield 1.

**ACTION: ROC Curve**

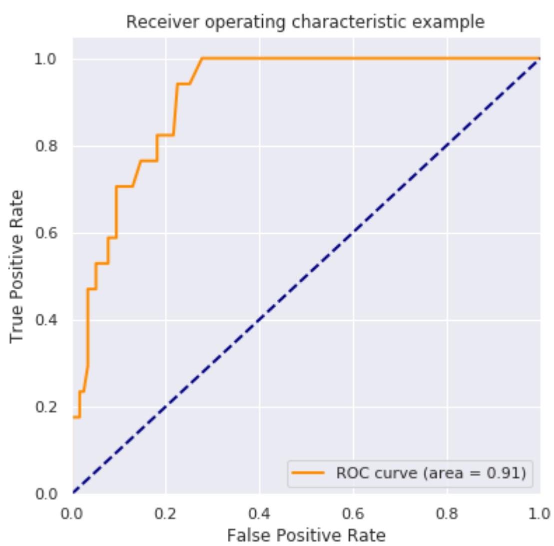
Comment on the performance of your final algorithm. Repeat analysis from earlier in the Notebook if necessary.

Explain any observations about the ROC results.

Here we plot the ROC curve.

In [90]:
```
# ROC curve

from sklearn.metrics import roc_curve, auc

# Compute predicted probabilities: y_pred_prob
y_pred_rf_prob = finalmodel.predict_proba(x_test)[:,1]

# Generate ROC curve values: fpr, tpr, thresholds
fpr, tpr, threholds = roc_curve(y_test, y_pred_rf_prob)

roc_auc = auc(fpr,tpr)

plt.figure()
lw = 2
plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
# Plot classifier ROC
```

<Figure size 432x288 with 0 Axes>




Now that you have a validated model, we can potentially analyze the features of the model, to understand which ones have had the most impact on predicting an artist's success.

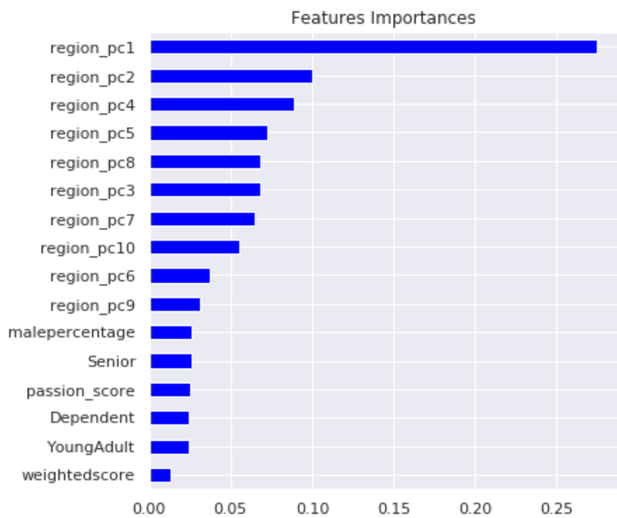
To do this, we can plot the feature importance as determined by the classifier:

**ACTION: Feature importance**

Where possible, comment on the feature selection and performance of your final algorithm. Repeat analysis from earlier in the Notebook if necessary. Explain any observations about the sensitivity of your final analysis.

At last, we analyze the feature importance and create a pd.Series of features importances. Plotting the features importance and sort the importance level from highest to lowest. We can see that region_pca 1 is significantly higher than others and the total weighted score is the lowest one.

In [91]:
```
# Feature importance analysis
import seaborn as sns

# Create a pd.Series of features importances
importances = pd.Series(data=finalmodel.feature_importances_,
                        index=x_final.columns)

# Sort importances
importances_sorted = importances.sort_values()

# Draw a horizontal barplot of importances_sorted
plt.figure(figsize = (6,6))
sns.set(font_scale=1.0)
importances_sorted.plot(kind='barh', color='blue')
plt.title('Features Importances')
plt.show()
```




## Summary

In this analysis, we firstly create dependent variable which is the success or not for each artist. Then we did the feature engineering part which includes Artist Features, Playlist Features and User-base features. Principle Component Analysis (PCA) is did after that. We combined all of our features that we generated into a new dataframe that can be processed by a machine learning algorithm. We work with the missing values of variables by filling them to the average or 0 and normalize every independent variables. The multicollinearity has been checked and the highly correlated independent variables has been dropped. Before building the final model, we split the data into train and validation set and resample in class balance part. After calculating the cross validation scores of different models, we select random forest with the highest score to improve the result. We use grid search to find the best parameters of the random forest classifier. Then, the performance of model has been improved by ensemble modeling. Finally, to get a better idea of the quality of our predictions, we plot a confusion matrix and ROC curve and also analyze the feature importances.

## Tips completing the coursework

- **Sherlock** - You are free to run the code on your local machine, but if training timings and memory become an issue then use Sherlock to complete the coursework. Technical support for using Sherlock will be provided as necessary. https://sherlockml.com (https://sherlockml.com)
- **Kanban** - Assess the different potential work packages and break the overall objectitves into a set of tasks and queue them up in the backlog column of the Kanban board. Create new tasks as and when necessary during the course of your analysis.
- **Fast First Pass**. Make a first-pass through the project steps as fast as possible. This will give you confidence that you have all the parts that you need and a baseline from which to improve.
- **Cycles**. The process in not linear but cyclic. You will loop between steps, and probably spend most of your time in tight loops between steps 3-4 or 3-4-5 until you achieve a level of accuracy that is sufficient or you run out of time. The write up in the final submitted Notebook in more linear - you do not need to include all of your work, ie. including all dead-ends, and it should be consise and consistent.
- **Attempt Every Step**. It is easy to skip steps, especially if you are not confident or familiar with the tasks of that step. Try and do something at each step in the process, even if it does not improve accuracy. You can always build upon it later. Don’t skip steps, just reduce their contribution.
- **Ratchet Accuracy**. The goal of the project is to achieve good model performance (which ever metric you use to measure this). Every step contributes towards this goal. Treat changes that you make as experiments that increase accuracy as the golden path in the process and reorganize other steps around them. Performance is a ratchet that can only move in one direction (better, not worse).
- **Adapt As Needed**. Modify the steps as you need on a project, especially as you become more experienced with the template. Blur the edges of tasks, such as steps 4-5 to best serve model accuracy. The final submitted Notebook does not need to preserve the current sections and structure if you think something else is more apporpirate.

In [ ]: