GROUP COURSEWORK

MSIN0097 Predictive Analytics

Student Name: Jiarui Xin

ID: 19058391

Email: jiarui.xin.19@ucl.ac.uk

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

There should be around 100 words per ACTION cell, but use the wordcount over the duration of the Notebook at your discretion.

- **Please use the below green cell, when writing your comments in markup.**
- **Please feel free to add extra code cells in the notebook if needed.**

Title (Optional)

Content

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 and 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 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

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

# WEEK 2

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

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

# WEEK 3

# 3. Prepare Data
# a) Data Cleaning
# b) Feature Selection
# c) Data Transforms (PCA, missing values, multi-collinearity, normalizing, class balance)

# WEEK 5

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

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

ACTION: Guidance

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

---

WEEK 2

Submission Deadline: 30.01.2020

1. Prepare the problem

Run your code on Faculty. 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')
#sfs.get('/input/cleaned_data.csv', 'cleaned_data.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
```

In [3]:

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import os
import glob
import collections
import networkx as nx
import seaborn
import warnings
import random
```

In [6]:

```
#Import data
import csv
data=pd.read_csv('cleaned_data.csv')
data
```

Out[6]:

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	day	log_time	mobile	track_id	
0	0	9	('small_artists_2016.csv', 9)	10	20160510T12:15:00	True	8f1924eab3804f308427c31d925c1b3f	USAT216(
1	1	19	('small_artists_2016.csv', 19)	10	20160510T12:15:00	True	8f1924eab3804f308427c31d925c1b3f	USAT216(
2	2	29	('small_artists_2016.csv', 29)	10	20160510T14:00:00	True	8f1924eab3804f308427c31d925c1b3f	USAT216(
3	3	39	('small_artists_2016.csv', 39)	10	20160510T10:45:00	True	8f1924eab3804f308427c31d925c1b3f	USAT216(
4	4	49	('small_artists_2016.csv', 49)	10	20160510T10:15:00	True	8f1924eab3804f308427c31d925c1b3f	USAT216(
...	...	...	...	...	...	...	...	
3805494	3805494	38054949	1301551	10	20170610T10:30:00	True	4cb959db5be04d2fa5ca4c137b651a99	GBAHS16(

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	day	log_time	mobile	track_id	
3805495	3805495	38054959	1301561	10	20170710T18:15:00	True	4cb959db5be04d2fa5ca4c137b651a99	GBAHS160
3805496	3805496	38054969	1301571	10	20170710T18:00:00	True	4cb959db5be04d2fa5ca4c137b651a99	GBAHS160
3805497	3805497	38054979	1301581	10	20170710T14:15:00	False	4cb959db5be04d2fa5ca4c137b651a99	GBAHS160
3805498	3805498	38054989	1301591	10	20170710T12:00:00	True	4cb959db5be04d2fa5ca4c137b651a99	GBAHS160

3805499 rows × 45 columns

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 Faculty 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 [7]:

```
# %%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 [9]:

```
%%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'])
```

```
rows: 3805499
CPU times: user 21.9 s, sys: 2.46 s, total: 24.3 s
Wall time: 24.3 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()**, **data.info()**, etc.

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.

In [10]:

```
data.head()
```

Out[10]:

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	day	log_time	mobile	track_id	isrc	7
0	0	9	('small_artists_2016.csv', 9)	10	20160510T12:15:00	True	8f1924eab3804f308427c31d925c1b3f	USAT21600547	7
1	1	19	('small_artists_2016.csv', 19)	10	20160510T12:15:00	True	8f1924eab3804f308427c31d925c1b3f	USAT21600547	7
2	2	29	('small_artists_2016.csv', 29)	10	20160510T14:00:00	True	8f1924eab3804f308427c31d925c1b3f	USAT21600547	7
3	3	39	('small_artists_2016.csv', 39)	10	20160510T10:45:00	True	8f1924eab3804f308427c31d925c1b3f	USAT21600547	7
4	4	49	('small_artists_2016.csv', 49)	10	20160510T10:15:00	True	8f1924eab3804f308427c31d925c1b3f	USAT21600547	7

5 rows × 45 columns

In [11]:

```
data.info
```

Out[11]:

```
<bound method DataFrame.info of          Unnamed: 0  Unnamed: 0.1                 Unnamed: 0.1.1
day  \
0                 0             9   ('small_artists_2016.csv', 9)    10
1                 1            19  ('small_artists_2016.csv', 19)    10
2                 2            29  ('small_artists_2016.csv', 29)    10
3                 3            39  ('small_artists_2016.csv', 39)    10
4                 4            49  ('small_artists_2016.csv', 49)    10
...             ...           ...                             ...   ...
3805494     3805494      38054949                         1301551    10
3805495     3805495      38054959                         1301561    10
```

```
3805496     3805496      38054969                          1301571   10
3805497     3805497      38054979                          1301581   10
3805498     3805498      38054989                          1301591   10

                 log_time  mobile                          track_id  \
0        20160510T12:15:00    True  8f1924eab3804f308427c31d925c1b3f
1        20160510T12:15:00    True  8f1924eab3804f308427c31d925c1b3f
2        20160510T14:00:00    True  8f1924eab3804f308427c31d925c1b3f
3        20160510T10:45:00    True  8f1924eab3804f308427c31d925c1b3f
4        20160510T10:15:00    True  8f1924eab3804f308427c31d925c1b3f
...                    ...     ...                             ...
3805494  20170610T10:30:00    True  4cb959db5be04d2fa5ca4c137b651a99
3805495  20170710T18:15:00    True  4cb959db5be04d2fa5ca4c137b651a99
3805496  20170710T18:00:00    True  4cb959db5be04d2fa5ca4c137b651a99
3805497  20170710T14:15:00   False  4cb959db5be04d2fa5ca4c137b651a99
3805498  20170710T12:00:00    True  4cb959db5be04d2fa5ca4c137b651a99

                isrc           upc       artist_name  ... hour minute week  \
0        USAT21600547  7.567991e+10  Sturgill Simpson  ...   12     15   19
1        USAT21600547  7.567991e+10  Sturgill Simpson  ...   12     15   19
2        USAT21600547  7.567991e+10  Sturgill Simpson  ...   14      0   19
3        USAT21600547  7.567991e+10  Sturgill Simpson  ...   10     45   19
4        USAT21600547  7.567991e+10  Sturgill Simpson  ...   10     15   19
...               ...           ...               ...  ...  ...    ...  ...
3805494  GBAHS1600395  1.902959e+11        Anne-Marie  ...   10     30   23
3805495  GBAHS1600395  1.902959e+11        Anne-Marie  ...   18     15   28
3805496  GBAHS1600395  1.902959e+11        Anne-Marie  ...   18      0   28
3805497  GBAHS1600395  1.902959e+11        Anne-Marie  ...   14     15   28
3805498  GBAHS1600395  1.902959e+11        Anne-Marie  ...   12      0   28

        month  year        date weekday  weekday_name playlist_id  \
0           5  2016  2016-05-10       1       Tuesday         NaN
1           5  2016  2016-05-10       1       Tuesday         NaN
2           5  2016  2016-05-10       1       Tuesday         NaN
3           5  2016  2016-05-10       1       Tuesday         NaN
4           5  2016  2016-05-10       1       Tuesday         NaN
...       ...   ...         ...     ...           ...         ...
3805494     6  2017  2017-06-10       5      Saturday         NaN
3805495     7  2017  2017-07-10       0        Monday         NaN
3805496     7  2017  2017-07-10       0        Monday         NaN
3805497     7  2017  2017-07-10       0        Monday         NaN
3805498     7  2017  2017-07-10       0        Monday         NaN

        playlist_name
0                 NaN
1                 NaN
2                 NaN
3                 NaN
4                 NaN
...               ...
3805494           NaN
3805495           NaN
3805496           NaN
3805497           NaN
3805498           NaN

[3805499 rows x 45 columns]>
```

In [12]:

```
data.describe
```

Out[12]:

```
<bound method NDFrame.describe of          Unnamed: 0  Unnamed: 0.1                  Unnamed: 0.1.1
day  \
0                 0             9   ('small_artists_2016.csv', 9)   10
1                 1            19  ('small_artists_2016.csv', 19)   10
2                 2            29  ('small_artists_2016.csv', 29)   10
3                 3            39  ('small_artists_2016.csv', 39)   10
4                 4            49  ('small_artists_2016.csv', 49)   10
...             ...           ...                             ...  ...
3805494     3805494      38054949                         1301551   10
3805495     3805495      38054959                         1301561   10
3805496     3805496      38054969                         1301571   10
3805497     3805497      38054979                         1301581   10
```

```
3805497     3805497      38054979                          1301581   10
3805498     3805498      38054989                          1301591   10

                 log_time  mobile                          track_id  \
0        20160510T12:15:00    True  8f1924eab3804f308427c31d925c1b3f
1        20160510T12:15:00    True  8f1924eab3804f308427c31d925c1b3f
2        20160510T14:00:00    True  8f1924eab3804f308427c31d925c1b3f
3        20160510T10:45:00    True  8f1924eab3804f308427c31d925c1b3f
4        20160510T10:15:00    True  8f1924eab3804f308427c31d925c1b3f
...                    ...     ...                               ...
3805494  20170610T10:30:00    True  4cb959db5be04d2fa5ca4c137b651a99
3805495  20170710T18:15:00    True  4cb959db5be04d2fa5ca4c137b651a99
3805496  20170710T18:00:00    True  4cb959db5be04d2fa5ca4c137b651a99
3805497  20170710T14:15:00   False  4cb959db5be04d2fa5ca4c137b651a99
3805498  20170710T12:00:00    True  4cb959db5be04d2fa5ca4c137b651a99

                isrc           upc       artist_name  ... hour minute week  \
0        USAT21600547  7.567991e+10  Sturgill Simpson  ...   12     15   19
1        USAT21600547  7.567991e+10  Sturgill Simpson  ...   12     15   19
2        USAT21600547  7.567991e+10  Sturgill Simpson  ...   14      0   19
3        USAT21600547  7.567991e+10  Sturgill Simpson  ...   10     45   19
4        USAT21600547  7.567991e+10  Sturgill Simpson  ...   10     15   19
...               ...           ...               ...  ...  ...    ...  ...
3805494  GBAHS1600395  1.902959e+11        Anne-Marie  ...   10     30   23
3805495  GBAHS1600395  1.902959e+11        Anne-Marie  ...   18     15   28
3805496  GBAHS1600395  1.902959e+11        Anne-Marie  ...   18      0   28
3805497  GBAHS1600395  1.902959e+11        Anne-Marie  ...   14     15   28
3805498  GBAHS1600395  1.902959e+11        Anne-Marie  ...   12      0   28

        month  year        date weekday  weekday_name playlist_id  \
0           5  2016  2016-05-10       1       Tuesday         NaN
1           5  2016  2016-05-10       1       Tuesday         NaN
2           5  2016  2016-05-10       1       Tuesday         NaN
3           5  2016  2016-05-10       1       Tuesday         NaN
4           5  2016  2016-05-10       1       Tuesday         NaN
...       ...   ...         ...     ...           ...         ...
3805494     6  2017  2017-06-10       5      Saturday         NaN
3805495     7  2017  2017-07-10       0        Monday         NaN
3805496     7  2017  2017-07-10       0        Monday         NaN
3805497     7  2017  2017-07-10       0        Monday         NaN
3805498     7  2017  2017-07-10       0        Monday         NaN

        playlist_name
0                 NaN
1                 NaN
2                 NaN
3                 NaN
4                 NaN
...               ...
3805494           NaN
3805495           NaN
3805496           NaN
3805497           NaN
3805498           NaN

[3805499 rows x 45 columns]>
```

In [13]:

```
data.isnull().sum()
```

Out[13]:

```
Unnamed: 0                0
Unnamed: 0.1              0
Unnamed: 0.1.1            0
day                       0
log_time                  0
mobile                    0
track_id                  0
isrc                      4
upc                       0
artist_name               0
track_name                0
album_name                0
customer_id               0
```

```
postal_code          1352181
access                     0
country_code               0
gender                 40422
birth_year             10021
filename                   0
region_code           261956
referral_code        3805499
partner_name         3378646
financial_product    2329099
user_product_type      22992
offline_timestamp    3805499
stream_length              0
stream_cached        3805499
stream_source              0
stream_source_uri    2761628
stream_device              0
stream_os                  0
track_uri                  0
track_artists              0
source               3805499
DateTime                   0
hour                       0
minute                     0
week                       0
month                      0
year                       0
date                       0
weekday                    0
weekday_name               0
playlist_id          2761628
playlist_name        2826389
dtype: int64
```

In [14]:

```
data.artist_name.nunique()
```

Out[14]:

```
661
```

In [15]:

```
print('The number of free user is {}'.format(sum(data.access=="free")))
print('The number of premium user is {}'.format(sum(data.access=="premium")))
print('The number of free user is {}'.format(sum(data.access=="basic-desktop")))
```

```
The number of free user is 1124692
The number of premium user is 2676048
The number of free user is 4753
```

In [16]:

```
print('The number of user using Android is {}'.format(sum(data.stream_os=="Android")))
print('The number of user using iOS is {}'.format(sum(data.stream_os=="iOS")))
print('The number of user using Windows is {}'.format(sum(data.stream_os=="Windows")))
print('The number of user using Mac is {}'.format(sum(data.stream_os=="Mac")))
```

```
The number of user using Android is 696625
The number of user using iOS is 2232632
The number of user using Windows is 358517
The number of user using Mac is 210700
```

Inspect the data

Learn more about the basic information of the statistical data and the column names, so as to find out the useful information. Find missing values and interested values for our next step.

EXPLORATORY ANALYSIS AND PLOTS

Now look at the data set in more detail.

> **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.

In [17]:

```
#my interest column name
my_list=["stream_length","log_time","artist_name","year","mobile","access","playlist_id","playlist_
name","track_name","isrc","customer_id","birth_year","region_code","gender","stream_source_uri"]
data1=data[my_list]
```

In [18]:

```
#look at the first five rows
data1.head()
```

Out[18]:

	stream_length	log_time	artist_name	year	mobile	access	playlist_id	playlist_name	track_name	isrc	
0	277.0	20160510T12:15:00	Sturgill Simpson	2016	True	free	NaN	NaN	Call To Arms	USAT21600547	6c022
1	53.0	20160510T12:15:00	Sturgill Simpson	2016	True	free	NaN	NaN	Call To Arms	USAT21600547	6c022
2	326.0	20160510T14:00:00	Sturgill Simpson	2016	True	premium	NaN	NaN	Call To Arms	USAT21600547	35
3	330.0	20160510T10:45:00	Sturgill Simpson	2016	True	premium	NaN	NaN	Call To Arms	USAT21600547	c3f2b
4	90.0	20160510T10:15:00	Sturgill Simpson	2016	True	premium	NaN	NaN	Call To Arms	USAT21600547	6a06a

Gender

In [19]:

```
data1['gender'].value_counts() #look at how many female and male
```

Out[19]:

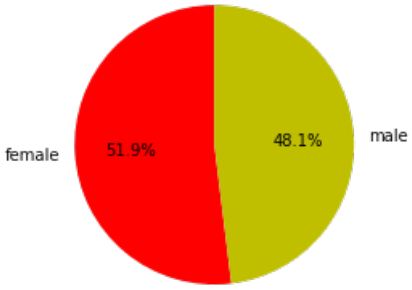
```
female    1955719
male      1809358
Name: gender, dtype: int64
```

In [20]:

```
female_male=[1955719,1809358]
user=['female','male']
colors=['r','y']
plt.pie(female_male, labels=user, colors=colors,startangle=90,autopct='%.1f%%')
plt.show
```

Out[20]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



The Gender ratio of users

We want to see the gender ratio of users. From the pie chart above, we can see that 51.9% of users are female and 48.1% are male.

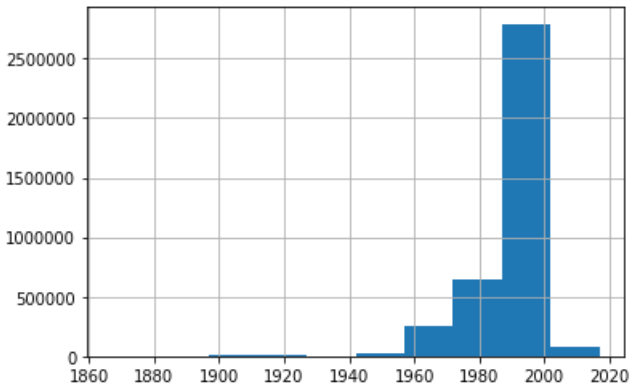
The Age of User Distribution

In [21]:

```
data1['birth_year'].hist()
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ffd8a3668>
```



In [42]:

```
#add a new column
import warnings
warnings.filterwarnings("ignore")
data1['user_age']=2020-data1['birth_year']
data1['user_age']
```

Out[42]:

```
0          52.0
1          52.0
2          25.0
3          28.0
4          41.0
           ...
3805494    30.0
3805495    39.0
3805496    21.0
3805497    33.0
3805498    28.0
Name: user_age, Length: 3805499, dtype: float64
```

In [41]:

```
new_cloumn=["stream_length","log_time","artist_name","year","mobile","access","playlist_id","playli
st_name","track_name","isrc","customer_id","birth_year","region_code","gender","stream_source_uri"
,"user_age"]
data_new=data1[new_cloumn]
data_new.head()
```

Out[41]:

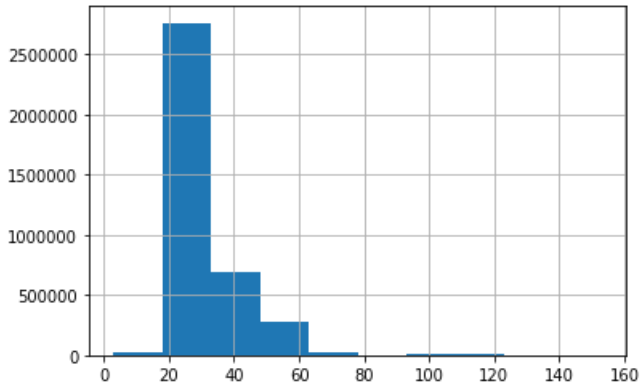
	stream_length	log_time	artist_name	year	mobile	access	playlist_id	playlist_name	track_name	isrc	
0	277.0	20160510T12:15:00	Sturgill Simpson	2016	True	free	NaN	NaN	Call To Arms	USAT21600547	6c022
1	53.0	20160510T12:15:00	Sturgill Simpson	2016	True	free	NaN	NaN	Call To Arms	USAT21600547	6c022
2	326.0	20160510T14:00:00	Sturgill Simpson	2016	True	premium	NaN	NaN	Call To Arms	USAT21600547	35
3	330.0	20160510T10:45:00	Sturgill Simpson	2016	True	premium	NaN	NaN	Call To Arms	USAT21600547	c3f2b
4	90.0	20160510T10:15:00	Sturgill Simpson	2016	True	premium	NaN	NaN	Call To Arms	USAT21600547	6a06a

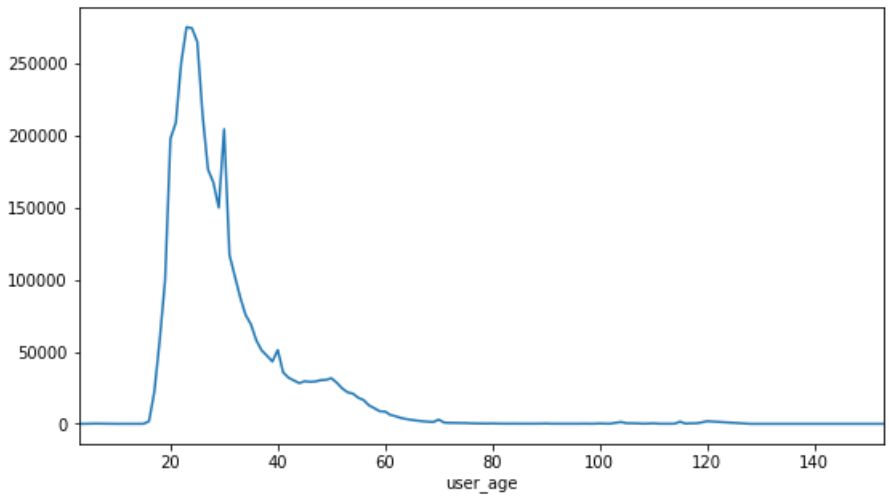
In [25]:

```
data_new['user_age'].hist()
plt.figure(figsize=(9,5))
X1=data_new.groupby("user_age")['customer_id'].count()
X1.plot()
plt.show
```

Out[25]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```





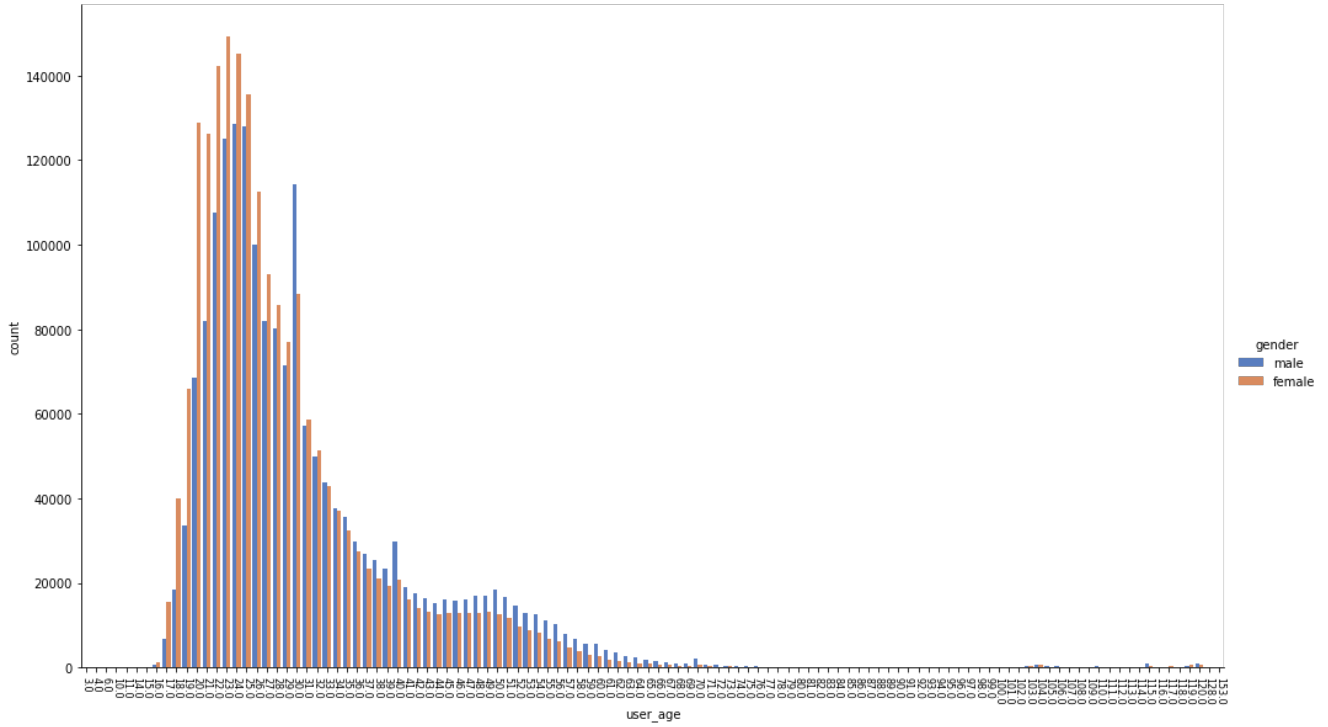
Users age

From the bar chart, we can see that most users were born between 1990 and 2000, which means that most users are between 20 and 30 years old.Spotify is very popular with young people.

The relatinship between the gender and age of users

In [26]:

```
# Birth year by gender
seaborn.catplot(x="user_age", hue="gender",
            data=data_new[data_new['customer_id'].isin(data_new['customer_id'].unique())], kind="co
unt",
            height=8.20, aspect=14/8.27, palette="muted")
plt.xticks(rotation=270, fontsize=8)
plt.show()
```



gender and age

This graph shows that there are more female users than male users in the lower age bands, but the situation reverses in higher age bands.

mobile VS non-mobile user

In [22]:

```
data1['mobile'].value_counts()
```

Out[22]:

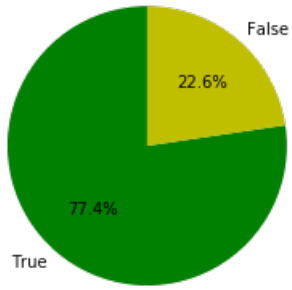
```
True     2943869
False     861630
Name: mobile, dtype: int64
```

In [23]:

```
moble_not=[2943869,861630]
user=['True','False']
colors=['g','y']
plt.pie(moble_not, labels=user, colors=colors,startangle=90,autopct='%.1f%%')
plt.show
```

Out[23]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



Mobile

Mobile phones are more popular because they are easy to carry.

Paying users and others

In [27]:

```
data1['access'].value_counts()
```

Out[27]:

```
premium          2676048
free             1124692
basic-desktop       4753
deleted                6
Name: access, dtype: int64
```

In [28]:

```
#deal with "delected"
data_access = data[data.access != "deleted"]
access = data_access["access"].value_counts()
access
```

Out[28]:

```
premium          2676048
free             1124692
basic-desktop       4753
Name: access, dtype: int64
```

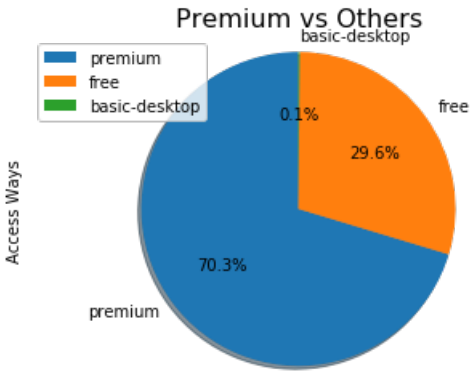
In [29]:

```
# Plot access pie chart
# Add labels
labels = ["premium", "free", "basic-desktop"]
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 Others", fontsize=16) # Add title
access_pie.axis("equal")  # Equal aspect ratio ensures that pie is drawn as a circle.
```

Out[29]:

```
(-1.1134317323758902,
 1.1130837153020834,
```

```
-1.1178022984784375,
1.1008477284989733)
```



Premium vs Others

From the pie above, 70.3% of customers are premium, Others are belong to free users. So there are huge potential to increase the revenue of spotify.

premium service paying through different methods

In [31]:

```
# the number of customers using different payment methods for premium service
data_F = data[data.financial_product != "deleted"]
# Drop the missing value
financial_P = data_F["financial_product"].value_counts()
# Number of customers with premium service paying it through different methods
financial_P
```

Out[31]:

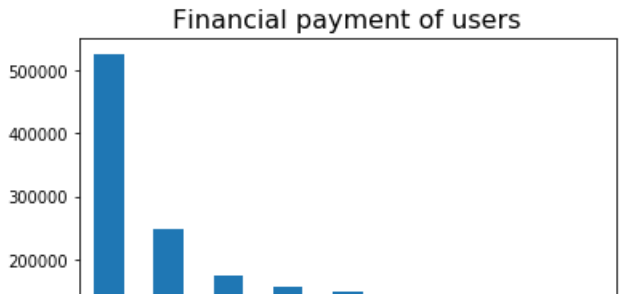
```
student          525367
standalone       248559
hardbundle       176395
family-sub       156374
promo            150833
family-master     98953
30Y               64478
7N                54628
employee            807
Name: financial_product, dtype: int64
```

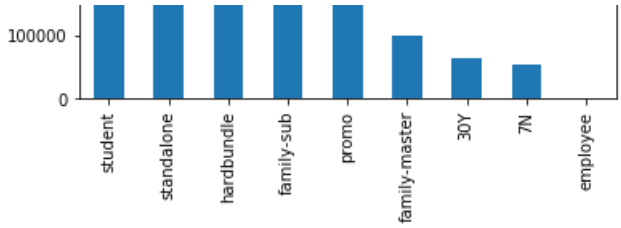
In [32]:

```
# Plot payment method bar chart
financial_bar = data_F["financial_product"].value_counts().plot(kind="bar")
financial_bar.set_title("Financial payment of users", fontsize=16)
# Add title
```

Out[32]:

```
Text(0.5, 1.0, 'Financial payment of users')
```





Payment methods

In terms of the plan users subscribe to, most paid users subscribe to 'student' plan. Firstly, such subscription profile corresponds to the age profile of Spotify users. Secondly, this is potentially an evidence that users are very sensitive to price.

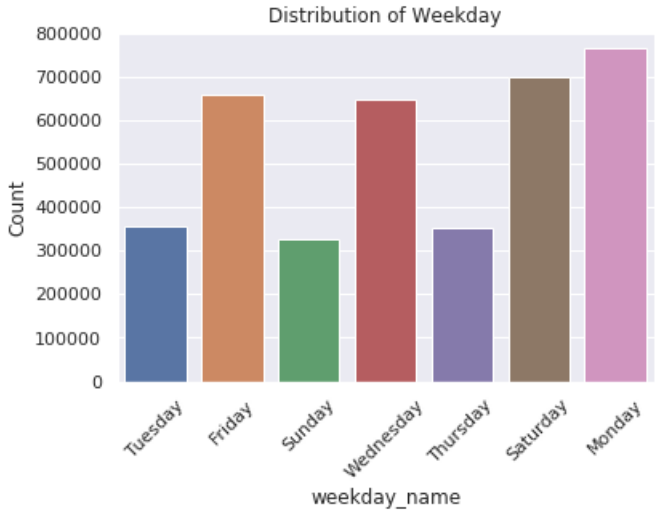
Distribution of Weekend

In [9]:

```
#Creating functions for drawing histogram plots

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

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

In [11]:

```
categorical_plot('weekday_name','Weekday','Count','Distribution of Weekday')
```



Distribution of Weekend

One interesting fact we observe from the dataset was that people listen to music more on certain days of a week. People listen to music more on Monday, Wednesday, Friday and Saturday significantly more than on the rest of the week.

The number of songs per year

In [68]:

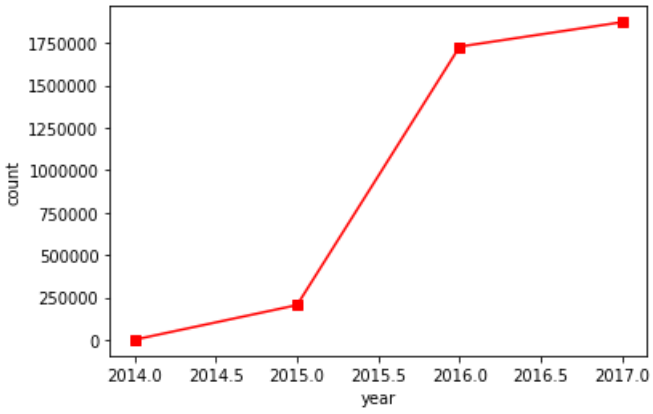
```
data1['year'].value_counts()
```

Out[68]:

```
2017    1871744
2016    1727360
2015     205293
2014       1102
Name: year, dtype: int64
```

In [33]:

```
x = [2014,2015,2016,2017]#Abscissa
k1 = [1102,205293,1727360,1871744]#Y-axis
plt.plot(x,k1,'s-',color = 'r',label=" the number of songs per year")#s-:square
plt.xlabel("year")# X name
plt.ylabel("count")#Y name
plt.show()
```



the number of songs per year

Spotify's user community saw a constant growth since 2014. In particular, the fastest growth was in 2015, where the number of Spotify users grew nearly 600%. The slowdown after 2015 was probably attributable to the introduction of one of Spotify's major competitors: Apple music.

Top5 Loving music regions

In [34]:

```
#define top.5 loving music regions
region_lovemusice=data1['region_code'].value_counts()
region_lovemusice[:5]
```

Out[34]:

```
GB-LND    495661
GB-BIR    105316
GB-MAN     78696
GB-SRY     76532
GB-GLG     72776
Name: region_code, dtype: int64
```

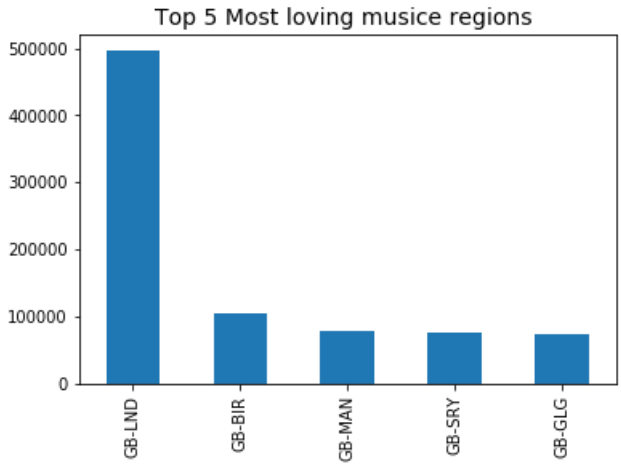
In [35]:

```
#plot top.5 loving music regions
```

```
region_lovemusice_bar = region_lovemusice[:5].plot(kind="bar")
region_lovemusice_bar.set_title("Top 5 Most loving musice regions ", fontsize=14)
```

Out[35]:

```
Text(0.5, 1.0, 'Top 5 Most loving musice regions ')
```



Loving music regions

London plays the most music among UK cities. How much music is played is highly correlated with the population of the city. London is the largest city in the UK, with about 4 times the population of the second largest city: Manchester. Therefore, the large volume of music played in London is, to a large degree, explained by the population residing in this city.

Top 10 Most Streamed Artists

In [36]:

```
#define top.10 popular artists
artists = data["artist_name"].value_counts()
artists[:10]
```

Out[36]:

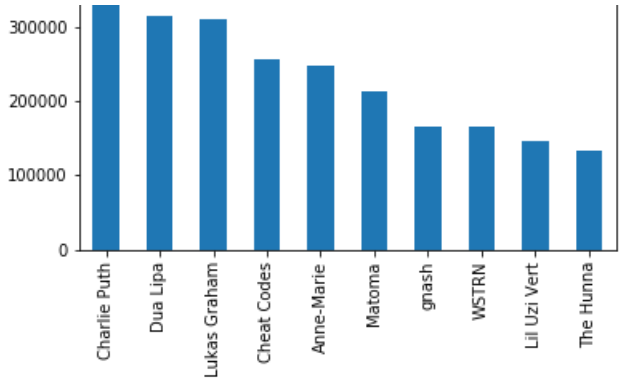
```
Charlie Puth    447873
Dua Lipa        315663
Lukas Graham    311271
Cheat Codes     255820
Anne-Marie      247934
Matoma          212210
gnash           165683
WSTRN           164885
Lil Uzi Vert    146692
The Hunna       132287
Name: artist_name, dtype: int64
```

In [37]:

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

Out[37]:

```
Text(0.5, 1.0, 'Top 10 Most Popular Artists')
```



Popular artists

Charlie Puth tops the 'Most popular artist' list, followed by Dua Lipa and Lukas Graham.

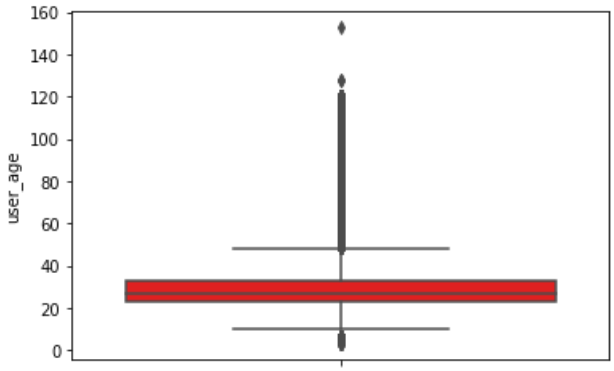
Box-plot

In [38]:

```
seaborn.boxplot(data_new['user_age'],orient='v',color='red')
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fc82db780>
```

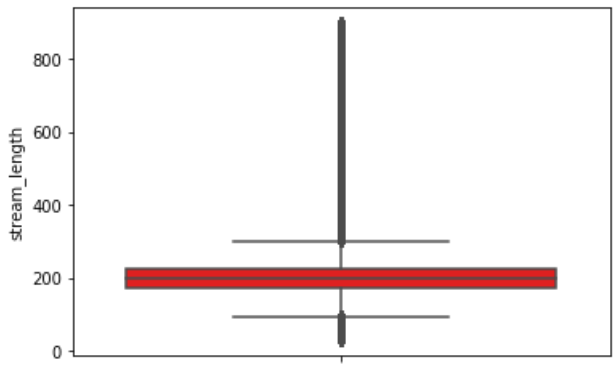


In [39]:

```
seaborn.boxplot(data_new['stream_length'],orient='v',color='red')
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fc60848d0>
```



Box-plot for user age and stream length

Most of Spotify's users fall into the age band of 22-35. The medium of user age is 28, lower than 30. This means the majority of Spotify's users are relatively young, with more born after 1990 than not. Spotify is yet to reach the elder population. The data does not seem entirely accurate, with some outliers at the age of 140, or 0. This brings the quality of the dataset to our attention: some users do not correctly state their ages. The median of stream time is 200 minutes. 50% users stream 180 to 220 minutes on one go. Most streams last from 100 to 300 minutes. Some long streams can last up to 950 minutes.

WEEK 3 Assignment

Submission Deadline: 13.02.2020

3. Data Preperation and Feature Engineering

From our business understanding, we know that our criteria for success is whether or not an artist has been on one of 4 key playlists. The column 'stream_source_uri', contains data about the source of the stream – whether it was from an artist's page, an album, a playlist etc.

For streams coming from different playlists, only the Spotify URI code is provided. To make sense of this column and identify our key playlists, we can use the additional table provided that we cleaned above and named 'playlist_mapper'.

We can being by out data preperation by subsetting the 4 key playlists we are interested in and creating our dependent variable:

In [ ]:

```
```

Create Dependent Variable

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 [87]:

```
# 4 key Playlists

# select relevant playlists

# Define Dependent Variable

# def get_successful_artists(data):

# def get_successful_before_2017(data):
```

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:

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.

In [90]:

```
# Stream count per artist
```

In [91]:

```
# Number of users per artist
```

In [92]:

```
# Passion Score
```

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

In [96]:

```
# you could divide up the work in the group by getting different people to calculate different fea
tures

#def playlist_avg_stream_counts(data):


#def playlist_avg_number_of_users(data):


#def playlist_avg_passion_score(data):


# make sure you think they are actually being calculated correctly
# how could you demonstrate the code you write is working correctly?
```

User-base features

We can use the age and gender columns to create an audience profile per artist.

- Gender Percentage Breakdown
- 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 [98]:

```
# Gender breakdown
```

In [ ]:

```
# Age breakdown

#def age_percentages(df):
```

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

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.

In [ ]:

```
```

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.

In [9]:

```
# Region Code PCA

from sklearn import decomposition

#pca = decomposition.PCA()
#pca.fit(X)

#pca_regions_output
```

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.

Check the PCA feature table to make sure the dataframe looks as expected. Comment on anything the looks important.

In [11]:

```
#pca_regions_output['pca_ouput_df'].head()
```

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.

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

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.

Now we can combine all of our features that we generated above, into a dataframe that can be processed by a machine learning algorithm:

In [103]:

```
# variables
```

In [104]:

```
#final_df = pd.DataFrame(variables)
```

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

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.

> **ACTION: Missing values**
>
> Use the **Imputer** class to alter your final Dataframe that contains your feature vector.

In [108]:

```
# Handle missing values
#from sklearn.preprocessing import Imputer

#imp = Imputer(missing_values='NaN', strategy='median', axis=0)

#fill remaining nan with 0
```

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.

In [14]:

```
# Check for multicollinearity
```

In [15]:

```
# Remove one of highly correlated varibles (test removing other as well)
```

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.

In [17]:

```
# Class balance
```

WEEK 5 Assignment

Submission Deadline: 20.02.2020

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

> **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 [115]:

```
# Split into train and validation set
```

In [116]:

```
# from sklearn.tree import DecisionTreeClassifier
```

```
# YOU CAN EXPERIMENT WITH CLASSIFIERS NOT EXPLICITLY COVERED IN CLASS

# classifiers
```

In [117]:

```
# def get_feature_names
```

In [18]:

```
#for clf in classifiers:
    #clf.fit(X, y)
    # score = clf.score(X_test, y_test)
```

5. 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 [ ]:

```
# Confusion Matrix
```

In [ ]:

```
# Plot Confusion Matrix
```

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.

In [ ]:

```
# # ROC curve
```

```
# # ROC curve

# Plot classifier ROC
```

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.

In [ ]:

```
# Feature importance analysis
```

Summary

Please provide summaries of the work completed and the outcomes of the analysis

Tips completing the coursework

- **Faculty** - You are free to run the code on your local machine, but if training timings and memory become an issue then use Faculty to complete the coursework. Technical support for using Faculty will be provided as necessary.
- **JIRA** - Assess the different potential work packages and break the overall objectives 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.
- **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 contribute to improved 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 relatively good model performance (which ever metric you use to measure this) and give you confidence about the ML project structure and workflow. 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** - Do not limit your analysis to the instructions provided in Guidelines cells, feel free to expand your analysis beyond them.