FinalCode_Part1_Edit3 (2)

April 1, 2021

GROUP COURSEWORK

MSIN0097 Predictive Analytics

Group Name: Group 9

Emails:

daniela.avramioti.20@ucl.ac.uk kunal.kalani.20@ucl.ac.uk christopher.kindl.20@ucl.ac.uk jonas.sunde.20@ucl.ac.uk valentina.isbasoiu.20@ucl.ac.uk

[2]:

```
# #### RUN THE FOLLOWING PIP INSTALL FOR NEW SERVER
# !pip install -U kaleido
# !pip install imblearn
# !pip install xgboost
# ##### ONCE RUN, RESTART KERNEL AND RUN THE FOLLOWING NOTEBOOK
```

```
 .:::.     .::.
…yy:    .yy.
:.  .yy.    y.
     :y:   .:
     .yy  .:
      yy..:
      :y:.
      .y.
     .:.
…:.
:::.
```

• Project files and data should be stored in /project. This is shared among everyone
  in the project.

• Personal files and configuration should be stored in /home/faculty.

• Files outside /project and /home/faculty will be lost when this server is terminated.

• Create custom environments to setup your servers reproducibly.

```
Collecting kaleido
  Using cached kaleido-0.2.1-py2.py3-none-manylinux1_x86_64.whl (79.9 MB)
Installing collected packages: kaleido
Successfully installed kaleido-0.2.1
```

```
 .:::.    .::.
…yy:    .yy.
:.  .yy.    y.
     :y:   .:
     .yy  .:
      yy..:
      :y:.
      .y.
     .:.
…:.
:::.
```

• Project files and data should be stored in /project. This is shared among everyone
  in the project.

• Personal files and configuration should be stored in /home/faculty.

• Files outside /project and /home/faculty will be lost when this server is terminated.

• Create custom environments to setup your servers reproducibly.

```
Collecting imblearn
  Using cached imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Using cached imbalanced_learn-0.8.0-py3-none-any.whl (206 kB)
Requirement already satisfied: joblib>=0.11 in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from imbalanced-
learn->imblearn) (0.16.0)
Requirement already satisfied: numpy>=1.13.3 in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from imbalanced-
learn->imblearn) (1.18.5)
Collecting scikit-learn>=0.24
  Using cached scikit_learn-0.24.1-cp38-cp38-manylinux2010_x86_64.whl (24.9 MB)
Requirement already satisfied: scipy>=0.19.1 in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from imbalanced-
learn->imblearn) (1.5.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
```

```
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from scikit-
learn>=0.24->imbalanced-learn->imblearn) (2.1.0)
Installing collected packages: scikit-learn, imbalanced-learn, imblearn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 0.23.1
    Uninstalling scikit-learn-0.23.1:
      Successfully uninstalled scikit-learn-0.23.1
Successfully installed imbalanced-learn-0.8.0 imblearn-0.0 scikit-learn-0.24.1
```

```
         .:::.     .::.
        ...yy:    .yy.
        :.  .yy.    y.
             :y:   .:
             .yy  .:
              yy..:
              :y:.
              .y.
             .:.
        ...:.
        :::.
```

• Project files and data should be stored in /project. This is shared among everyone
  in the project.
• Personal files and configuration should be stored in /home/faculty.
• Files outside /project and /home/faculty will be lost when this server is terminated.
• Create custom environments to setup your servers reproducibly.

```
Collecting xgboost
  Using cached xgboost-1.3.3-py3-none-manylinux2010_x86_64.whl (157.5 MB)
Requirement already satisfied: numpy in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from xgboost) (1.18.5)
Requirement already satisfied: scipy in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from xgboost) (1.5.0)
Installing collected packages: xgboost
Successfully installed xgboost-1.3.3
```

[3]:
```python
# To display full output in Notebook, instead of only the last result

from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast_node_interactivity = "all"
```

1 COURSEWORK: WARNER MUSIC

1.0.1 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

1.1 0. Business Case Understanding

1.1.1 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.

1.1.2 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.

1.1.3 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.

1.1.4 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:

1.1.5 *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.

1.2 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.

[4]:
```
# Preamble
import pandas as pd
import random
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
from matplotlib.colors import ListedColormap
from __future__ import division
```

```python
from matplotlib import colors as mcolors
import seaborn as sns

# Add more stuff here as necessary
from scipy.stats import spearmanr




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

1.3 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)

Read in the data

[5]:
```python
# Read in sampled data
data = pd.read_csv('cleaned_data.csv',low_memory=False)
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
```

Check Streaming data

[6]:
```python
# Displaying all the columns
pd.set_option('display.max_columns', 45)
```

[7]:
```python
# Check head
all_artists.head()
```

[7]:
```
   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

            log_time  mobile                          track_id          isrc  \
0  20160510T12:15:00    True  8f1924eab3804f308427c31d925c1b3f  USAT21600547
1  20160510T12:15:00    True  8f1924eab3804f308427c31d925c1b3f  USAT21600547
2  20160510T14:00:00    True  8f1924eab3804f308427c31d925c1b3f  USAT21600547
3  20160510T10:45:00    True  8f1924eab3804f308427c31d925c1b3f  USAT21600547
4  20160510T10:15:00    True  8f1924eab3804f308427c31d925c1b3f  USAT21600547

            upc       artist_name    track_name                 album_name  \
0  7.567991e+10  Sturgill Simpson  Call To Arms  A Sailor's Guide to Earth
1  7.567991e+10  Sturgill Simpson  Call To Arms  A Sailor's Guide to Earth
2  7.567991e+10  Sturgill Simpson  Call To Arms  A Sailor's Guide to Earth
3  7.567991e+10  Sturgill Simpson  Call To Arms  A Sailor's Guide to Earth
4  7.567991e+10  Sturgill Simpson  Call To Arms  A Sailor's Guide to Earth

                        customer_id postal_code   access country_code gender  \
0  6c022a8376c10aae37abb839eb7625fe          NE     free           GB   male
1  6c022a8376c10aae37abb839eb7625fe          NE     free           GB   male
2  352292382ff3ee0cfd3b73b94ea0ff8f           1  premium           GB   male
3  c3f2b54e76696ed491d9d8f964c97774          MK  premium           GB   male
4  6a06a9bbe042c73e8f1a3596ec321636          KT  premium           GB   male

   birth_year                    filename region_code  referral_code  \
0      1968.0  streams_20160510_GB.004.gz      GB-DUR            NaN
1      1968.0  streams_20160510_GB.004.gz      GB-DUR            NaN
2      1995.0  streams_20160510_GB.002.gz      GB-ESS            NaN
3      1992.0  streams_20160510_GB.007.gz      GB-HRT            NaN
4      1979.0  streams_20160510_GB.004.gz      GB-LND            NaN

  partner_name financial_product user_product_type  offline_timestamp  \
0          NaN               NaN                ad                NaN
1          NaN               NaN                ad                NaN
2          NaN           student              paid                NaN
3          NaN           student              paid                NaN
4          NaN               NaN              paid                NaN
```

```
   stream_length  stream_cached stream_source stream_source_uri stream_device  \
0          277.0            NaN         album               NaN        mobile
1           53.0            NaN         album               NaN        mobile
2          326.0            NaN    collection               NaN        mobile
3          330.0            NaN    collection               NaN        tablet
4           90.0            NaN    collection               NaN        mobile

  stream_os                             track_uri     track_artists  source  \
0   Android  spotify:track:4m1opmaYT9zk5OP7IHUb9R  Sturgill Simpson     NaN
1   Android  spotify:track:4m1opmaYT9zk5OP7IHUb9R  Sturgill Simpson     NaN
2   Android  spotify:track:4m1opmaYT9zk5OP7IHUb9R  Sturgill Simpson     NaN
3       iOS  spotify:track:4m1opmaYT9zk5OP7IHUb9R  Sturgill Simpson     NaN
4       iOS  spotify:track:4m1opmaYT9zk5OP7IHUb9R  Sturgill Simpson     NaN

             DateTime  hour  minute  week  month  year        date  weekday  \
0  2016-05-10 12:15:00    12      15    19      5  2016  2016-05-10        1
1  2016-05-10 12:15:00    12      15    19      5  2016  2016-05-10        1
2  2016-05-10 14:00:00    14       0    19      5  2016  2016-05-10        1
3  2016-05-10 10:45:00    10      45    19      5  2016  2016-05-10        1
4  2016-05-10 10:15:00    10      15    19      5  2016  2016-05-10        1

  weekday_name playlist_id playlist_name
0      Tuesday         NaN           NaN
1      Tuesday         NaN           NaN
2      Tuesday         NaN           NaN
3      Tuesday         NaN           NaN
4      Tuesday         NaN           NaN
```

[8]:
```
#Check info
all_artists.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3805499 entries, 0 to 3805498
Data columns (total 45 columns):
 #   Column             Dtype
---  ------             -----
 0   Unnamed: 0         int64
 1   Unnamed: 0.1       int64
 2   Unnamed: 0.1.1     object
 3   day                int64
 4   log_time           object
 5   mobile             bool
 6   track_id           object
 7   isrc               object
 8   upc                float64
 9   artist_name        object
```

```
 10  track_name         object
 11  album_name         object
 12  customer_id        object
 13  postal_code        object
 14  access             object
 15  country_code       object
 16  gender             object
 17  birth_year         float64
 18  filename           object
 19  region_code        object
 20  referral_code      float64
 21  partner_name       object
 22  financial_product  object
 23  user_product_type  object
 24  offline_timestamp  float64
 25  stream_length      float64
 26  stream_cached      float64
 27  stream_source      object
 28  stream_source_uri  object
 29  stream_device      object
 30  stream_os          object
 31  track_uri          object
 32  track_artists      object
 33  source             float64
 34  DateTime           object
 35  hour               int64
 36  minute             int64
 37  week               int64
 38  month              int64
 39  year               int64
 40  date               object
 41  weekday            int64
 42  weekday_name       object
 43  playlist_id        object
 44  playlist_name      object
dtypes: bool(1), float64(7), int64(9), object(28)
memory usage: 1.3+ GB
```

[9]:
```
#Check stats
all_artists.describe()
```

[9]:
```
         Unnamed: 0  Unnamed: 0.1        day           upc    birth_year  \
count  3.805499e+06  3.805499e+06  3805499.0  3.805499e+06  3.795478e+06
mean   1.902749e+06  1.902750e+07       10.0  2.389062e+11  1.990107e+03
std    1.098553e+06  1.098553e+07        0.0  2.757391e+11  1.068282e+01
min    0.000000e+00  9.000000e+00       10.0  1.686134e+10  1.867000e+03
25%    9.513745e+05  9.513754e+06       10.0  7.567991e+10  1.987000e+03
```

```
50%     1.902749e+06  1.902750e+07       10.0  1.902958e+11  1.993000e+03
75%     2.854124e+06  2.854124e+07       10.0  1.902960e+11  1.997000e+03
max     3.805498e+06  3.805499e+07       10.0  5.414940e+12  2.017000e+03

       referral_code  offline_timestamp  stream_length  stream_cached  source  \
count            0.0                0.0   3.805499e+06            0.0     0.0
mean             NaN                NaN   1.891587e+02            NaN     NaN
std              NaN                NaN   6.105546e+01            NaN     NaN
min              NaN                NaN   3.000000e+01            NaN     NaN
25%              NaN                NaN   1.720000e+02            NaN     NaN
50%              NaN                NaN   2.000000e+02            NaN     NaN
75%              NaN                NaN   2.240000e+02            NaN     NaN
max              NaN                NaN   9.000000e+02            NaN     NaN

               hour        minute          week         month          year  \
count  3.805499e+06  3.805499e+06  3.805499e+06  3.805499e+06  3.805499e+06
mean   1.373665e+01  2.254671e+01  2.316008e+01  5.970407e+00  2.016437e+03
std    5.400456e+00  1.675157e+01  1.320996e+01  3.036840e+00  5.964080e-01
min    0.000000e+00  0.000000e+00  1.000000e+00  1.000000e+00  2.014000e+03
25%    1.000000e+01  1.500000e+01  1.400000e+01  4.000000e+00  2.016000e+03
50%    1.400000e+01  3.000000e+01  2.300000e+01  6.000000e+00  2.016000e+03
75%    1.800000e+01  4.500000e+01  3.200000e+01  8.000000e+00  2.017000e+03
max    2.300000e+01  4.500000e+01  5.000000e+01  1.200000e+01  2.017000e+03

            weekday
count  3.805499e+06
mean   2.837800e+00
std    2.001057e+00
min    0.000000e+00
25%    1.000000e+00
50%    3.000000e+00
75%    5.000000e+00
max    6.000000e+00
```

[10]:
```python
#Checking Null Values
null_columns=all_artists.columns[all_artists.isnull().any()]
all_artists[null_columns].isnull().sum()
```

[10]:
```
isrc                      4
postal_code         1352181
gender                40422
birth_year            10021
region_code          261956
referral_code       3805499
partner_name        3378646
financial_product   2329099
user_product_type     22992
```

```
offline_timestamp    3805499
stream_cached        3805499
stream_source_uri    2761628
source               3805499
playlist_id          2761628
playlist_name        2826389
dtype: int64
```

Check Playlist data

[11]:
```python
#Checking playlists original data
playlist_ids_and_titles.head()
```

[11]:

	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

[12]:
```python
playlist_ids_and_titles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 194560 entries, 0 to 194559
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   id      194559 non-null  object
 1   name    194486 non-null  object
dtypes: object(2)
memory usage: 3.0+ MB
```

[13]:
```python
#Use this playlist data as it is cleaned
playlist_mapper.head()
```

[13]:

	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

[14]:
```python
playlist_mapper.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 149589 entries, 0 to 194559
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
```

```
---  ------  --------------   -----
 0   id      149589 non-null  object
 1   name    149584 non-null  object
dtypes: object(2)
memory usage: 3.4+ MB
```

1.4 Exploratory Analysis

[15]:
```python
#Copying all_artists to df for visuals
df = all_artists.copy()
```

[16]:
```python
#applying the lowercase for artists throughout the entire dataframe, to aovid␣
↪any duplicates
df['artist_name']=df['artist_name'].astype(str).str.lower()
```

1.4.1 Defining Success

[17]:
```python
#9235 unique playlist ids
df['playlist_id'].nunique()
```

[17]:
```
9235
```

Hot Hits UK

[18]:
```python
#Checking playlist ids for 'Hot Hits UK'
df[df['playlist_name']=='Hot Hits UK']['playlist_id'].unique()
```

[18]:
```
array(['6FfOZSAN3N6u7v81uS7mxZ', '37i9dQZF1DWY4lFlS4Pnso'], dtype=object)
```

[19]:
```python
#Checking playlist ids for 'Hot Hits UK' in mapper
playlist_mapper[playlist_mapper['name']=='Hot Hits UK']['id'].unique() #Need to␣
↪select the appropriate playlist
```

[19]:
```
array(['6FfOZSAN3N6u7v81uS7mxZ', '37i9dQZF1DWY4lFlS4Pnso'], dtype=object)
```

[20]:
```python
#Identifying target playlist for 'Hot Hits' based on highest streams
hothits = df.groupby(['playlist_id','playlist_name'])['log_time'].agg(['count'])
hothits = hothits.sort_values(by='count', ascending = False)
hothits.reset_index(inplace=True)
hh =hothits[hothits['playlist_name']=='Hot Hits UK'].head(1)
hh
```

[20]:

	playlist_id	playlist_name	count
0	6FfOZSAN3N6u7v81uS7mxZ	Hot Hits UK	146552

Massive Dance Hits

```
[21]: df[df['playlist_name']=='Massive Dance Hits']['playlist_id'].unique() # No␣
↪Problem....
```

```
[21]: array(['37i9dQZF1DX5uokaTN4FTR'], dtype=object)
```

```
[22]: playlist_mapper[playlist_mapper['name']=='Massive Dance Hits']['id'].unique()
```

```
[22]: array(['37i9dQZF1DX5uokaTN4FTR'], dtype=object)
```

```
[23]: #Identifying target playlist for 'Massive Dance Hits' based on highest streams
hothits = df.groupby(['playlist_id','playlist_name'])['log_time'].agg(['count'])
hothits = hothits.sort_values(by='count', ascending = False)
hothits.reset_index(inplace=True)
mdh = hothits[hothits['playlist_name']=='Massive Dance Hits']
mdh
```

[23]:

	playlist_id	playlist_name	count
24	37i9dQZF1DX5uokaTN4FTR	Massive Dance Hits	7087

The Indie List

```
[24]: df[df['playlist_name']=='The Indie List']['playlist_id'].unique()
```

```
[24]: array(['37i9dQZF1DWVTKDs2aOkxu'], dtype=object)
```

```
[25]: playlist_mapper[playlist_mapper['name']=='The Indie List']['id'].unique()
```

```
[25]: array(['37i9dQZF1DWVTKDs2aOkxu'], dtype=object)
```

```
[26]: #Identifying target playlist for 'Indie List' based on highest streams
hothits = df.groupby(['playlist_id','playlist_name'])['log_time'].agg(['count'])
hothits = hothits.sort_values(by='count', ascending = False)
hothits.reset_index(inplace=True)
indie_list=hothits[hothits['playlist_name']=='The Indie List']
indie_list
```

[26]:

	playlist_id	playlist_name	count
76	37i9dQZF1DWVTKDs2aOkxu	The Indie List	1572

New Music Friday

```
[27]: df[df['playlist_name']=='New Music Friday']['playlist_id'].unique()
```

```
[27]: array(['37i9dQZF1DX4JAvHpjipBk', '1EnTBEgCWiTX2YHyAzkcFn',
       '0dLTdpGyf0OPSyYXInvRd5', '3DL9G1ApvJDIR4IhWIJ8AQ',
       '6wx0wiD9V6JJ2EOh4KM3Ox'], dtype=object)
```

```
[28]: playlist_mapper[playlist_mapper['name']=='New Music Friday']['id'].unique()
```

[28]:
```
array(['3DL9G1ApvJDIR4IhWIJ8AQ', '2mnRUIMJWqooAWlMjrlghi',
       '1EnTBEgCWiTX2YHyAzkcFn', '37i9dQZF1DWXJfnUiYjUKT',
       '0dLTdpGyfOOPSyYXInvRd5', '6wxOwiD9V6JJ2EOh4KM3Ox',
       '35PofY2z4SqqbynOKXmdYV', '4vGgUbD6tW2xMTABaVzCXo',
       '37i9dQZF1DX4JAvHpjipBk', '37i9dQZF1DWT2SPAYawYcO'], dtype=object)
```

[29]:
```python
#Identifying target playlist for 'New Music Friday' based on highest streams

hothits = df.groupby(['playlist_id','playlist_name'])['log_time'].agg(['count'])
hothits = hothits.sort_values(by='count', ascending = False)
hothits.reset_index(inplace=True)
nmf=hothits[hothits['playlist_name']=='New Music Friday'].head(1)
nmf
```

[29]:

	playlist_id	playlist_name	count
174	37i9dQZF1DX4JAvHpjipBk	New Music Friday	452

Top 4 playlists identified

[30]:
```python
#Joing top 4 playlist identities
top4 = pd.concat([hh, mdh,indie_list,nmf], ignore_index=True)
top4
```

[30]:

	playlist_id	playlist_name	count
0	6FfOZSAN3N6u7v81uS7mxZ	Hot Hits UK	146552
1	37i9dQZF1DX5uokaTN4FTR	Massive Dance Hits	7087
2	37i9dQZF1DWVTKDs2aOkxu	The Indie List	1572
3	37i9dQZF1DX4JAvHpjipBk	New Music Friday	452

Success

[31]:
```python
#Defining SUCCESS Based on entry into top 4 playlists
playlist_conditions = [(df['playlist_id'] == '6FfOZSAN3N6u7v81uS7mxZ') &␣
 ↪(df['playlist_name'] =='Hot Hits UK' ),
    (df['playlist_id'] == '37i9dQZF1DX5uokaTN4FTR') & (df['playlist_name'] ==␣
 ↪'Massive Dance Hits'),
    (df['playlist_id'] == '37i9dQZF1DWVTKDs2aOkxu') & (df['playlist_name'] ==␣
 ↪'The Indie List'),
    (df['playlist_id'] == '37i9dQZF1DX4JAvHpjipBk') & (df['playlist_name'] ==␣
 ↪'New Music Friday')]

playlist_values = ['Hot Hits UK','Massive Dance Hits','New Music Friday', 'The␣
 ↪Indie List']

df['top4'] = np.select(playlist_conditions, playlist_values)
df['success']= np.where(df['top4']!='0', 1, 0)
df['success'].unique()
```

[31]: array([0, 1])

[32]:
```
success = df.groupby(['artist_name','success'])['log_time'].agg(['count'])
success.reset_index(inplace=True)
```

[33]:
```
#Some artists became successful (entered top 4) after a few streams, hence,␣
↪need to consider them as 1 and not 0
success['success'].value_counts()
```

[33]:
```
0    639
1     70
Name: success, dtype: int64
```

[34]:
```
#Dropping artist 0s for artists which have 0 and 1 both.
success = success.drop_duplicates(['artist_name'],keep = 'last')
```

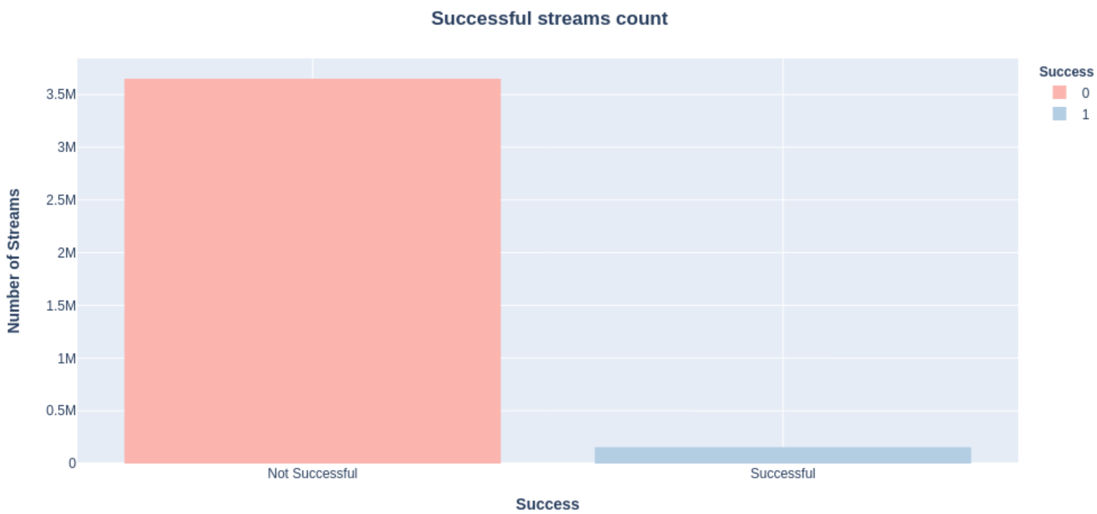
[35]:
```
#Final Split of successful and not successful artists
success.success.value_counts()
```

[35]:
```
0    569
1     70
Name: success, dtype: int64
```

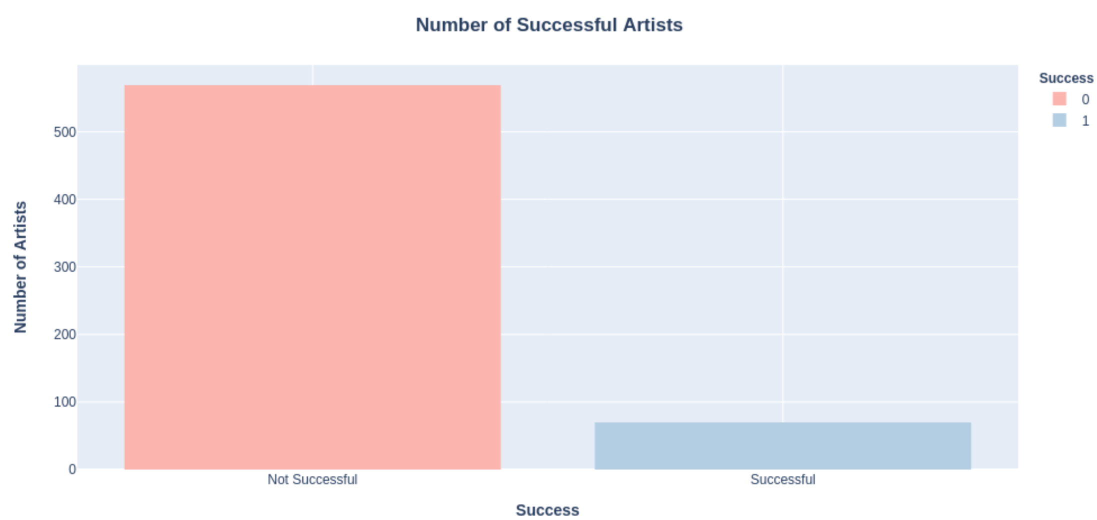
[36]:
```
# OUTPUT IN PNG FORMAT
import plotly.io as pio
png_renderer = pio.renderers["png"]
png_renderer.width = 1000
png_renderer.height = 500
pio.renderers.default = "png"
```

[37]:
```
# To display full output in Notebook, instead of only the last result

from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast_node_interactivity = "last_expr"
```

[38]:
```
# Checking streams from top 4 and without top 4 playlists
fig = px.histogram(df, x="success", color='success',
                   labels={'success': "<b>Success</b>"},
                   color_discrete_sequence=px.colors.qualitative.Pastel1
                  )
fig.update_xaxes(type='category',ticktext=["Not Successful", "Successful"],␣
↪tickvals=["0", "1"], showgrid=True)
fig.update_layout(title={'text': '<b>Successful streams count</b>','x':0.5},
                  yaxis_title_text='<b>Number of Streams</b>')
fig.show()
```



[39]:
```
#Checking Successful vs not successful artists
fig = px.histogram(success, x="success", color='success',
                   color_discrete_sequence=px.colors.qualitative.Pastel1,
                   labels={'success': "<b>Success</b>"} )

fig.update_xaxes(type='category',ticktext=["Not Successful", "Successful"],
                 tickvals=["-0", "1"], showgrid=True)
fig.update_layout(title={'text': '<b>Number of Successful Artists</b>','x':0.5},
                  yaxis_title_text='<b>Number of Artists</b>')
fig.show()
```



Creating a function that will get the successfull artists at present, as well as the successful artists who have played on the top 4 playlists prior to 2017.

[40]:
```
#rename the key_playlists
key_playlists = top4

# filter the main dataframe with the relevant playlists
df = df.assign(success= (df.playlist_id.isin(key_playlists.playlist_id)) & (df.
↪playlist_name.isin(key_playlists.playlist_name)).astype(int))
df['success'] = df['success'].astype(int) #had to add this, not sure why .
↪astype(int) didn't work in the previous line of code



# Define Dependent Variable
artists_labels = df['success'].copy()

#Defining a new function to get the succesful artists
#key_artists= only the artists that have been played on the "success" playlists.
def get_successful_artists(data):
    try:
        #key_artists = all_artists where 'success'=1
        key_artists = df.loc[(df['playlist_id'].isin(key_playlists.playlist_id)␣
↪& (df['playlist_name'].isin(key_playlists.playlist_name)))].copy() #should␣
↪we do an outter merge?
        key_artists['artist_name'] = key_artists['artist_name'].astype(str).str.
↪lower()
        key_artists.artist_name = key_artists.artist_name.str.replace(' ', '_')
        key_artists = key_artists.drop_duplicates(subset = ['artist_name'])

        return(key_artists)
    except:
        "Cannot merge data"

#key_artists= only the artists that have been played on the "success" playlists␣
↪prior to 2017
def get_successful_before_2017(data):
    try:
        key_old_artists = df.loc[(df['playlist_id'].isin(key_playlists.
↪playlist_id) & (df['playlist_name'].isin(key_playlists.playlist_name)))].
↪copy()
        key_old_artists['artist_name'] = key_old_artists['artist_name'].
↪astype(str).str.lower()
        key_old_artists.artist_name = key_old_artists.artist_name.str.replace('␣
↪', '_')
        key_old_artists = key_old_artists[key_old_artists.year<2017] #want the␣
↪function to first filter by year
```

```
        key_old_artists = key_old_artists.drop_duplicates(subset =␣
↪['artist_name']) #then filter by duplicates
        return (key_old_artists)
    except:
        "Cannot merge data"
```

[41]:
```
#applying the functions to get the lists of key artists
key_artists = get_successful_artists(df)
key_old_artists = get_successful_before_2017(df)

#key artists in the successful playlists
key_artists.shape #should be 71
#compare the outcome of the spotfunc functions to this
key_old_artists.shape
```

[41]:
```
(28, 47)
```

[42]:
```
#Filter only rows with successful artists
all_artists_filter=df.loc[(df['success'] == 1) & (df['playlist_name'].
↪notnull())]
all_artists_filter
```

[42]:
```
         Unnamed: 0  Unnamed: 0.1                     Unnamed: 0.1.1  day  \
633             633          6339    ('small_artists_2016.csv', 6339)   10
17270         17270        172709  ('small_artists_2016.csv', 172709)   10
26996         26996        269969  ('small_artists_2016.csv', 269969)   10
29244         29244        292449  ('small_artists_2016.csv', 292449)   10
60803         60803        608039  ('small_artists_2016.csv', 608039)   10
…                 …             …                                 …   …
3779860     3779860      37798609                             1045211   10
3780407     3780407      37804079                             1050681   10
3785409     3785409      37854099                             1100701   10
3786427     3786427      37864279                             1110881   10
3792533     3792533      37925339                             1171941   10

                   log_time  mobile                          track_id  \
633       20160410T12:45:00   False  db62b1d507bc4fd1bc8b4785d82d6356
17270     20160210T18:30:00   False  bcdbf945cb194356b39ec0d36476e641
26996     20160710T10:00:00    True  de3c49e047a945aba049b7467f9a20ad
29244     20160510T17:00:00   False  3ccdfba451974b848e509b3a97b553ba
60803     20160510T11:15:00   False  5e6ae0c4967047dbb832caec9b1df082
…                       …       …                                 …
3779860   20170110T21:00:00    True  1ac77530b0c64409b125257b61d557ba
3780407   20170110T20:15:00    True  1ac77530b0c64409b125257b61d557ba
3785409   20170210T14:30:00    True  1ac77530b0c64409b125257b61d557ba
3786427   20170210T10:30:00   False  1ac77530b0c64409b125257b61d557ba
3792533   20170310T23:45:00    True  1ac77530b0c64409b125257b61d557ba
```

```
                isrc           upc      artist_name  \
633      USAT21601204  7.567991e+10     vinyl on hbo
17270    AUUQU1600001  8.256463e+11      xavier dunn
26996    USAT21601112  7.567991e+10  sir the baptist
29244    FR9W11520485  1.902960e+11             amir
60803    FR43Y1600020  1.902960e+11       starlovers
…                 …             …              …
3779860  GBAHS1600223  1.902960e+11       anne-marie
3780407  GBAHS1600223  1.902960e+11       anne-marie
3785409  GBAHS1600223  1.902960e+11       anne-marie
3786427  GBAHS1600223  1.902960e+11       anne-marie
3792533  GBAHS1600223  1.902960e+11       anne-marie

                                          track_name  \
633                                Where Are You Now?
17270                       Fancy - Xavier Dunn Cover
26996                  Raise Hell (feat. ChuchPeople)
29244                                    J'ai cherché
60803    Feeling Good (feat. B. Lauren) - Radio Edit
…                                                  …
3779860                                         Alarm
3780407                                         Alarm
3785409                                         Alarm
3786427                                         Alarm
3792533                                         Alarm

                                                album_name  \
633      VINYL: Music From The HBO® Original Series - V…
17270                                              BIMYOU
26996                      Raise Hell (feat. ChuchPeople)
29244                                        J'ai cherché
60803                      Feeling Good (feat. B. Lauren)
…                                                       …
3779860                                             Alarm
3780407                                             Alarm
3785409                                             Alarm
3786427                                             Alarm
3792533                                             Alarm

                              customer_id postal_code   access country_code  \
633      b6dc09bcc7ed512dc268f17cfb35a116         NaN     free           GB
17270    285bc2e475578285c8dcc4073ef0f5a8          12     free           GB
26996    7af71efffd6e31350dd33975fafe9263          12  premium           GB
29244    d4a7d0836ddb867b887470983528O2a3          12     free           GB
60803    6f4bb297abefad0130fe2f6ce4ac2e64          No  premium           GB
…                                     …           …        …            …
```

```
3779860  735c88b5699fadc8bd40a1588c97b0fc          No     free           GB
3780407  86e0042ac5979a9d0927172640527a5c          No     free           GB
3785409  2d8968f0cdb42a58ee3be65e54b824bd           1  premium           GB
3786427  e344ca7326761cd57350173846c976ba          12  premium           GB
3792533  078f4e4b17b5c3008e7357ed5351d5d4          12  premium           GB

         gender  birth_year                    filename region_code  \
633        male      1992.0  streams_20160410_GB.006.gz         NaN
17270    female      1986.0  streams_20160210_GB.001.gz      GB-GLG
26996    female      1997.0  streams_20160710_GB.004.gz      GB-KEN
29244      male      1999.0  streams_20160510_GB.008.gz      GB-ENF
60803    female      1996.0  streams_20160510_GB.004.gz      GB-POW
…           …         …                         …          …
3779860  female      2000.0  streams_20170110_GB.006.gz      GB-LND
3780407  female      1996.0  streams_20170110_GB.007.gz      GB-BOL
3785409  female      1996.0  streams_20170210_GB.002.gz      GB-LIN
3786427    male      1993.0  streams_20170210_GB.013.gz      GB-WSX
3792533  female      1960.0  streams_20170310_GB.000.gz      GB-KEN

         referral_code partner_name  … stream_length stream_cached  \
633                NaN          NaN  …          42.0           NaN
17270              NaN          NaN  …          57.0           NaN
26996              NaN  vodafone-uk  …         225.0           NaN
29244              NaN          NaN  …          51.0           NaN
60803              NaN          NaN  …          45.0           NaN
…                    …          …   …           …            …
3779860            NaN          NaN  …         206.0           NaN
3780407            NaN          NaN  …         206.0           NaN
3785409            NaN          NaN  …         206.0           NaN
3786427            NaN          NaN  …         206.0           NaN
3792533            NaN          NaN  …         206.0           NaN

           stream_source                                  stream_source_uri  \
633      others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…
17270    others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…
26996    others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…
29244    others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…
60803    others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…
…                    …                                                 …
3779860  others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…
3780407  others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…
3785409  others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…
3786427  others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…
3792533  others_playlist  spotify:user:spotify_uk_:playlist:6FfOZSAN3N6u…

         stream_device stream_os                             track_uri  \
633            desktop   Browser  spotify:track:6FYqeL3oEPuUf1d0KVhDRs
```

```
17270         desktop     other  spotify:track:5Kn5jBrNiTF1V5woyKOfkt
26996          mobile       iOS  spotify:track:6LlQBOQweWj8N5TK4S2HtH
29244         desktop     other  spotify:track:1QJFNfsVQA7VfUJFKgQJzI
60803         desktop     other  spotify:track:2Sa7zqp8M7L9eiChXhtp8C
…                 …       …                                     …
3779860        mobile   Android  spotify:track:0OwX5aROoW1Iip8FV51Efg
3780407        mobile       iOS  spotify:track:0OwX5aROoW1Iip8FV51Efg
3785409        mobile       iOS  spotify:track:0OwX5aROoW1Iip8FV51Efg
3786427       desktop     other  spotify:track:0OwX5aROoW1Iip8FV51Efg
3792533        mobile       iOS  spotify:track:0OwX5aROoW1Iip8FV51Efg

                    track_artists source            DateTime hour  minute  \
633      Royal Blood, Vinyl on HBO    NaN  2016-04-10 12:45:00   12      45
17270                  Xavier Dunn    NaN  2016-02-10 18:30:00   18      30
26996             Sir the Baptist    NaN  2016-07-10 10:00:00   10       0
29244                         Amir    NaN  2016-05-10 17:00:00   17       0
60803                   Starlovers    NaN  2016-05-10 11:15:00   11      15
…                               …    …                    …   …      …
3779860                 Anne-Marie    NaN  2017-01-10 21:00:00   21       0
3780407                 Anne-Marie    NaN  2017-01-10 20:15:00   20      15
3785409                 Anne-Marie    NaN  2017-02-10 14:30:00   14      30
3786427                 Anne-Marie    NaN  2017-02-10 10:30:00   10      30
3792533                 Anne-Marie    NaN  2017-03-10 23:45:00   23      45

        week  month  year        date  weekday  weekday_name  \
633       14      4  2016  2016-04-10        6        Sunday
17270      6      2  2016  2016-02-10        2     Wednesday
26996     27      7  2016  2016-07-10        6        Sunday
29244     19      5  2016  2016-05-10        1       Tuesday
60803     19      5  2016  2016-05-10        1       Tuesday
…        …     …    …          …       …           …
3779860    2      1  2017  2017-01-10        1       Tuesday
3780407    2      1  2017  2017-01-10        1       Tuesday
3785409    6      2  2017  2017-02-10        4        Friday
3786427    6      2  2017  2017-02-10        4        Friday
3792533   10      3  2017  2017-03-10        4        Friday

                    playlist_id  playlist_name         top4 success
633      6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1
17270    6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1
26996    6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1
29244    6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1
60803    6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1
…                             …            …          …     …
3779860  6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1
3780407  6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1
3785409  6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1
```

```
3786427  6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1
3792533  6FfOZSAN3N6u7v81uS7mxZ    Hot Hits UK  Hot Hits UK       1

[155663 rows x 47 columns]
```

1.4.2 Stream Counts

Artists Stream Counts

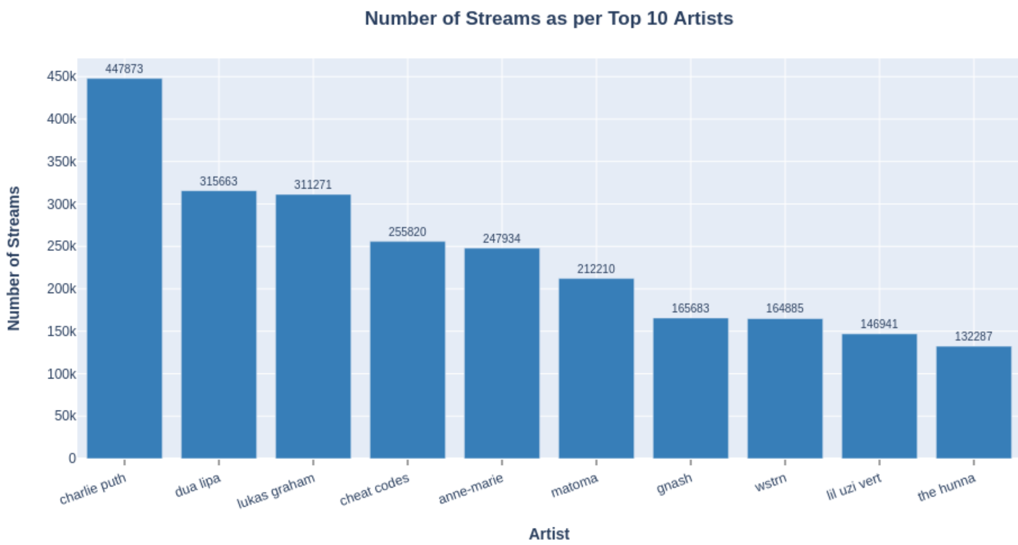
[43]:
```python
# COUNTING NUMBER OF STREAMS FOR EACH ARTIST
artists_stream_count = df.groupby('artist_name')['log_time'].agg(['count'])
artists_stream_count = artists_stream_count.sort_values(by='count', ascending = False)
artists_stream_count.reset_index(inplace=True)
```

[44]:
```python
artists_stream_count.head(10)
```

[44]:

	artist_name	count
0	charlie puth	447873
1	dua lipa	315663
2	lukas graham	311271
3	cheat codes	255820
4	anne-marie	247934
5	matoma	212210
6	gnash	165683
7	wstrn	164885
8	lil uzi vert	146941
9	the hunna	132287

[45]:
```python
#VISUAL FOR NUMBER OF STREAMS FOR EACH ARTIST (TOP 10 ONLY)
fig = px.bar(artists_stream_count[:10], x="artist_name", y='count', barmode='group',
                  labels={'artist_name': "<b>Artist",'count':'<b>Number of Streams'},
              color_discrete_sequence=px.colors.qualitative.Set1[1:4], text = 'count'
                   )
fig.update_xaxes(tickangle = 340, showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside', textfont_size=10)
fig.update_layout(title={'text': '<b>Number of Streams as per Top 10 Artists</b>','x':0.5})
fig.show()
```



Playlist Stream Counts

[46]:
```
#VISUAL FOR NUMBER OF STREAMS FOR EACH PLAYLIST
playlists_stream_count = df.
↪groupby(['playlist_id','playlist_name'])['log_time'].agg(['count'])
playlists_stream_count = playlists_stream_count.sort_values(by='count',␣
↪ascending = False)
playlists_stream_count.reset_index(inplace=True)
```

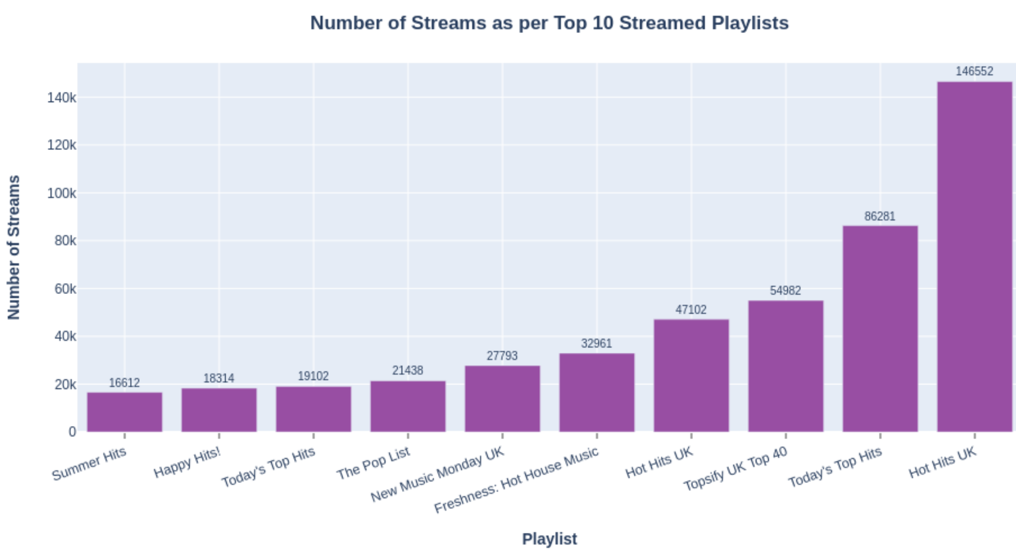
[47]:
```
playlists_stream_count[:10]
```

[47]:

	playlist_id	playlist_name	count
0	6FfOZSAN3N6u7v81uS7mxZ	Hot Hits UK	146552
1	5FJXhjdILmRA2z5bvz4nzf	Today's Top Hits	86281
2	1QM1qz09ZzsAPiXphF1l4S	Topsify UK Top 40	54982
3	37i9dQZF1DWY4lFlS4Pnso	Hot Hits UK	47102
4	7wUUwoxU2S6BRKA2bDPYKD	Freshness: Hot House Music	32961
5	6LY8RItOWg6IkpJBtxP2xu	New Music Monday UK	27793
6	1Tv8NFvQY2aRuGi2JrOeyN	The Pop List	21438
7	37i9dQZF1DXcBWIGoYBM5M	Today's Top Hits	19102
8	65V6djkcVRyOStLd8nza8E	Happy Hits!	18314
9	6QcSOqFBwxfJPwVV4Ybjp6	Summer Hits	16612

[48]:
```
#VISUAL FOR NUMBER OF STREAMS FOR EACH PLAYLIST (TOP 10 OUT OF ALL PLAYLISTS)

fig = px.bar(playlists_stream_count[0:10].
↪sort_values(by='count',ascending=True),
             x="playlist_id", y='count', barmode='group',
```

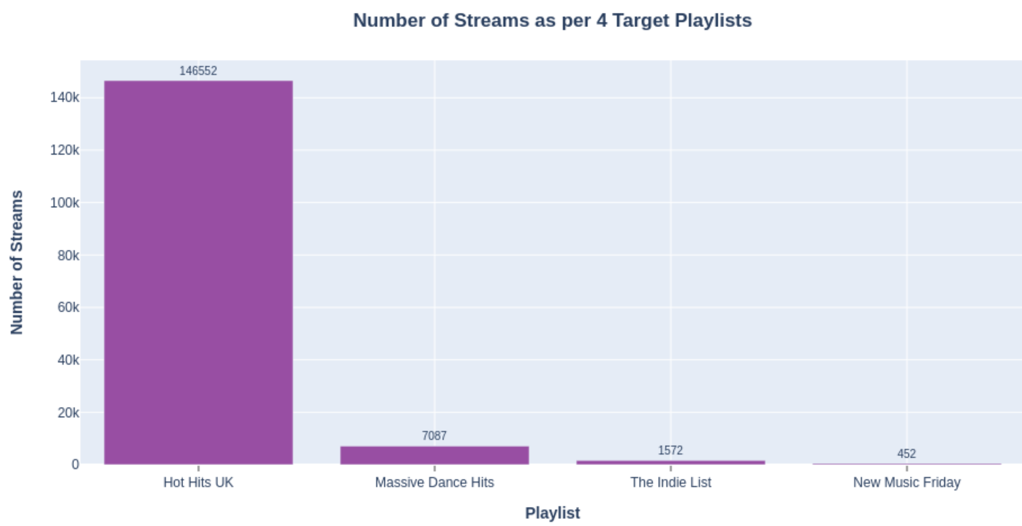
```
            labels={'playlist_id': "<b>Playlist",'count':'<b>Number of␣
↪Streams'},
            color_discrete_sequence=px.colors.qualitative.Set1[3:4], text =␣
↪'count'
                 )
fig.update_xaxes(type='category',
                 ticktext=playlists_stream_count[0:10].
↪sort_values(by='count',ascending=True)[-10:]['playlist_name'],
                 tickvals=playlists_stream_count[0:10].
↪sort_values(by='count',ascending=True)[-10:]['playlist_id'],
                 tickangle = 340,showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside',␣
↪textfont_size=10)
fig.update_layout(title={'text': '<b>Number of Streams as per Top 10 Streamed␣
↪Playlists<b>','x':0.5})
fig.show()
```



Top 4 Playlist Stream Counts

[49]:
```
#VISUAL FOR NUMBER OF STREAMS FOR EACH PLAYLIST (TOP  4 TARGET PLAYLISTS)

fig = px.bar(top4,
            x="playlist_name", y='count', barmode='group',
            labels={'playlist_name': "<b>Playlist",'count':'<b>Number of␣
↪Streams'},
            color_discrete_sequence=px.colors.qualitative.Set1[3:4], text =␣
↪'count'
                 )
```

```
fig.update_xaxes(tickangle = 0, showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside',␣
 ↪textfont_size=10)
fig.update_layout(title={'text': '<b>Number of Streams as per 4 Target␣
 ↪Playlists<b>','x':0.5})
fig.show()
```



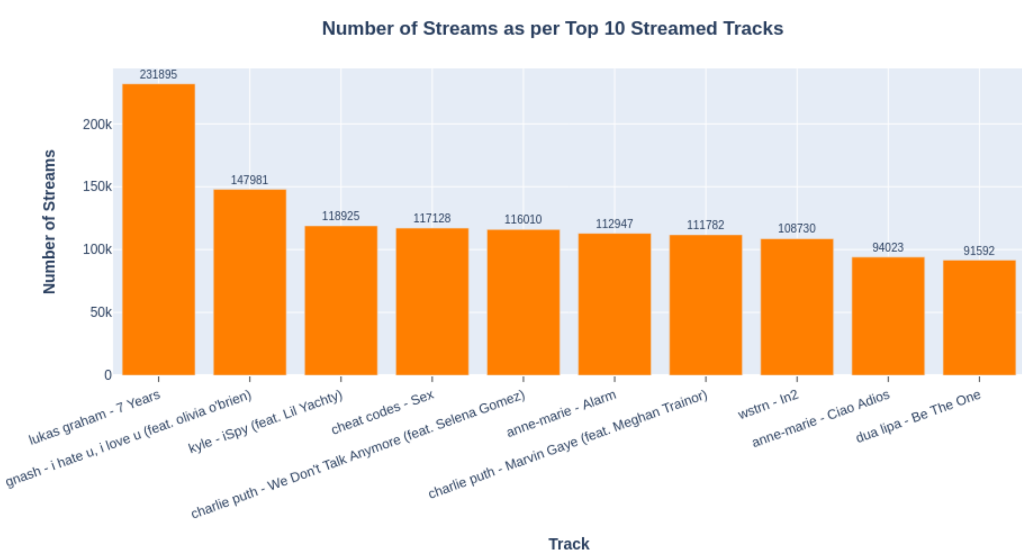
Song Stream Counts

[50]:
```
#Calculating number of streams for each track

songs_stream_count = df.groupby(['track_name','artist_name'])['log_time'].
 ↪agg(['count'])
songs_stream_count = songs_stream_count.sort_values(by='count', ascending =␣
 ↪False)
songs_stream_count.reset_index(inplace=True)
# Joining track name and artist name
songs_stream_count["artist_track"] = songs_stream_count["artist_name"] +' - '+␣
 ↪songs_stream_count["track_name"]
songs_stream_count.head(10)
```

[50]:
```
                                        track_name   artist_name   count  \
0                                          7 Years  lukas graham  231895
1    i hate u, i love u (feat. olivia o'brien)           gnash  147981
2                       iSpy (feat. Lil Yachty)            kyle  118925
3                                              Sex   cheat codes  117128
4   We Don't Talk Anymore (feat. Selena Gomez)  charlie puth  116010
5                                            Alarm    anne-marie  112947
```

```
6          Marvin Gaye (feat. Meghan Trainor)  charlie puth  111782
7                                         In2         wstrn  108730
8                                  Ciao Adios    anne-marie   94023
9                                  Be The One      dua lipa   91592

                                      artist_track
0                            lukas graham - 7 Years
1  gnash - i hate u, i love u (feat. olivia o'brien)
2                       kyle - iSpy (feat. Lil Yachty)
3                                 cheat codes - Sex
4  charlie puth - We Don't Talk Anymore (feat. Se…
5                               anne-marie - Alarm
6  charlie puth - Marvin Gaye (feat. Meghan Trainor)
7                                      wstrn - In2
8                          anne-marie - Ciao Adios
9                            dua lipa - Be The One
```

[51]:
```
#Visual for number of streams for top 10 songs
fig = px.bar(songs_stream_count[:10],
             x="artist_track", y='count', barmode='group',
             labels={'artist_track': "<b>Track",'count':'<b>Number of Streams'},
             color_discrete_sequence=px.colors.qualitative.Set1[4:5], text =␣
 ↪'count'
                  )
fig.update_xaxes(tickangle = 340,showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside',␣
 ↪textfont_size=10)
fig.update_layout(title={'text': '<b>Number of Streams as per Top 10 Streamed␣
 ↪Tracks<b>','x':0.5})
fig.show()
```



1.4.3 Number of Songs

per artist

[52]:
```
#Artists stream count (note the list of top 10 is different to Kunal's)
artists_stream_count = df.groupby('artist_name')['log_time'].agg(['count'])
artists_stream_count = artists_stream_count.sort_values(by='count', ascending =␣
↪False)
artists_stream_count.reset_index(inplace=True)
artists_stream_count
```

[52]:
```
             artist_name   count
0           charlie puth  447873
1               dua lipa  315663
2           lukas graham  311271
3            cheat codes  255820
4             anne-marie  247934
..                   …      …
634      rebecka karlsson       1
635  los tres paraguayos       1
636                deuspi       1
637            vince pope       1
638            los romeos       1

[639 rows x 2 columns]
```

[53]:
```
artists_stream_count
```

[53]:
```
             artist_name   count
0           charlie puth  447873
1               dua lipa  315663
2           lukas graham  311271
3            cheat codes  255820
4             anne-marie  247934
..                   …      …
634      rebecka karlsson       1
635  los tres paraguayos       1
636                deuspi       1
637            vince pope       1
638            los romeos       1

[639 rows x 2 columns]
```

[54]:
```
# Calculating number of songs from each artist
song_count = df.groupby(['artist_name'])['track_name'].agg(['nunique'])
```

```
song_count.reset_index(inplace=True)
song_count = song_count.rename(columns= {'nunique':'number_songs'})
```

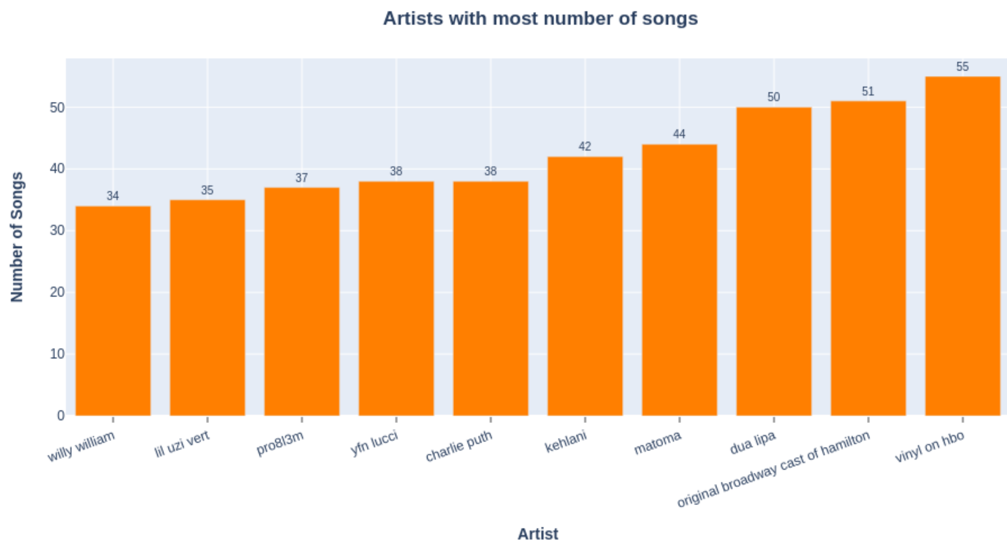
[55]:
```
song_count
```

[55]:
```
          artist_name  number_songs
0         #90s update             1
1          17 memphis             1
2                  2d             1
3                 3js             4
4          99 percent             2
..                ...           ...
634          zak abel            17
635         zakopower             1
636           zarcort            16
637  zbigniew kurtycz             1
638     zion & lennox            19

[639 rows x 2 columns]
```

[56]:
```
# adding number of songs to artists_new dataset
#THIS WILL BE THE RUNNING DATASET FOR ARTISTS
artists_new = artists_stream_count.
 ↪merge(song_count[['artist_name','number_songs']], on='artist_name')
```

[57]:
```
#Adding success to artists data
artists_new = artists_new.merge(success[['artist_name','success']],␣
 ↪on='artist_name')
```

[58]:
```
# visual for artists with most number of songs
fig = px.bar(artists_new.sort_values(by='number_songs')[-10:],
             x="artist_name", y='number_songs', barmode='group',
             labels={'artist_name': "<b>Artist",'number_songs':'<b>Number of␣
 ↪Songs'},
             color_discrete_sequence=px.colors.qualitative.Set1[4:5], text =␣
 ↪'number_songs'
                  )
fig.update_xaxes(tickangle = 340,showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside',␣
 ↪textfont_size=10)
fig.update_layout(title={'text': '<b> Artists with most number of songs</
 ↪b>','x':0.5})
fig.show()
```



per playlist

[59]:
```
#Calculating number of songs in each playlist
song_count_p = df.groupby(['playlist_id'])['track_name'].agg(['nunique'])
song_count_p.reset_index(inplace=True)
song_count_p = song_count_p.rename(columns= {'nunique':'number_songs'})
```

[60]:
```
song_count_p
```

[60]:
```
                 playlist_id  number_songs
0     0015UsoeSdMREOCWu0Dt1R            10
1     0078dWhzCWQpJKViaP4Y6j            18
2     007MG3bjc3vzffd4smEkiu             1
3     00H4E3crh1SeOP42u0VbSr             4
4     00K2xasnm9pDQk53SzNCht             2
…                        …           …
9230  7zjkU5CYNvxrfMlv2IUczL             2
9231  7zmNDikLeBiqpxBV5mZRG2             1
9232  7znqFJ2PCpQubSmg8jp45A            14
9233  7zp2jy8ir8ESw11yiwIaN7             1
9234  7zyCl1UAvFb1ZVCTz0LGFI             1

[9235 rows x 2 columns]
```

[61]:
```
# adding number of songs to playlists_new dataset
#THIS WILL BE THE RUNNING DATASET FOR PLAYLISTS
playlists_new = playlists_stream_count.
↪merge(song_count_p[['playlist_id','number_songs']], on='playlist_id')
```

[62]:
```python
playlists_new
```

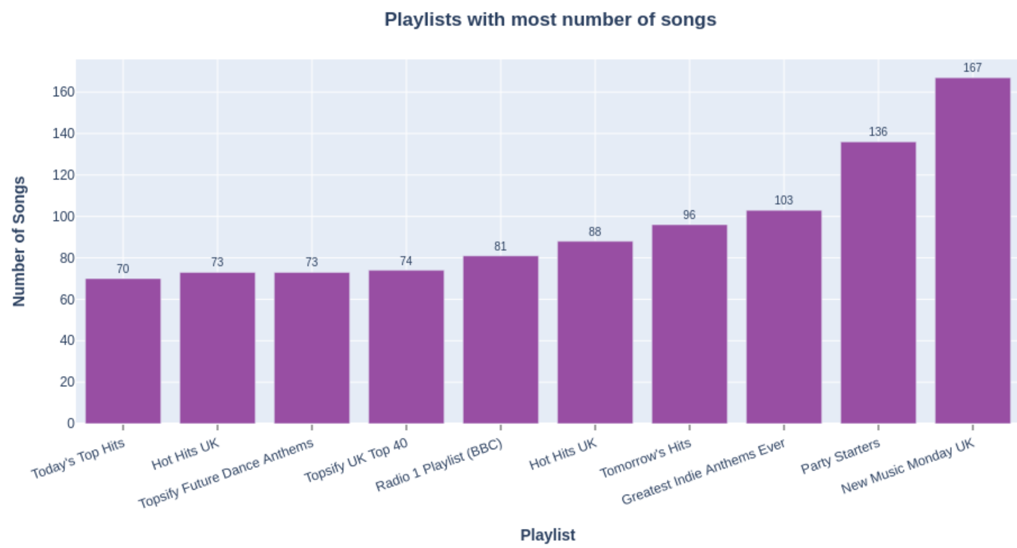
[62]:
```
                 playlist_id                playlist_name   count  number_songs
0     6FfOZSAN3N6u7v81uS7mxZ                  Hot Hits UK  146552            88
1     5FJXhjdILmRA2z5bvz4nzf               Today's Top Hits   86281            70
2     1QM1qz09ZzsAPiXphF1l4S            Topsify UK Top 40   54982            74
3     37i9dQZF1DWY4lFlS4Pnso                  Hot Hits UK   47102            73
4     7wUUwoxU2S6BRKA2bDPYKD  Freshness: Hot House Music   32961            53
…                        …                            …     …             …
7518  3foltCsFMcH6Sp4XtSQcgc     Aprés-ski La Folie Douce       1             1
7519  3ft2HOPNriZG0q2GXsYwNw                  SUMMER 2017       1             1
7520  3gAY2MQl7v75gnn3Nqxhvg            Really Cool Stuff       1             1
7521  3gDuLKpBKdinsB4wCOyBQu              Llegando a Casa       1             1
7522  7zyCl1UAvFb1ZVCTzOLGFI     sad & acoustic favorites       1             1

[7523 rows x 4 columns]
```

[63]:
```python
playlists_new.sort_values(by='number_songs')[-10:]['playlist_name']
```

[63]:
```
1                   Today's Top Hits
3                        Hot Hits UK
142     Topsify Future Dance Anthems
2                  Topsify UK Top 40
12           Radio 1 Playlist (BBC)
0                        Hot Hits UK
207                  Tomorrow's Hits
234       Greatest Indie Anthems Ever
27                    Party Starters
5                 New Music Monday UK
Name: playlist_name, dtype: object
```

[64]:
```python
#Visual for playlists with most number of songs
fig = px.bar(playlists_new.sort_values(by='number_songs')[-10:],
             x="playlist_id", y='number_songs', barmode='group',
             labels={'playlist_id': "<b>Playlist",'number_songs':'<b>Number of Songs'},
             color_discrete_sequence=px.colors.qualitative.Set1[3:5], text = 'number_songs'
                  )
fig.update_xaxes(type='category',ticktext=playlists_new.sort_values(by='number_songs')[-10:]['playlist_name'],
                 tickvals=playlists_new.sort_values(by='number_songs')[-10:]['playlist_id']
                 ,tickangle = 340,showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside', textfont_size=10)
```

```
fig.update_layout(title={'text': '<b> Playlists with most number of songs</
↪b>','x':0.5})
fig.show()
```



1.4.4 Number of Playlists per Artist

[65]:
```
#Calculating number of playlists an artist is in
playlist_num_per_artist = pd.DataFrame(df.
↪groupby(['artist_name'])['playlist_id'].nunique())
playlist_num_per_artist.reset_index(inplace=True)
playlist_num_per_artist=playlist_num_per_artist.rename(columns= {'playlist_id':
↪'playlists','artist_name':'artist_name'})
```

[66]:
```
#Adding to dataset
artists_new = artists_new.
↪merge(playlist_num_per_artist[['artist_name','playlists']], on='artist_name')
```

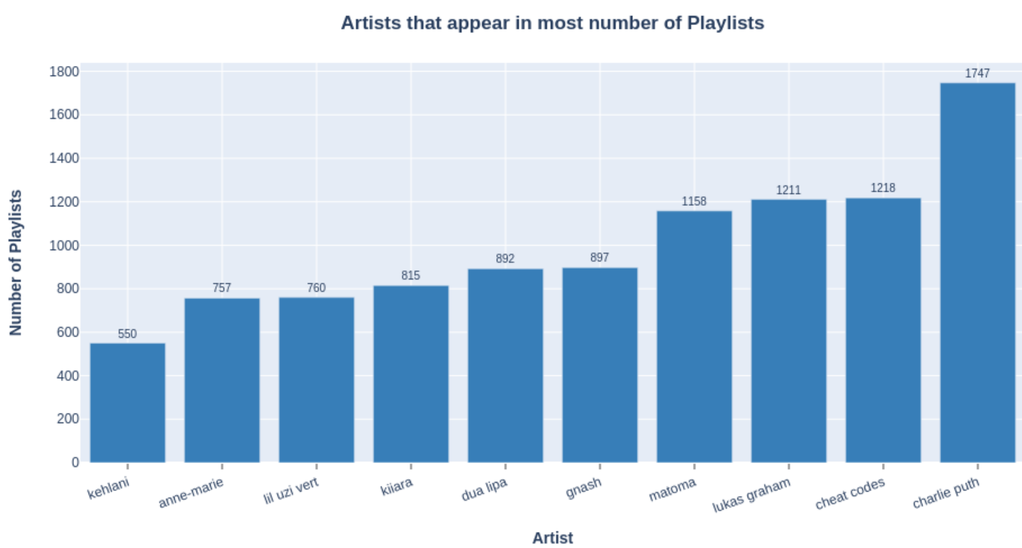
[67]:
```
artists_new
```

[67]:

	artist_name	count	number_songs	success	playlists
0	charlie puth	447873	38	1	1747
1	dua lipa	315663	50	1	892
2	lukas graham	311271	22	1	1211
3	cheat codes	255820	16	1	1218
4	anne-marie	247934	28	1	757
..	…	…	…	…	…
634	rebecka karlsson	1	1	0	0
635	los tres paraguayos	1	1	0	0

```
636          deuspi     1          1       0        1
637      vince pope     1          1       0        1
638     los romeos      1          1       0        0

[639 rows x 5 columns]
```

[68]:
```
#Visuals for playlists per artist (top 10)
fig = px.bar(artists_new.sort_values(by='playlists')[-10:],
             x="artist_name", y='playlists', barmode='group',
             labels={'artist_name': "<b>Artist",'playlists':'<b>Number of␣
 ↪Playlists'},
             color_discrete_sequence=px.colors.qualitative.Set1[1:4], text =␣
 ↪'playlists'
                  )
fig.update_xaxes(tickangle = 340, showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside',␣
 ↪textfont_size=10)
fig.update_layout(title={'text': '<b>Artists that appear in most number of␣
 ↪Playlists</b>','x':0.5})
fig.show()
```



1.4.5 Number of Artists per Playlist

[69]:
```
#Number of artists in each playlist
artist_num_per_playlist = pd.DataFrame(df.
 ↪groupby(['playlist_id','playlist_name'])['artist_name'].nunique())
artist_num_per_playlist.reset_index(inplace=True)
```

```
artist_num_per_playlist=artist_num_per_playlist.rename(columns= {'playlist_id':
↪'playlist_id','artist_name':'artists'})
```

[70]:
```
#Adding to dataset
playlists_new = playlists_new.
↪merge(artist_num_per_playlist[['playlist_id','artists']], on='playlist_id')
```

[71]:
```
playlists_new
```

[71]:
```
                 playlist_id              playlist_name   count  \
0     6FfOZSAN3N6u7v81uS7mxZ                 Hot Hits UK  146552
1     5FJXhjdILmRA2z5bvz4nzf             Today's Top Hits   86281
2     1QM1qz09ZzsAPiXphF1l4S           Topsify UK Top 40   54982
3     37i9dQZF1DWY4lFlS4Pnso                 Hot Hits UK   47102
4     7wUUwoxU2S6BRKA2bDPYKD  Freshness: Hot House Music   32961
…                        …                          …     …
7518  3foltCsFMcH6Sp4XtSQcgc    Aprés-ski La Folie Douce       1
7519  3ft2HOPNriZG0q2GXsYwNw                 SUMMER 2017       1
7520  3gAY2MQl7v75gnn3Nqxhvg           Really Cool Stuff       1
7521  3gDuLKpBKdinsB4wCOyBQu             Llegando a Casa       1
7522  7zyCl1UAvFb1ZVCTzOLGFI    sad & acoustic favorites       1

      number_songs  artists
0               88       41
1               70       34
2               74       40
3               73       41
4               53       33
…              …       …
7518             1        1
7519             1        1
7520             1        1
7521             1        1
7522             1        1

[7523 rows x 5 columns]
```

1.4.6 Playlist performance before and after getting accpeted into top-tier playlist per artist

[72]:
```
#create filter for top-tier playlists
playlist_filter = ['6FfOZSAN3N6u7v81uS7mxZ', '37i9dQZF1DX4JAvHpjipBk',␣
↪'37i9dQZF1DX5uokaTN4FTR', '37i9dQZF1DWVTKDs2aOkxu']

#filter only top-tier playlists and use min-function to identify on which date␣
↪the artist was played the first time in the playlist
```

```
a = df[df['playlist_id'].isin(playlist_filter)].groupby(['artist_name']).
 ↪agg({'date': 'min'})
df["date_first_played_top_playlist"] = ''

#assign date of first time artist's song was played in top-tier playlist
first_time = {}
for i in range(len(df[df['playlist_id'].isin(playlist_filter)].
 ↪groupby(['artist_name']).agg({'date': 'min'}).index)):
    first_time[a.index[i]] = a['date'][i]

#assign value back to dataframe using apply lambda (applies only to artists who␣
 ↪are in a top-tier playlist)
df["date_first_played_top_playlist"] = df["artist_name"].apply(lambda x:␣
 ↪first_time.get(x))
```

[73]:
```
#convert column into datetime format
df["date_first_played_top_playlist"]  = pd.
 ↪to_datetime(df["date_first_played_top_playlist"])
df["date"]  = pd.to_datetime(df["date"])
```

[74]:
```
#calculate delta between first time played in top-tier playlist and stream date
df['days_between'] = df['date'] - df['date_first_played_top_playlist']
```

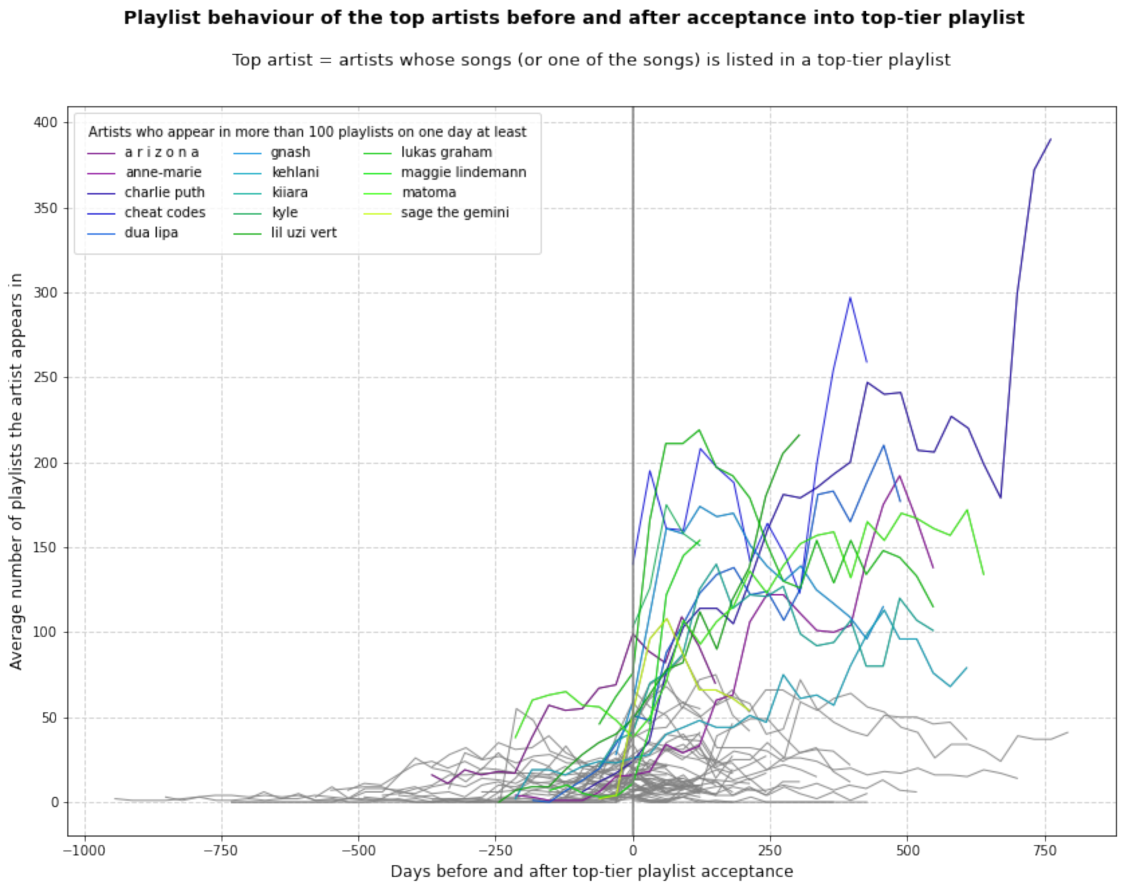
[75]:
```
#create filter to only select artists who appear in a top-tier playlist
artist_filter = [artist for artist in df[df['playlist_id'].
 ↪isin(playlist_filter)].groupby(['artist_name']).agg({'date': 'min'}).index]

#copy dataframe and only select the top-artists who have a value in column␣
 ↪first time played
df_copy = df[df['artist_name'].isin(artist_filter)]
```

[76]:
```
#convert timedelta into int format
df_copy['days_between'] = (df_copy['days_between'] / np.timedelta64(1, 'D')).
 ↪astype(int)
```

```
<ipython-input-76-93e198c265c1>:2: 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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[77]:
```
#create list for artists who appear on more than 100 playlists and for the ones␣
↪who do not
df_playlists = pd.DataFrame(df_copy.groupby(['artist_name',␣
↪'days_between'])['playlist_id'].nunique())

#make multiindex to columns
df_playlists = df_playlists.reset_index(level=['artist_name', 'days_between'])

#split df
playlist100 = [artist for artist in df_playlists[df_playlists['playlist_id'] >␣
↪100]['artist_name']]
playlist_below100 = [artist for artist in␣
↪df_playlists[df_playlists['playlist_id'] <= 100]['artist_name']]

#make lists unique
playlist100 = list(dict.fromkeys(playlist100))
playlist_below100 = list(dict.fromkeys(playlist_below100))

#create a color palette
palette = plt.get_cmap('nipy_spectral')

#plot figure
fig = plt.figure(figsize=(14,10))
ax = fig.add_subplot(111)

#plot multiple lines within the chart for artists who appear in less than 100␣
↪different playlists and color them gray

for artist in playlist_below100:
    x = df_copy[df_copy['artist_name'] == artist].groupby(['artist_name',␣
↪'days_between'])['playlist_id'].nunique().index.get_level_values(1)
    y = df_copy[df_copy['artist_name'] == artist].groupby(['artist_name',␣
↪'days_between'])['playlist_id'].nunique()
    plt.plot(x, y, marker='', color='gray', linewidth=1, alpha=0.9)

#plot multiple lines within the chart for artists who appear in more than 100␣
↪different playlists
num=0
for artist in playlist100:
    num+=12
    x = df_copy[df_copy['artist_name'] == artist].groupby(['artist_name',␣
↪'days_between'])['playlist_id'].nunique().index.get_level_values(1)
    y = df_copy[df_copy['artist_name'] == artist].groupby(['artist_name',␣
↪'days_between'])['playlist_id'].nunique()
    plt.plot(x, y, marker='', color=palette(num), linewidth=1, alpha=0.9,␣
↪label=artist)
```

```
#add legend
plt.legend(loc=2, ncol=3, title='Artists who appear in more than 100 playlists␣
↪on one day at least', borderpad = 1);

plt.suptitle('Playlist behaviour of the top artists before and after acceptance␣
↪into top-tier playlist', fontsize = 14, weight = 'bold')
plt.title('Top artist = artists whose songs (or one of the songs) is listed in␣
↪a top-tier playlist', fontsize = 13, pad=30)

ax.set_xlabel('Days before and after top-tier playlist acceptance', fontsize =␣
↪12)
ax.set_ylabel('Average number of playlists the artist appears in', fontsize =␣
↪12)

ax.xaxis.grid(color='lightgray', linestyle='--', linewidth=1);
ax.yaxis.grid(color='lightgray', linestyle='--', linewidth=1);

plt.axvline(x=0, color = 'gray');

#show the graph
plt.show()

#export to pdf
fig.savefig("playlist_before_after.pdf");
```



Growth change of average number of playlists artists appear in

[78]:
```
#calculate average number of playlists artists appear before acceptance
before = df_copy[(df_copy['days_between'] < 0) & (df_copy['artist_name'].
↪isin(artist_filter))].groupby(['artist_name'])['playlist_id']\
    .nunique().sum() / len(df_copy[(df_copy['days_between'] < 0) &␣
↪(df_copy['artist_name'].isin(artist_filter))].
↪groupby(['artist_name'])['playlist_id']\
    .nunique())
```

[79]:
```
#calculate average number of playlists artists appear before acceptance
after = df_copy[(df_copy['days_between'] >= 0) & (df_copy['artist_name'].
↪isin(artist_filter))].groupby(['artist_name'])['playlist_id']\
    .nunique().sum() / len(df_copy[(df_copy['days_between'] >= 0) &␣
↪(df_copy['artist_name'].isin(artist_filter))].
↪groupby(['artist_name'])['playlist_id']\
    .nunique())
```

```
[80]: #calculate %-change between average number of playlists per artists before and␣
       ↪after acceptance
      round(((after-before)/before)*100, 2)
```

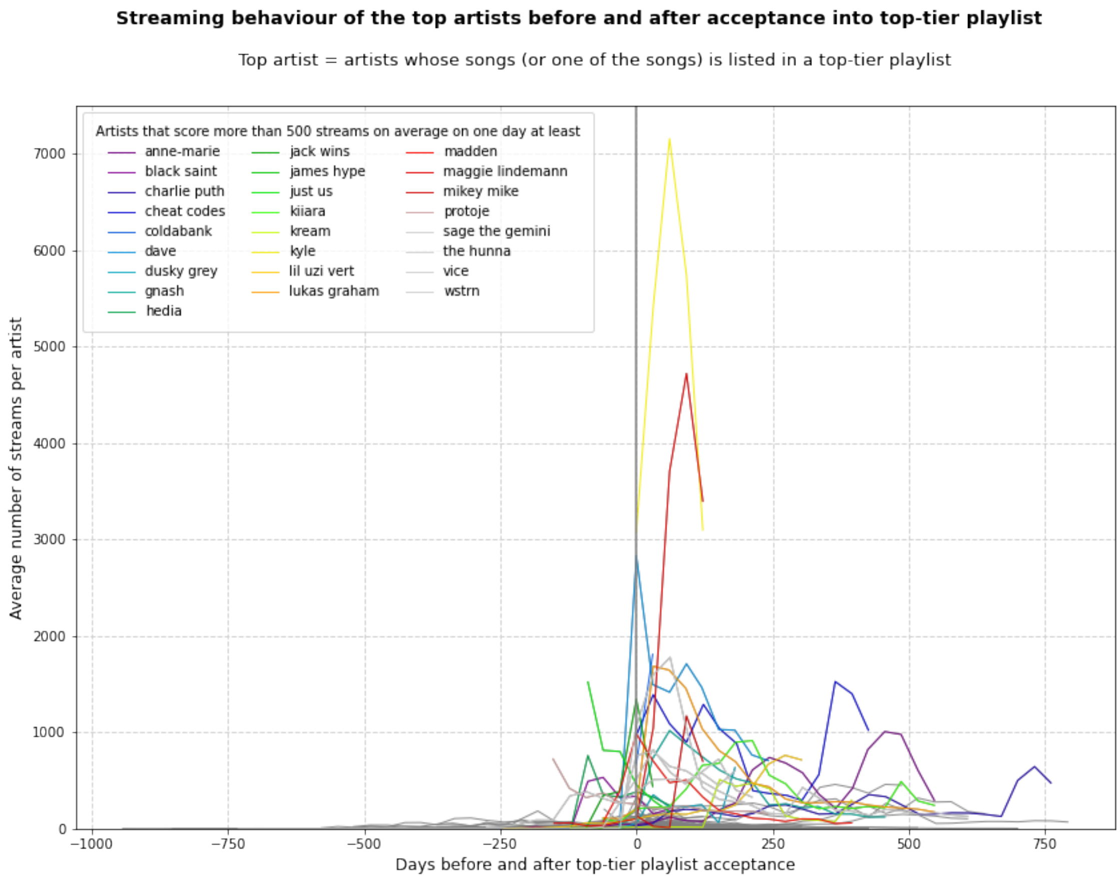
```
[80]: 399.51
```

1.4.7 Streaming performance before and after getting into top-tier playlist per artist

```
[81]: #calculate number of streams per song of an artist per DAY to standardise and,␣
       ↪thus, better compare
      df_streams = pd.DataFrame(round((df_copy.groupby(['artist_name',␣
       ↪'days_between'])['Unnamed: 0'].count()\
                                       / df_copy.groupby(['artist_name'])['track_id'].
       ↪nunique()), 2), columns = ['streams'])

      #make multiindex to columns
      df_streams =df_streams.reset_index(level=['artist_name', 'days_between'])

      #create list for artists who have more than 500 streams on average per day and␣
       ↪for the ones who have not
      stream500 = [artist for artist in df_streams[df_streams['streams'] >␣
       ↪500]['artist_name']]
      stream_below500 = [artist for artist in df_streams[df_streams['streams'] <=␣
       ↪500]['artist_name']]

      #make lists unique
      stream500 = list(dict.fromkeys(stream500))
      stream_below500 = list(dict.fromkeys(stream_below500))
```

```
[82]: #create a color palette
      palette = plt.get_cmap('nipy_spectral')

      #plot figure
      fig = plt.figure(figsize=(14,10))
      ax = fig.add_subplot(111)

      #plot multiple lines within the chart for artists who have not more than 500␣
       ↪streams per song per day
      for artist in stream_below500:
          x = df_streams[df_streams['artist_name'] == artist].days_between
          y = df_streams[df_streams['artist_name'] == artist].streams
          plt.plot(x, y, marker='', color='gray', linewidth=1, alpha=0.9)

      #plot multiple lines within the chart for artists who have more than 500␣
       ↪streams per song per day
      num=0
```

```
for artist in stream500:
    num+=12
    x = df_streams[df_streams['artist_name'] == artist].days_between
    y = df_streams[df_streams['artist_name'] == artist].streams
    plt.plot(x, y, marker='', color=palette(num), linewidth=1, alpha=0.9, label␣
 ↪= artist)

#add legend
#plt.legend(labels = top_artist, ncol=3);
plt.legend(loc=2, ncol=3, title='Artists that score more than 500 streams on␣
 ↪average on one day at least', borderpad = 1);

plt.suptitle('Streaming behaviour of the top artists before and after␣
 ↪acceptance into top-tier playlist', fontsize = 14, weight = 'bold')
plt.title('Top artist = artists whose songs (or one of the songs) is listed in␣
 ↪a top-tier playlist', fontsize = 13, pad=30)

ax.set_xlabel('Days before and after top-tier playlist acceptance', fontsize =␣
 ↪12)
ax.set_ylabel('Average number of streams per artist', fontsize = 12)

ax.xaxis.grid(color='lightgray', linestyle='--', linewidth=1);
ax.yaxis.grid(color='lightgray', linestyle='--', linewidth=1);

plt.axvline(x=0, color = 'gray');
ax.set_ylim(0, 7500)

#show the graph
plt.show()

#export to pdf
fig.savefig("./streaming_before_after.pdf")
```



1.4.8 Number of Unique Listeners

Unique listeners per artist

[83]:
```
#Number of listeners per artists
users_artist = pd.DataFrame(df.groupby('artist_name')['customer_id'].nunique())
users_artist.reset_index(inplace=True)
users_artist=users_artist.rename(columns= {'artist_name':
↪'artist_name','customer_id':'listeners'})
users_artist=users_artist.sort_values(by='listeners', ascending = False)
users_artist
```

[83]:

	artist_name	listeners
98	charlie puth	367023
158	dua lipa	260778
333	lukas graham	247580
101	cheat codes	225658
37	anne-marie	220413
..	…	…
534	ted mulry gang	1

```
198  giuseppe gibboni          1
205  helena majdaniec          1
214            hunter          1
319     local connect          1

[639 rows x 2 columns]
```

[84]:
```
users_artist
```

[84]:
```
          artist_name  listeners
98       charlie puth     367023
158          dua lipa     260778
333      lukas graham     247580
101       cheat codes     225658
37         anne-marie     220413
..                ...        ...
534    ted mulry gang          1
198  giuseppe gibboni          1
205  helena majdaniec          1
214            hunter          1
319     local connect          1

[639 rows x 2 columns]
```

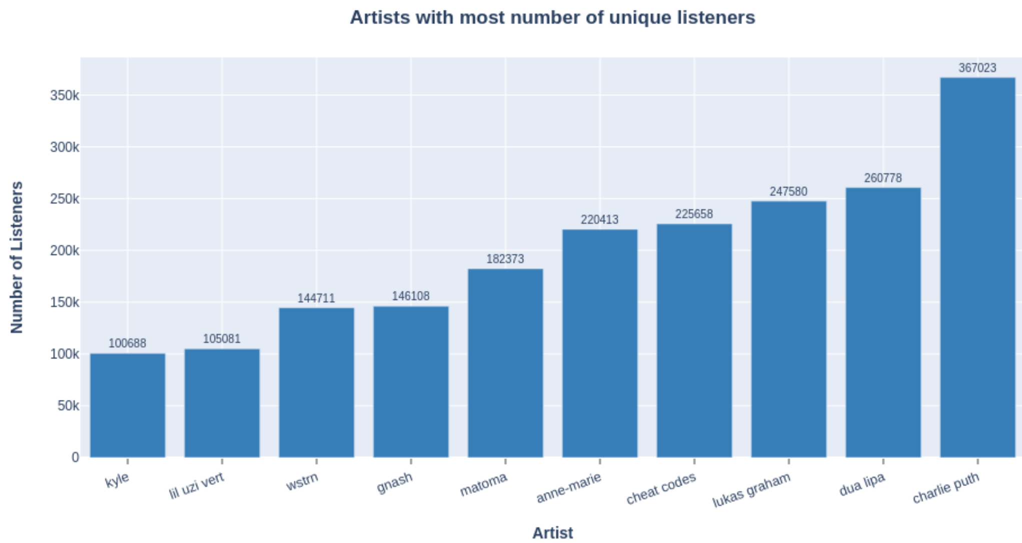
[85]:
```
#Adding to dataframe
artists_new = artists_new.merge(users_artist[['artist_name','listeners']],␣
 ↪on='artist_name')
```

[86]:
```
artists_new
```

[86]:
```
             artist_name   count  number_songs  success  playlists  listeners
0           charlie puth  447873            38        1       1747     367023
1               dua lipa  315663            50        1        892     260778
2           lukas graham  311271            22        1       1211     247580
3            cheat codes  255820            16        1       1218     225658
4             anne-marie  247934            28        1        757     220413
..                   ...     ...           ...      ...        ...        ...
634     rebecka karlsson       1             1        0          0          1
635  los tres paraguayos       1             1        0          0          1
636               deuspi       1             1        0          1          1
637           vince pope       1             1        0          1          1
638           los romeos       1             1        0          0          1

[639 rows x 6 columns]
```

[87]:
```
#Visuals for unique listeners per artist (top 10)
fig = px.bar(artists_new.sort_values(by='listeners')[-10:],
```

```
x="artist_name", y='listeners', barmode='group',
            labels={'artist_name': "<b>Artist",'listeners':'<b>Number of␣
↪Listeners'},
            color_discrete_sequence=px.colors.qualitative.Set1[1:4], text =␣
↪'listeners'
                 )
fig.update_xaxes(tickangle = 340, showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside',␣
↪textfont_size=10)
fig.update_layout(title={'text': '<b>Artists with most number of unique␣
↪listeners</b>','x':0.5})
fig.show()
```



Unique listeners per playlist

[88]:
```
#Calculating for unique listeners per playlist
users_playlist = pd.DataFrame(df.groupby('playlist_id')['customer_id'].
↪nunique())
users_playlist.reset_index(inplace=True)
users_playlist=users_playlist.rename(columns= {'customer_id':'listeners'})
users_playlist=users_playlist.sort_values(by='listeners', ascending = False)
users_playlist.head()
```

[88]:

	playlist_id	listeners
7500	6Ff0ZSAN3N6u7v81uS7mxZ	116235
6406	5FJXhjdILmRA2z5bvz4nzf	68255
3524	37i9dQZF1DWY4lFlS4Pnso	40748
1461	1QM1qz09ZzsAPiXphF1l4S	31516

```
9178  7wUUwoxU2S6BRKA2bDPYKD      27686
```

[89]:
```python
users_playlist.head()
```

[89]:
```
                 playlist_id  listeners
7500  6FfOZSAN3N6u7v81uS7mxZ     116235
6406  5FJXhjdILmRA2z5bvz4nzf      68255
3524  37i9dQZF1DWY4lFlS4Pnso      40748
1461  1QM1qz09ZzsAPiXphF1l4S      31516
9178  7wUUwoxU2S6BRKA2bDPYKD      27686
```

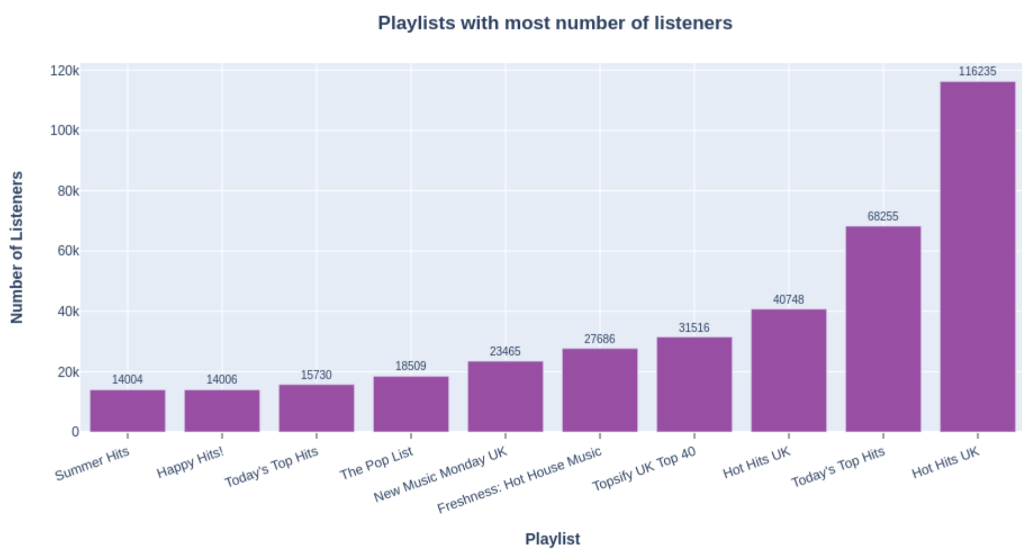
[90]:
```python
#Adding to playlists
playlists_new = playlists_new.
↪merge(users_playlist[['playlist_id','listeners']], on='playlist_id')
```

[91]:
```python
playlists_new
```

[91]:
```
                 playlist_id               playlist_name   count  \
0     6FfOZSAN3N6u7v81uS7mxZ                  Hot Hits UK  146552
1     5FJXhjdILmRA2z5bvz4nzf              Today's Top Hits   86281
2     1QM1qz09ZzsAPiXphF1l4S            Topsify UK Top 40   54982
3     37i9dQZF1DWY4lFlS4Pnso                  Hot Hits UK   47102
4     7wUUwoxU2S6BRKA2bDPYKD  Freshness: Hot House Music   32961
…                        …                          …     …
7518  3foltCsFMcH6Sp4XtSQcgc    Aprés-ski La Folie Douce       1
7519  3ft2HOPNriZG0q2GXsYwNw                 SUMMER 2017       1
7520  3gAY2MQl7v75gnn3Nqxhvg            Really Cool Stuff       1
7521  3gDuLKpBKdinsB4wCOyBQu              Llegando a Casa       1
7522  7zyCl1UAvFb1ZVCTz0LGFI    sad & acoustic favorites       1

      number_songs  artists  listeners
0               88       41     116235
1               70       34      68255
2               74       40      31516
3               73       41      40748
4               53       33      27686
…              …        …        …
7518             1        1          1
7519             1        1          1
7520             1        1          1
7521             1        1          1
7522             1        1          1

[7523 rows x 6 columns]
```

[92]:
```python
#Visuals for unique listeners per playlist (top 10)
fig = px.bar(playlists_new.sort_values(by='listeners')[-10:],
```

```
              x="playlist_id", y='listeners', barmode='group',
              labels={'playlist_id': "<b>Playlist",'listeners':'<b>Number of␣
↪Listeners'},
              color_discrete_sequence=px.colors.qualitative.Set1[3:5], text =␣
↪'listeners'
                   )
fig.update_xaxes(type='category',ticktext=playlists_new.
↪sort_values(by='listeners')[-10:]['playlist_name'],
                 tickvals=playlists_new.sort_values(by='listeners')[-10:
↪]['playlist_id']
                 ,tickangle = 340,showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside',␣
↪textfont_size=10)
fig.update_layout(title={'text': '<b> Playlists with most number of listeners</
↪b>','x':0.5})
fig.show()
```



1.4.9 Passion Scores

Passion Score as per Artists

[93]:
```
#Calculating passion score per artist
artists_new['passion_score']=artists_new['count']/artists_new['listeners']
```

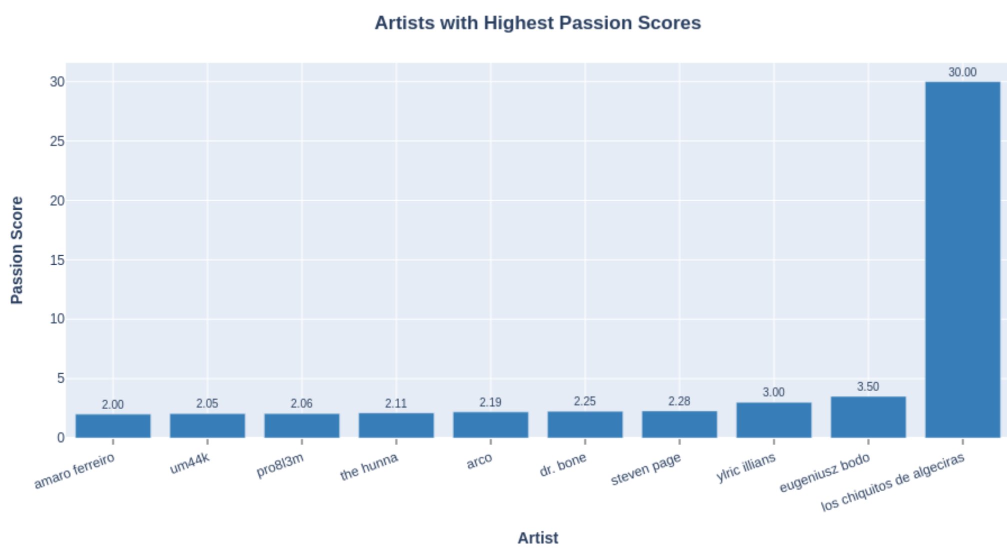
[94]:
```
artists_new
```

[94]:
```
            artist_name   count  number_songs  success  playlists  listeners  \
0          charlie puth  447873            38        1       1747     367023
1              dua lipa  315663            50        1        892     260778
```

```
2           lukas graham  311271            22          1        1211      247580
3            cheat codes  255820            16          1        1218      225658
4             anne-marie  247934            28          1         757      220413
..                   …         …         …         …         …         …
634      rebecka karlsson       1             1          0           0           1
635  los tres paraguayos       1             1          0           0           1
636                deuspi       1             1          0           1           1
637             vince pope       1             1          0           1           1
638             los romeos       1             1          0           0           1

     passion_score
0         1.220286
1         1.210466
2         1.257254
3         1.133662
4         1.124861
..             …
634       1.000000
635       1.000000
636       1.000000
637       1.000000
638       1.000000

[639 rows x 7 columns]
```

[95]:

```
#Visual for  passion score per artist (Top 10)

fig = px.bar(artists_new.sort_values(by='passion_score')[-10:],
             x="artist_name", y='passion_score', barmode='group',
             labels={'artist_name': "<b>Artist",'passion_score':'<b>Passion␣
↪Score'},
             color_discrete_sequence=px.colors.qualitative.Set1[1:4], text =␣
↪'passion_score'
                  )
fig.update_xaxes(tickangle = 340, showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:.2f}', textposition='outside',␣
↪textfont_size=10)
fig.update_layout(title={'text': '<b>Artists with Highest Passion Scores</
↪b>','x':0.5})
fig.show()
```



Passion Score as per Playlist

[96]:
```python
#Calculations for  passion score per playlist
playlists_new['passion_score']=playlists_new['count']/playlists_new['listeners']
```

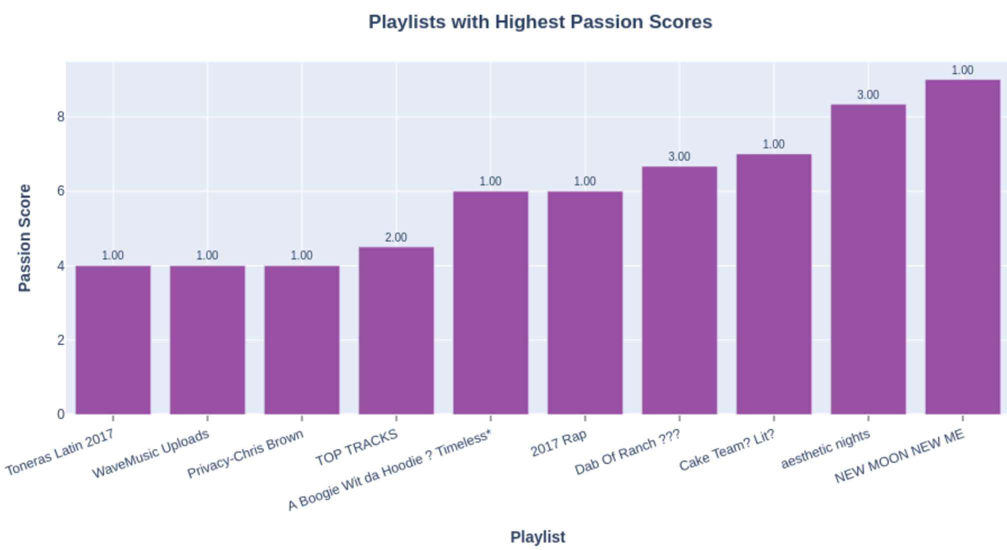
[97]:
```python
playlists_new
```

[97]:
```
                playlist_id                 playlist_name   count  \
0     6FfOZSAN3N6u7v81uS7mxZ                    Hot Hits UK  146552
1     5FJXhjdILmRA2z5bvz4nzf                 Today's Top Hits  86281
2     1QM1qz09ZzsAPiXphF1l4S              Topsify UK Top 40  54982
3     37i9dQZF1DWY4lFlS4Pnso                    Hot Hits UK   47102
4     7wUUwoxU2S6BRKA2bDPYKD  Freshness: Hot House Music   32961
…                          …                          …      …
7518  3foltCsFMcH6Sp4XtSQcgc     Aprés-ski La Folie Douce       1
7519  3ft2HOPNriZG0q2GXsYwNw                  SUMMER 2017       1
7520  3gAY2MQl7v75gnn3Nqxhvg              Really Cool Stuff       1
7521  3gDuLKpBKdinsB4wCOyBQu               Llegando a Casa       1
7522  7zyCl1UAvFb1ZVCTzOLGFI     sad & acoustic favorites       1

      number_songs  artists  listeners  passion_score
0               88       41     116235       1.260825
1               70       34      68255       1.264098
2               74       40      31516       1.744574
3               73       41      40748       1.155934
4               53       33      27686       1.190530
…              …        …         …              …
7518             1        1          1       1.000000
7519             1        1          1       1.000000
```

```
7520                1         1            1        1.000000
7521                1         1            1        1.000000
7522                1         1            1        1.000000

[7523 rows x 7 columns]
```

[98]:

```
#Visual for passion score per playlist (Top 10)

fig = px.bar(playlists_new.sort_values(by='passion_score')[-10:],
             x="playlist_id", y='passion_score', barmode='group',
             labels={'playlist_id': "<b>Playlist",'passion_score':'<b>Passion␣
 ↪Score'},
             color_discrete_sequence=px.colors.qualitative.Set1[3:5], text =␣
 ↪'listeners'
                  )
fig.update_xaxes(type='category',ticktext=playlists_new.
 ↪sort_values(by='passion_score')[-10:]['playlist_name'],
                 tickvals=playlists_new.sort_values(by='passion_score')[-10:
 ↪]['playlist_id']
                 ,tickangle = 340,showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:.2f}', textposition='outside',␣
 ↪textfont_size=10)
fig.update_layout(title={'text': '<b> Playlists with Highest Passion Scores</
 ↪b>','x':0.5})
fig.show()
```



1.4.10 Average Listening Time Per Stream

[99]:
```
#Average listening time per artist
artist_streams_lengths = df.groupby('artist_name')['stream_length'].
↪agg(['mean'])
artist_streams_lengths.reset_index(inplace=True)
artist_streams_lengths=artist_streams_lengths.rename(columns={'mean':
↪'avg_stream_time'})
```

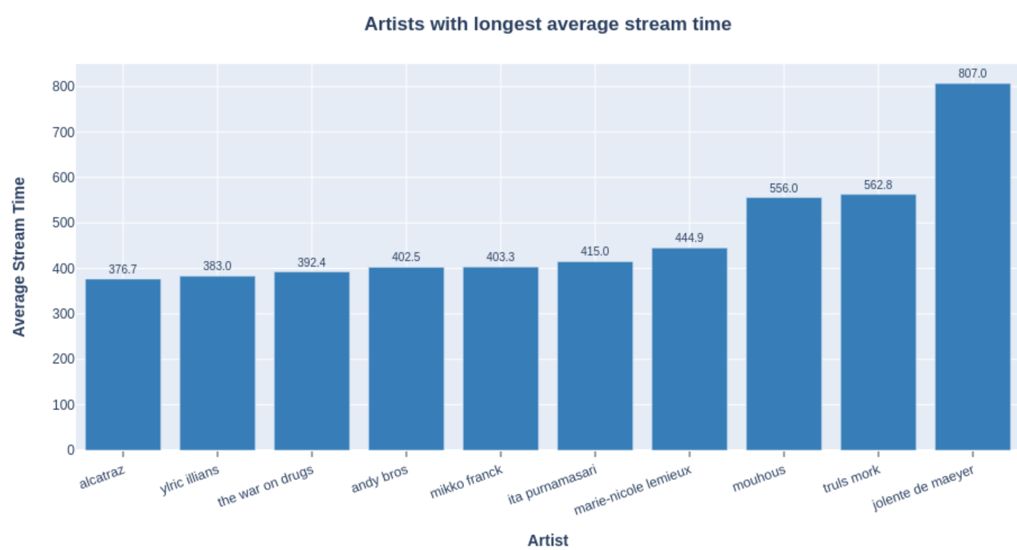
[100]:
```
artists_new = artists_new.
↪merge(artist_streams_lengths[['artist_name','avg_stream_time']],␣
↪on='artist_name')
```

[101]:
```
artists_new
```

[101]:
```
              artist_name   count  number_songs  success  playlists  listeners  \
0            charlie puth  447873            38        1       1747     367023
1                dua lipa  315663            50        1        892     260778
2            lukas graham  311271            22        1       1211     247580
3             cheat codes  255820            16        1       1218     225658
4              anne-marie  247934            28        1        757     220413
..                    ...     ...           ...      ...        ...
634      rebecka karlsson       1             1        0          0          1
635  los tres paraguayos        1             1        0          0          1
636                deuspi       1             1        0          1          1
637            vince pope       1             1        0          1          1
638            los romeos       1             1        0          0          1

     passion_score  avg_stream_time
0         1.220286       185.767816
1         1.210466       178.106221
2         1.257254       207.311259
3         1.133662       184.465644
4         1.124861       182.480559
..             ...              ...
634       1.000000       189.000000
635       1.000000       172.000000
636       1.000000       217.000000
637       1.000000        83.000000
638       1.000000       203.000000

[639 rows x 8 columns]
```

[102]:
```
#Average listening time per artist - visual (Top 10)
fig = px.bar(artists_new.sort_values(by='avg_stream_time')[-10:],
             x="artist_name", y='avg_stream_time', barmode='group',
```

```
              labels={'artist_name': "<b>Artist",'avg_stream_time':'<b>Average␣
↪Stream Time'},
              color_discrete_sequence=px.colors.qualitative.Set1[1:4], text =␣
↪'avg_stream_time'
                   )
fig.update_xaxes(tickangle = 340, showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:.1f}', textposition='outside',␣
↪textfont_size=10)
fig.update_layout(title={'text': '<b>Artists with longest average stream time</
↪b>','x':0.5})
fig.show()
```



1.4.11 Number of Repeat Listens

[103]:
```
df['date'] = pd.to_datetime(df.date)
```

[104]:
```
df1 = df.sort_values(by='date',ascending=True)
```

[105]:
```
#Checking repeat listens per artists per customer
repeat = pd.DataFrame(df1.
↪pivot_table(index=['customer_id','track_name','artist_name','date'],␣
↪aggfunc='size'))
repeat.reset_index(inplace=True)
repeat=repeat.rename(columns={0:'repeat_count'})
repeat['repeat_count'] = repeat['repeat_count']-1
repeat.sort_values(by='repeat_count')
```

[105]:
```
                               customer_id  \
0         0000074c93d4d2fe98a8b629f1a8b02d
2391060   aa2a8775f139891925d26557435693ba
2391061   aa2a8775f139891925d26557435693ba
2391062   aa2a8e76bc1f5a6806822376a04461cd
2391063   aa2a931bda00d34492651707c9f44800
…                                      …
1540578   6db4bd4c716d4eb77c9cb7ce2251b9b7
128779    093d4eb4c2e4aa9d2be7d83acfcdb943
2493951   b167d1676d04ccce1bb15b40eae59305
1245270   5885ca0aa24e14ee84890f142947dae8
3004800   d5394d44f4489772f1b680b26de16ad1

                                         track_name   artist_name        date  \
0                                              Flow      zak abel  2017-06-10
2391060                                     7 Years  lukas graham  2016-03-10
2391061                              Hotter Than Hell      dua lipa  2017-05-10
2391062                                  Come First     terror jr  2016-12-10
2391063                                   3 Strikes     terror jr  2017-06-10
…                                             …            …           …
1540578                                     7 Years  lukas graham  2016-03-10
128779                                          Sex   cheat codes  2016-08-10
2493951                                     7 Years  lukas graham  2016-05-10
1245270   i hate u, i love u (feat. olivia o'brien)         gnash  2016-07-10
3004800   i hate u, i love u (feat. olivia o'brien)         gnash  2016-07-10

         repeat_count
0                   0
2391060             0
2391061             0
2391062             0
2391063             0
…                   …
1540578            77
128779             80
2493951            88
1245270           100
3004800           120

[3616768 rows x 5 columns]
```

[106]:
```python
#creating df
repeat_artists = pd.DataFrame(repeat.groupby('artist_name')['repeat_count'].
 ↪sum())
repeat_artists.reset_index(inplace=True)
repeat_artists=repeat_artists.rename(columns={'repeat_count':'repeat_count'})
```

[107]:
```
repeat_artists.sort_values(by='repeat_count')
```

[107]:
```
       artist_name  repeat_count
319  local connect             0
236   jasmine kara             0
436        pitingo             0
437      pizzagang             0
438         plan 9             0
..             …             …
158       dua lipa         11671
101    cheat codes         11889
291           kyle         16876
98    charlie puth         23424
333   lukas graham         27625

[639 rows x 2 columns]
```

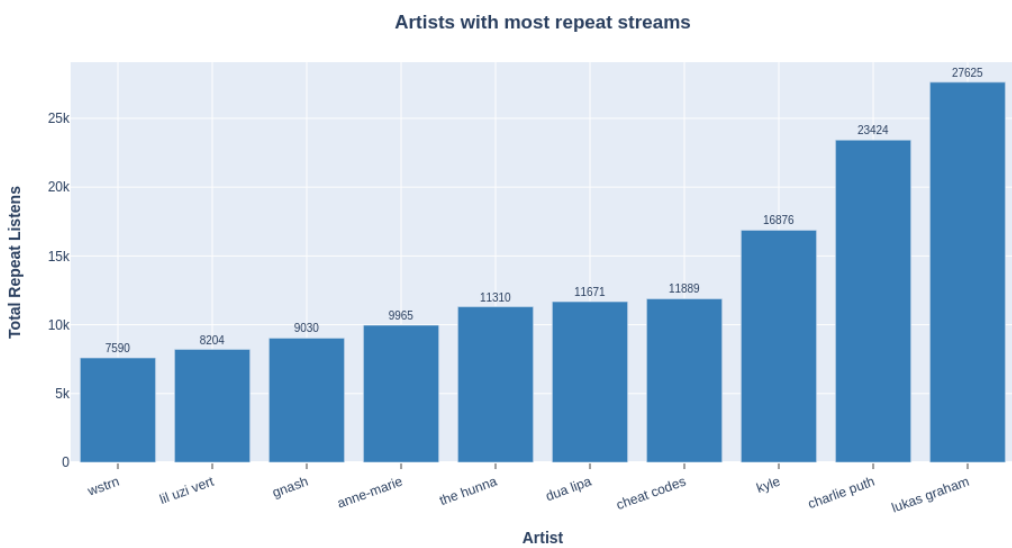
[108]:
```
#Adding to artists_new df
artists_new = artists_new.merge(repeat_artists[['artist_name','repeat_count']],␣
↪on='artist_name')
```

[109]:
```
artists_new
```

[109]:
```
             artist_name   count  number_songs  success  playlists  listeners  \
0           charlie puth  447873            38        1       1747     367023
1               dua lipa  315663            50        1        892     260778
2           lukas graham  311271            22        1       1211     247580
3            cheat codes  255820            16        1       1218     225658
4            anne-marie  247934            28        1        757     220413
..                     …       …             …        …          …
634      rebecka karlsson       1             1        0          0          1
635  los tres paraguayos       1             1        0          0          1
636               deuspi       1             1        0          1          1
637           vince pope       1             1        0          1          1
638           los romeos       1             1        0          0          1

     passion_score  avg_stream_time  repeat_count
0         1.220286       185.767816         23424
1         1.210466       178.106221         11671
2         1.257254       207.311259         27625
3         1.133662       184.465644         11889
4         1.124861       182.480559          9965
..             …               …            …
634       1.000000       189.000000             0
635       1.000000       172.000000             0
636       1.000000       217.000000             0
637       1.000000        83.000000             0
```

```
638         1.000000        203.000000             0

[639 rows x 9 columns]
```

[110]:
```
#Add Repeat Visualisation for artists
fig = px.bar(artists_new.sort_values(by='repeat_count')[-10:],
             x="artist_name", y='repeat_count', barmode='group',
             labels={'artist_name': "<b>Artist",'repeat_count':'<b>Total Repeat␣
↪Listens'},
             color_discrete_sequence=px.colors.qualitative.Set1[1:4], text =␣
↪'repeat_count'
                  )
fig.update_xaxes(tickangle = 340, showgrid=True, ticks="outside")
fig.update_traces(texttemplate='%{text:}', textposition='outside',␣
↪textfont_size=10)
fig.update_layout(title={'text': '<b>Artists with most repeat streams</b>','x':
↪0.5})
fig.show()
```

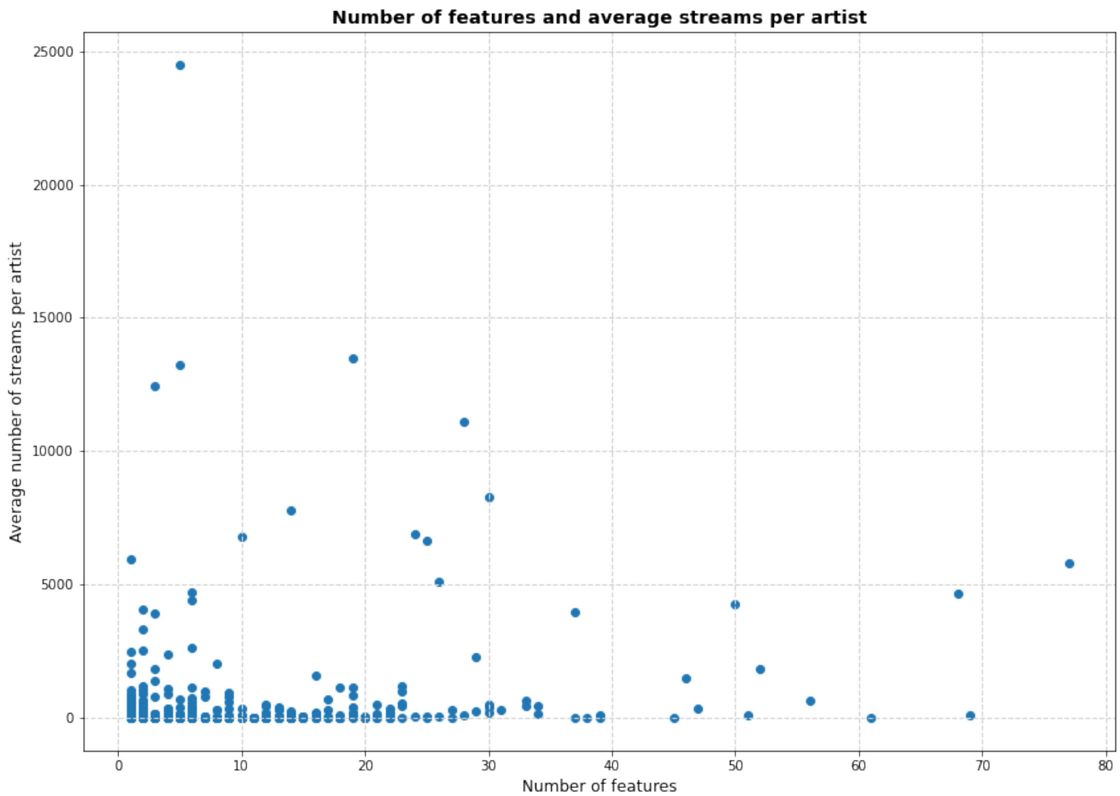


1.4.12 Number of Features

[111]:
```
#create new column that represents whether the track is a feature with another␣
↪artist
is_feature = [1 if 'feat.' in track_name else 0 for track_name in df.track_name]
df['is_feature'] = is_feature

#show corresponding value of is_feature to the particular song
df_copy = df.groupby(['artist_name', 'track_id']).agg({'is_feature': 'first'})
```

```
#group aggregated dataframe by artist and count how many song features in total
a = df_copy.groupby('artist_name')['is_feature'].count()
df["features_in_total"] = ''

#assign value of how many features an artist has in total back to the dataframe
features = {}
for i in range(len(df_copy.groupby('artist_name')['is_feature'].count().index)):
    features[a.index[i]] = a[i]

#assign value back to dataframe using apply lambda
df["features_in_total"] = df["artist_name"].apply(lambda x: features.get(x))
```

[112]:
```
#calculate average number of streams per artisted by number of features of the␣
↪artist
x = df.groupby('artist_name')['features_in_total'].first()
y = df.groupby(['artist_name'])['Unnamed: 0'].count() / df.
↪groupby(['artist_name'])['track_id'].nunique()


#plot figure
fig = plt.figure(figsize=(14,10))
ax = fig.add_subplot(111)
plt.scatter(x, y, label = 'artist')

plt.title('Number of features and average streams per artist', fontsize = 14,␣
↪weight = 'bold')

ax.set_xlabel('Number of features', fontsize = 12)
ax.set_ylabel('Average number of streams per artist', fontsize = 12)

ax.xaxis.grid(color='lightgray', linestyle='--', linewidth=1);
ax.yaxis.grid(color='lightgray', linestyle='--', linewidth=1);

plt.show();
```

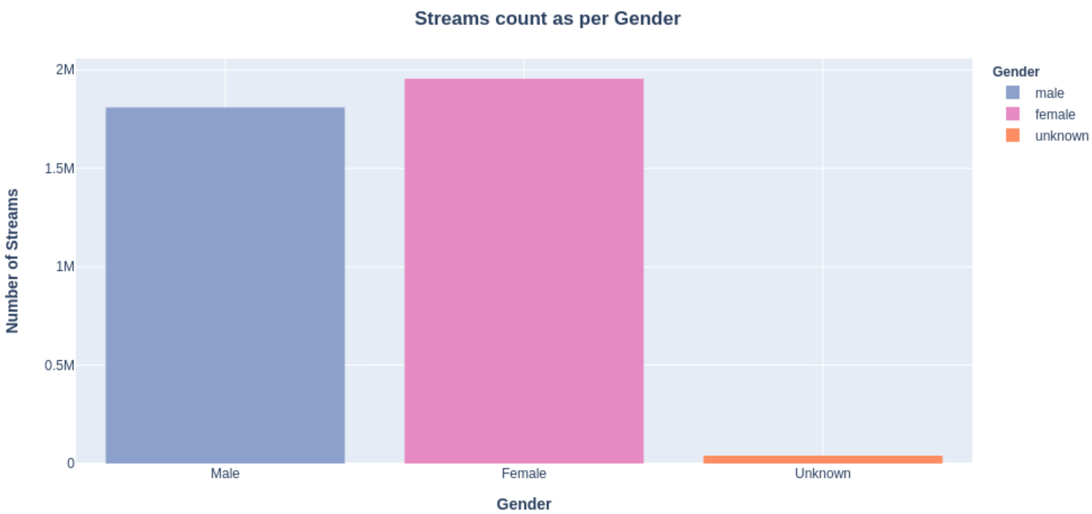


1.4.13 Gender

Gender Split

[113]:
```
#Filling NAs with unknown
df['gender']= df['gender'].fillna('unknown')
```

[114]:
```
## ADD GENDER SPLIT ACROSS STREAMS
fig = px.histogram(df, x="gender", color='gender',
                   labels={'gender': "<b>Gender</b>"},
                  ␣
 ↪color_discrete_sequence=['rgb(141,160,203)','rgb(231,138,195)','rgb(252,141,98)']
                  )
fig.update_xaxes(type='category',ticktext=["Male", "Female",'Unknown'],␣
 ↪tickvals=["0", "1",'2'], showgrid=True)
fig.update_layout(title={'text': '<b>Streams count as per Gender</b>','x':0.5},
                  yaxis_title_text='<b>Number of Streams</b>',
                 xaxis_title_text='<b>Gender</b>' )
fig.show()
```



Gender Domination per Artist

[115]:
```
#group by artist and gender and use max function to identify which gender␣
↪dominates for the particular artist
a = df.groupby(['artist_name', 'gender'])['Unnamed: 0'].count().sort_values().
↪groupby(level=0).tail(1).index
df["gender_domination"] = ''

#assign gender_domination value back to the dataframe for the visualisation
artist_type = {}
for i in range(len(df.groupby(['artist_name', 'gender'])['Unnamed: 0'].count().
↪sort_values().groupby(level=0).tail(1).index)):
    artist_type[a[i][0]] = a[i][1]

df["gender_domination"] = df["artist_name"].apply(lambda x: artist_type.get(x))
```

[116]:
```
#Compiling domination list as per artist
gender_dom = df.drop_duplicates(['artist_name'],keep = 'last')
```

[117]:
```
#Adding to df
artists_new = artists_new.
↪merge(gender_dom[['artist_name','gender_domination']], on='artist_name')
```

[118]:
```
artists_new
```

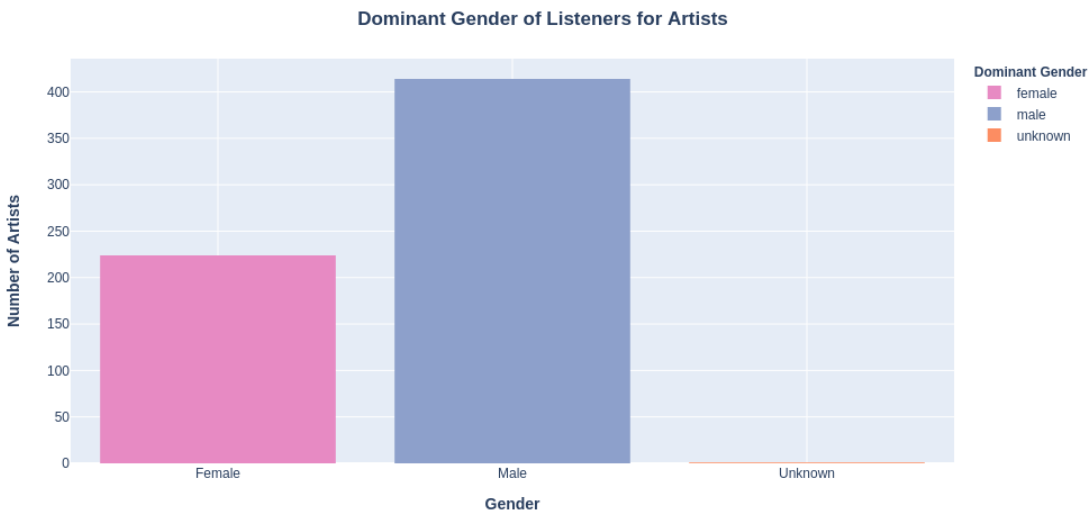
[118]:
```
    artist_name   count  number_songs  success  playlists  listeners  \
0  charlie puth  447873            38        1       1747     367023
1      dua lipa  315663            50        1        892     260778
```

```
2         lukas graham  311271          22       1       1211     247580
3          cheat codes  255820          16       1       1218     225658
4           anne-marie  247934          28       1        757     220413
..                 …      …          …      …      …       …
634    rebecka karlsson       1            1       0          0          1
635  los tres paraguayos      1            1       0          0          1
636              deuspi       1            1       0          1          1
637           vince pope      1            1       0          1          1
638          los romeos       1            1       0          0          1

     passion_score  avg_stream_time  repeat_count gender_domination
0         1.220286       185.767816         23424            female
1         1.210466       178.106221         11671            female
2         1.257254       207.311259         27625              male
3         1.133662       184.465644         11889            female
4         1.124861       182.480559          9965            female
..             …              …            …                …
634       1.000000       189.000000             0              male
635       1.000000       172.000000             0            female
636       1.000000       217.000000             0              male
637       1.000000        83.000000             0              male
638       1.000000       203.000000             0              male

[639 rows x 10 columns]
```

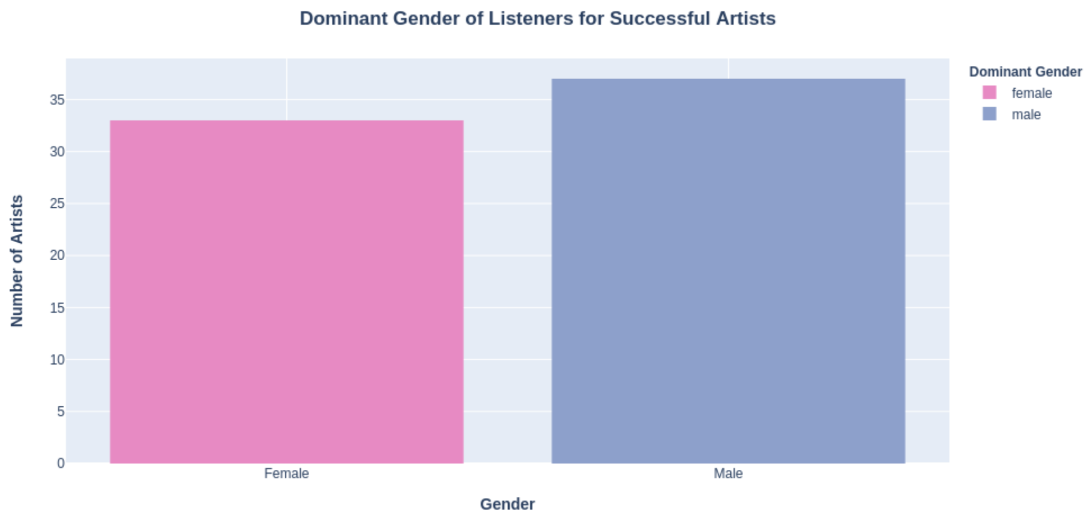
[119]:
```
artists_new['gender_domination'].value_counts()
```

[119]:
```
male       414
female     224
unknown      1
Name: gender_domination, dtype: int64
```

[120]:
```
#Visualising gender domination for each artists
fig = px.histogram(artists_new, x="gender_domination",␣
↪color='gender_domination',
                   labels={'gender_domination': "<b>Dominant Gender</b>"},
                   color_discrete_sequence=px.colors.qualitative.Set2_r[4:7]
                  )
fig.update_xaxes(type='category',ticktext=["Female", "Male",'Unknown'],␣
↪tickvals=["0", "1",'2'], showgrid=True)
fig.update_layout(title={'text': '<b>Dominant Gender of Listeners for Artists</
↪b>','x':0.5},
                  yaxis_title_text='<b>Number of Artists</b>',
                 xaxis_title_text='<b>Gender</b>' )
fig.show()
```



[121]:
```
#Visualising gender domination for each artists (SUCCESSFUL ONLY)
fig = px.histogram(artists_new[artists_new['success']==1],␣
↪x="gender_domination", color='gender_domination',
                   labels={'gender_domination': "<b>Dominant Gender</b>"},
                   color_discrete_sequence=px.colors.qualitative.Set2_r[4:7]
                  )
fig.update_xaxes(type='category',ticktext=["Female", "Male",'Unknown'],␣
↪tickvals=["0", "1",'2'], showgrid=True)
fig.update_layout(title={'text': '<b>Dominant Gender of Listeners for␣
↪Successful Artists</b>','x':0.5},
                  yaxis_title_text='<b>Number of Artists</b>',
                 xaxis_title_text='<b>Gender</b>' )
fig.show()
```



Percentage distribution of gender domination across all artists

[122]:
```
#calculate number of average streams per song per artist
overall = pd.DataFrame(round(df.groupby('artist_name')['Unnamed: 0'].count() / \
df.groupby('artist_name')['track_id'].nunique(), 2), columns = ['streams'])
```

[123]:
```
#convert index into columns
overall = overall.reset_index(level=['artist_name'])
```

[124]:
```
#create subset df
#group by artist and gender and use max function to identify which gender␣
↪dominates for the particular artist
a = df.groupby(['artist_name', 'gender'])['Unnamed: 0'].count().sort_values().
↪groupby(level=0).tail(1).index

#assign gender_domination value back to the dataframe for the visualisation
gender_type = {}
for i in range(len(df.groupby(['artist_name', 'gender'])['Unnamed: 0'].count().
↪sort_values().groupby(level=0).tail(1).index)):
    gender_type[a[i][0]] = a[i][1]

#assign value back to dataframe
overall["gender_domination"] = overall["artist_name"].apply(lambda x:␣
↪gender_type.get(x))
```

[125]:
```
#calculate percentage distribution
overall.groupby('gender_domination').agg({'gender_domination': 'count'}).
↪rename(columns={"gender_domination": "percentage_share"})\
```

```
    / len(overall)
```

[125]:

	percentage_share
gender_domination	
female	0.350548
male	0.647887
unknown	0.001565

Percentage distribution of gender domination across successful artists

[126]:
```
#create filter for top-tier playlists
playlist_filter = ['6FfOZSAN3N6u7v81uS7mxZ', '37i9dQZF1DX4JAvHpjipBk',␣
↪'37i9dQZF1DX5uokaTN4FTR', '37i9dQZF1DWVTKDs2aOkxu']

#create filter to only select artists who appear in top-tier playlist
artist_filter = [artist for artist in df[df['playlist_id'].
↪isin(playlist_filter)]['artist_name']]

#make list unique
artist_filter = list(dict.fromkeys(artist_filter))
```

[127]:
```
#calculate percentage distribution for artists who appear in one of the top-4␣
↪playlists
overall[overall['artist_name'].isin(artist_filter)].
↪groupby('gender_domination').agg({'gender_domination': 'count'})\
    .rename(columns={"gender_domination": "percentage_share"})/␣
↪len(overall[overall['artist_name'].isin(artist_filter)])
```

[127]:

	percentage_share
gender_domination	
female	0.471429
male	0.528571

Percentage distribution of gender domination for top 25 artists based on the number of playlists the appear

[128]:
```
# calculate percentage distribution among the top 25 artists based on number of␣
↪playlists they appear
# filter top 25 artists based on number of unique playlists they appear in
top25_artists = df.groupby('artist_name').agg({'playlist_id': 'nunique',␣
↪'gender_domination': 'first'})\
    .sort_values(by = 'playlist_id', ascending=False).nlargest(25, columns =␣
↪'playlist_id')


# calculate percentage distribution
```

```
top25_artists.groupby('gender_domination').count().
 ↪rename(columns={"playlist_id": "percentage_share"}) / len(top25_artists)
```

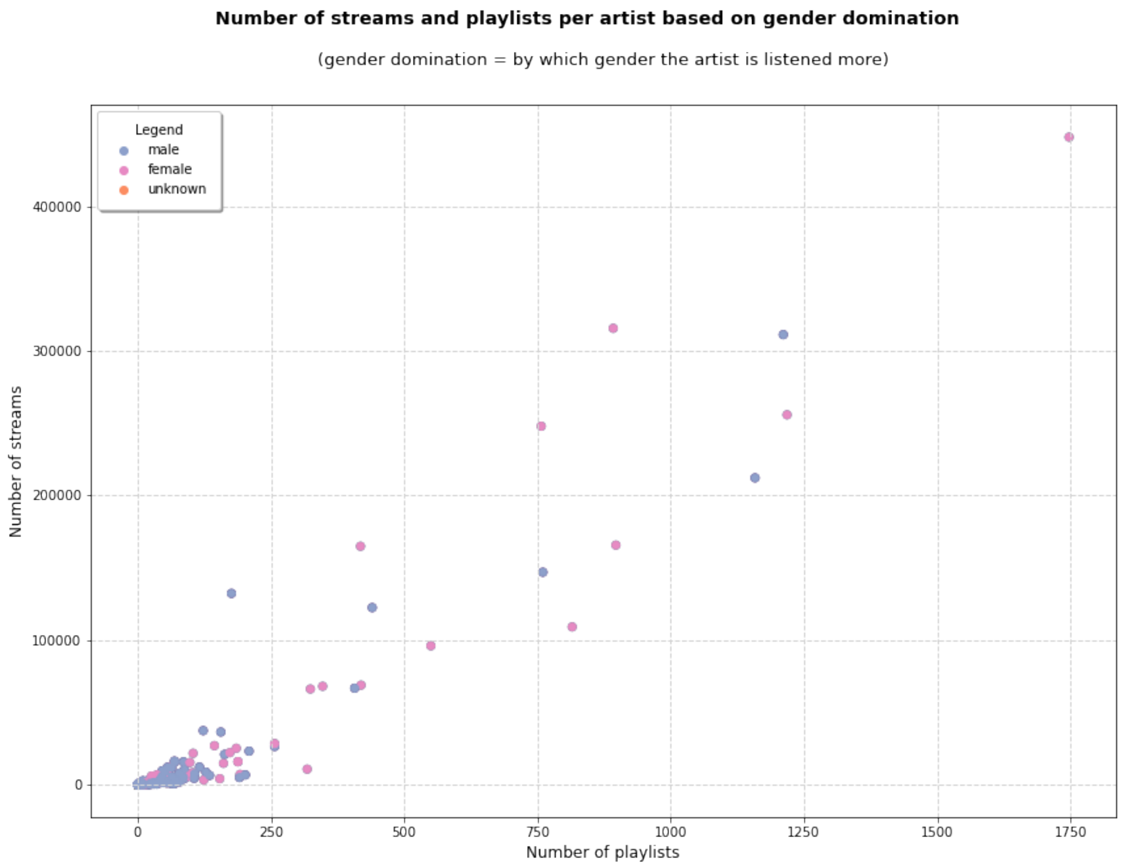
[128]:

	percentage_share
gender_domination	
female	0.64
male	0.36

[129]:

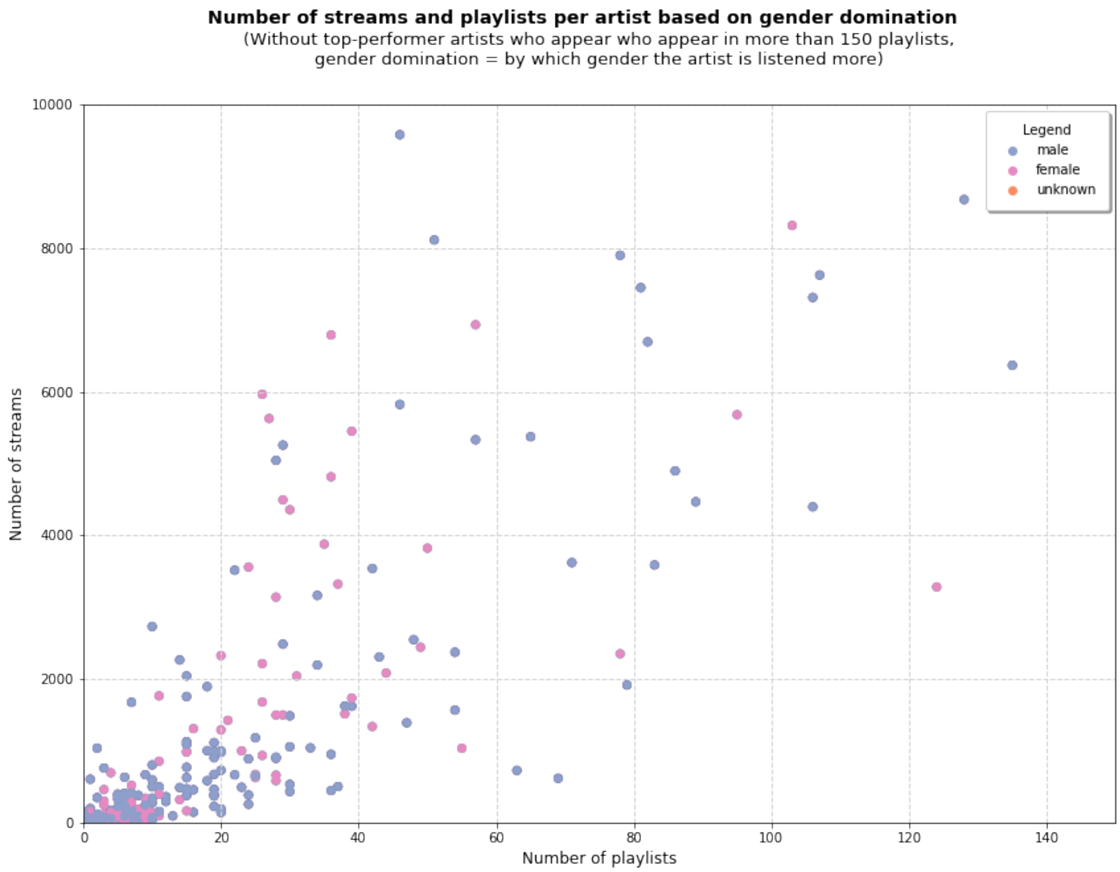
```
#create visualisation

#number of streams
y = df.groupby(['artist_name'])['Unnamed: 0'].count()

#number of playlists the artists appears
x = df.groupby('artist_name')['playlist_id'].nunique()

#color datapoints based on value of gender domination column
c = []
data = df.groupby('artist_name')['gender_domination'].unique()

for i in range(len(data)):
    if(data[i] == 'male'):
            c.append(0)
    if(data[i] == 'female'):
            c.append(1)
    #if gender is not applicable
    if(data[i] == 'unknown'):
            c.append(2)

classes = ['male', 'female', 'unknown']
colours = ListedColormap(['#8da0cb', '#e78ac3', '#fc8d62'])


#plot figure
fig = plt.figure(figsize=(14,10))
ax = fig.add_subplot(111)
ax.scatter(x, y, c=c)
plt.title('(gender domination = by which gender the artist is listened more)',␣
 ↪fontsize = 13, pad=30)
plt.suptitle('Number of streams and playlists per artist based on gender␣
 ↪domination', fontsize = 14, weight = 'bold')
ax.set_xlabel('Number of playlists', fontsize = 12)
ax.set_ylabel('Number of streams', fontsize = 12)

#limit axis to get rid of outliers and better analyse arists who appear in␣
 ↪fewer playlists
#ax.set_ylim(0, 10000)
```

```
#ax.set_xlim(0, 150)

ax.xaxis.grid(color='lightgray', linestyle='--', linewidth=1);
ax.yaxis.grid(color='lightgray', linestyle='--', linewidth=1);

scatter = plt.scatter(x,y,c = c, cmap =colours)

plt.legend(handles = scatter.legend_elements()[0], labels = classes,␣
↪shadow=True, title='Legend', borderpad = 1)


plt.show();
```




```
[130]: #create visualisation
#number of streams
y = df.groupby(['artist_name'])['Unnamed: 0'].count()

#number of playlists the artists appears
x = df.groupby('artist_name')['playlist_id'].nunique()
```

```python
#color datapoints based on value of gender domination column
c = []
data = df.groupby('artist_name')['gender_domination'].unique()

for i in range(len(data)):
    if(data[i] == 'male'):
            c.append(0)
    if(data[i] == 'female'):
            c.append(1)
    #if gender is not applicable
    if(data[i] == 'unknown'):
            c.append(2)

classes = ['male', 'female', 'unknown']
colours = ListedColormap(['#8da0cb', '#e78ac3', '#fc8d62'])

#plot figure
fig = plt.figure(figsize=(14,10))
ax = fig.add_subplot(111)
ax.scatter(x, y, c=c)
plt.title('(Without top-performer artists who appear who appear in more than␣
 ↪150 playlists,\ngender domination = by which gender the artist is listened␣
 ↪more)', fontsize = 13, pad=30)
plt.suptitle('Number of streams and playlists per artist based on gender␣
 ↪domination', fontsize = 14, weight = 'bold')
ax.set_xlabel('Number of playlists',  fontsize = 12)
ax.set_ylabel('Number of streams',  fontsize = 12)

#limit axis to get rid of outliers and better analyse arists who appear in␣
 ↪fewer playlists
ax.set_ylim(0, 10000)
ax.set_xlim(0, 150)

ax.xaxis.grid(color='lightgray', linestyle='--', linewidth=1);
ax.yaxis.grid(color='lightgray', linestyle='--', linewidth=1);



scatter = plt.scatter(x,y,c = c, cmap =colours)
plt.legend(handles = scatter.legend_elements()[0], labels = classes,␣
 ↪shadow=True, title='Legend', borderpad = 1)

# plt.savefig('gender_dominaton_150.pdf')
plt.show();
```



1.4.14 Age Group

[131]:
```
#calculate age based on birth_year
df['age'] = 2021 - df['birth_year']
```

[132]:
```
#replace nan values with 9999 to use them as a bin category
df['age'] = df['age'].replace(np.nan, 9999)
```

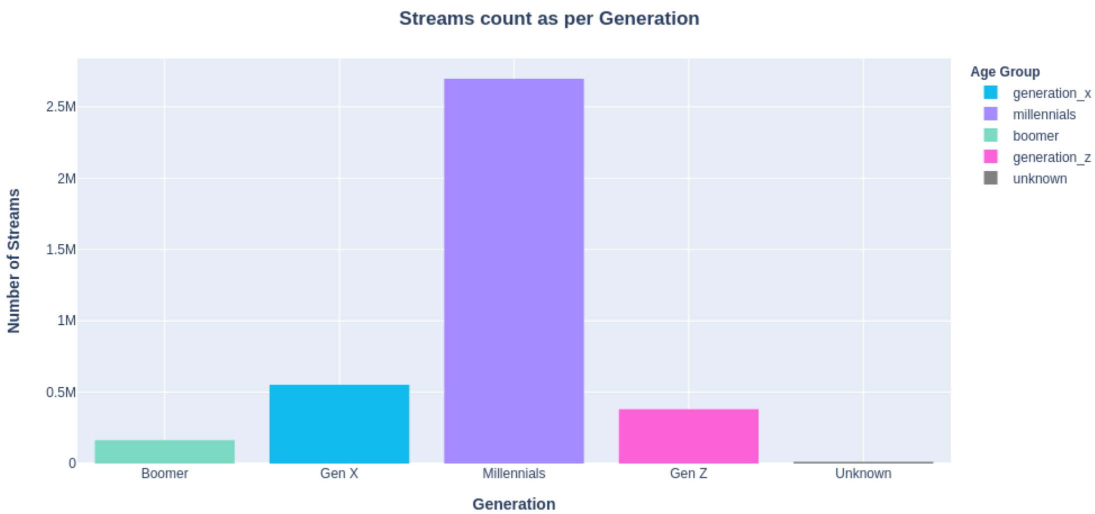
[133]:
```
#split ages into age groups
df["age_group"] = pd.cut(x=df['age'], bins=[0,21,37,53, 120,9999],
                         labels=['generation_z','millennials', 'generation_x',␣
 ↪'boomer','unknown'])
#group by artist and age_group and use max function to identify which␣
 ↪generation (generation z, millennials, generation x, boomer) dominates for␣
 ↪the particular artist
a = df.groupby(['artist_name', 'age_group'])['Unnamed: 0'].count().
 ↪sort_values().groupby(level=0).tail(1).index
df["generation_domination"] = ''
```

```
#assign value of which generation dominates for a particular artist back to the␣
↪dataframe
artist_type = {}
for i in range(len(df.groupby(['artist_name', 'age_group'])['Unnamed: 0'].
↪count().sort_values().groupby(level=0).tail(1).index)):
    artist_type[a[i][0]] = a[i][1]

#assign value back to dataframe
df["generation_domination"] = df["artist_name"].apply(lambda x: artist_type.
↪get(x))
```

[134]:
```
df['age'].isnull().sum()
```

[134]: 0

[135]:
```
## Generation visual as per stream count
fig = px.histogram(df, x="age_group", color='age_group',
                   labels={'age_group': "<b>Age Group</b>"},
                  ␣
↪color_discrete_sequence=['#0FBAEC','#A58AFF','#79D9C3','#FB61D7' ,'gray']

                  )
fig.update_xaxes(type='category',ticktext=["Boomer",'Gen X','Millennials','Gen␣
↪Z','Unknown'],
                 tickvals=["0", "1",'2','3','4'], showgrid=True)
fig.update_layout(title={'text': '<b>Streams count as per Generation</b>','x':0.
↪5},
                  yaxis_title_text='<b>Number of Streams</b>',
                 xaxis_title_text='<b>Generation</b>',
                 xaxis={'categoryorder':'array', 'categoryarray':
↪['boomer','generation_x',
                                                                      ␣
↪'millennials','generation_z',
                                                                       'unknown']}␣
↪)
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```

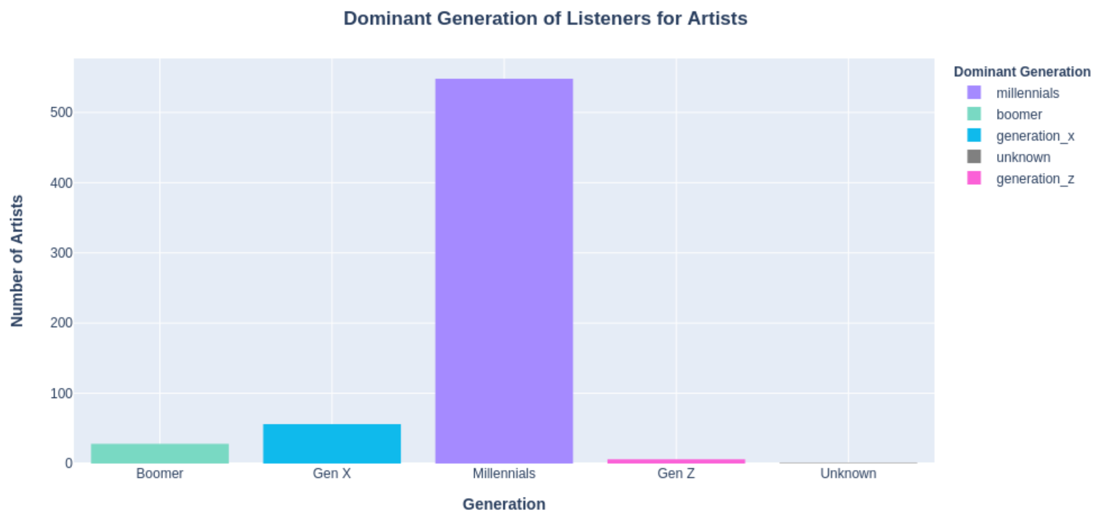


[136]:
```
#Adding to df for artists
generation_dom = df.drop_duplicates(['artist_name'],keep = 'last')
artists_new = artists_new.
↪merge(generation_dom[['artist_name','generation_domination']],␣
↪on='artist_name')
artists_new
```

[136]:
```
              artist_name   count  number_songs  success  playlists  listeners  \
0            charlie puth  447873            38        1       1747     367023
1                dua lipa  315663            50        1        892     260778
2            lukas graham  311271            22        1       1211     247580
3             cheat codes  255820            16        1       1218     225658
4              anne-marie  247934            28        1        757     220413
..                    ...     ...           ...      ...        ...        ...
634      rebecka karlsson       1             1        0          0          1
635  los tres paraguayos       1             1        0          0          1
636                deuspi       1             1        0          1          1
637            vince pope       1             1        0          1          1
638            los romeos       1             1        0          0          1

     passion_score  avg_stream_time  repeat_count gender_domination  \
0         1.220286       185.767816         23424            female
1         1.210466       178.106221         11671            female
2         1.257254       207.311259         27625              male
3         1.133662       184.465644         11889            female
4         1.124861       182.480559          9965            female
..             ...              ...           ...               ...
634       1.000000       189.000000             0              male
```

```
635       1.000000       172.000000             0             female
636       1.000000       217.000000             0               male
637       1.000000        83.000000             0               male
638       1.000000       203.000000             0               male

    generation_domination
0             millennials
1             millennials
2             millennials
3             millennials
4             millennials
..                    …
634          generation_x
635           millennials
636          generation_x
637           millennials
638          generation_x

[639 rows x 11 columns]
```

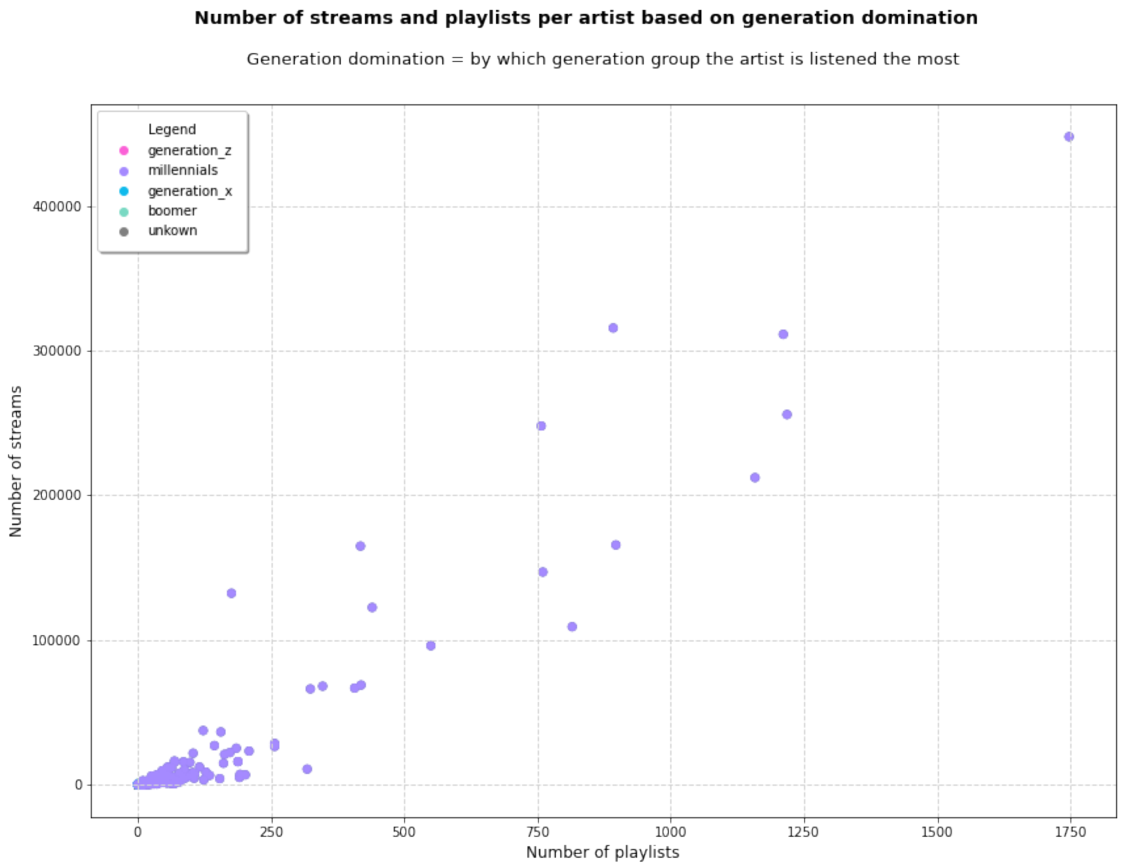
[137]:
```
#Dominant Generation split as per artists
fig = px.histogram(artists_new, x="generation_domination",␣
↪color='generation_domination',
                   labels={'generation_domination': "<b>Dominant Generation</
↪b>"},
                  ␣
↪color_discrete_sequence=['#A58AFF','#79D9C3','#0FBAEC','gray','#FB61D7']
                  )
fig.update_xaxes(type='category',ticktext=["Boomer",'Gen X','Millennials','Gen␣
↪Z','Unknown'],
                 tickvals=["0", "1",'2','3','4'], showgrid=True)
fig.update_layout(title={'text': '<b>Dominant Generation of Listeners for␣
↪Artists</b>','x':0.5},
                  yaxis_title_text='<b>Number of Artists</b>',
                 xaxis_title_text='<b>Generation</b>',
                  xaxis={'categoryorder':'array', 'categoryarray':
↪['boomer','generation_x',
                                                                      ␣
↪'millennials','generation_z',
                                                                       'unknown']}␣
↪)
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```



[138]:
```
#Dominant Generation split as per artists (SUCCESSFUL ONLY)
fig = px.histogram(artists_new[artists_new['success']==1],␣
↪x="generation_domination", color='generation_domination',
                   labels={'generation_domination': "<b>Dominant Generation</
↪b>"},
                  ␣
↪color_discrete_sequence=['#A58AFF','#79D9C3','#0FBAEC','gray','#FB61D7']
                  )
fig.update_xaxes(type='category',ticktext=["Boomer",'Gen X','Millennials','Gen␣
↪Z','Unknown'],
                 tickvals=["0", "1",'2','3','4'], showgrid=True)
fig.update_layout(title={'text': '<b>Dominant Generation of Listeners for␣
↪Successful Artists</b>','x':0.5},
                  yaxis_title_text='<b>Number of Artists</b>',
                 xaxis_title_text='<b>Generation</b>',
                  xaxis={'categoryorder':'array', 'categoryarray':
↪['boomer','generation_x',
                                                                              ␣
↪'millennials','generation_z',
                                                                               'unknown']}␣
↪)
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```

[139]:
```
#create visualisation



#number of streams
y = df.groupby(['artist_name'])['Unnamed: 0'].count()

#number of playlists the artists appears
x = df.groupby('artist_name')['playlist_id'].nunique()

#color datapoints based on value of generation domination column
c = []
data = df.groupby('artist_name')['generation_domination'].unique()

for i in range(len(data)):
    if(data[i] == 'generation_z'):
            c.append(0)
    if(data[i] == 'millennials'):
            c.append(1)
    if(data[i] == 'generation_x'):
            c.append(2)
    if(data[i] == 'boomer'):
            c.append(3)
    if(data[i] == 'unknown'):
            c.append(4)


classes = ['generation_z','millennials', 'generation_x', 'boomer','unkown']
colours = ListedColormap(['#FB61D7', '#A58AFF', '#0FBAEC', '#79D9C3','gray'])
```

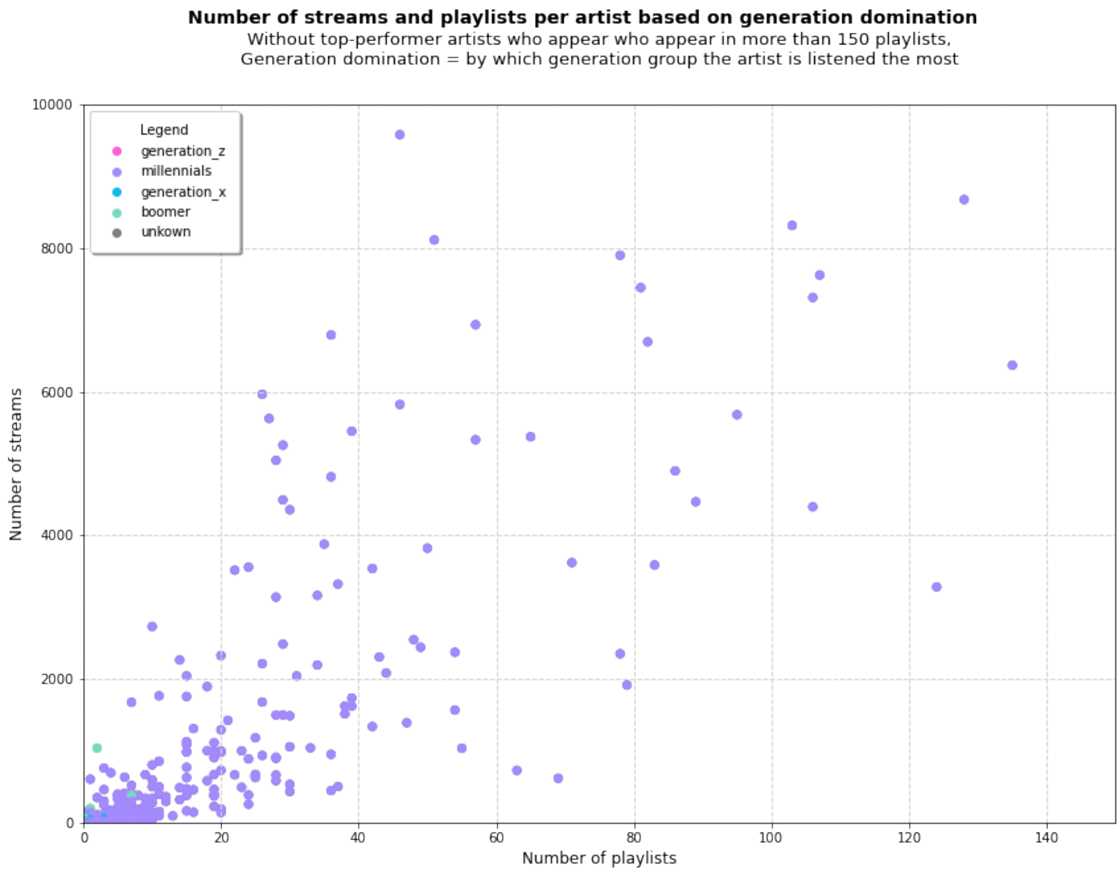
```python
#plot figure
fig = plt.figure(figsize=(14,10))
ax = fig.add_subplot(111)
ax.scatter(x, y, c=c)

plt.suptitle('Number of streams and playlists per artist based on generation␣
↪domination', fontsize = 14, weight = 'bold')
plt.title('Generation domination = by which generation group the artist is␣
↪listened the most', fontsize = 13, pad=30)

ax.set_xlabel('Number of playlists', fontsize = 12)
ax.set_ylabel('Number of streams', fontsize = 12)

#limit axis to get rid of outliers and better analyse arists who appear in␣
↪fewer playlists
#ax.set_ylim(0, 10000)
#ax.set_xlim(0, 150)

ax.xaxis.grid(color='lightgray', linestyle='--', linewidth=1);
ax.yaxis.grid(color='lightgray', linestyle='--', linewidth=1);

scatter = plt.scatter(x,y,c = c, cmap =colours)
plt.legend(handles = scatter.legend_elements()[0], labels = classes,␣
↪shadow=True, title='Legend', borderpad = 1)


plt.show();
```



[140]:

```
#create visualisation


#number of streams
y = df.groupby(['artist_name'])['Unnamed: 0'].count()

#number of playlists the artists appears
x = df.groupby('artist_name')['playlist_id'].nunique()

#color datapoints based on value of generation domination column
c = []
data = df.groupby('artist_name')['generation_domination'].unique()

for i in range(len(data)):
    if(data[i] == 'generation_z'):
            c.append(0)
    if(data[i] == 'millennials'):
            c.append(1)
    if(data[i] == 'generation_x'):
            c.append(2)
```

```
    if(data[i] == 'boomer'):
            c.append(3)
    if(data[i] == 'unknown'):
            c.append(4)


classes = ['generation_z','millennials', 'generation_x', 'boomer','unkown']
colours = ListedColormap(['#FB61D7', '#A58AFF', '#0FBAEC', '#79D9C3','gray'])


#plot figure
fig = plt.figure(figsize=(14,10))
ax = fig.add_subplot(111)
ax.scatter(x, y, c=c)

plt.suptitle('Number of streams and playlists per artist based on generation␣
 ↪domination', fontsize = 14, weight = 'bold')
plt.title('Without top-performer artists who appear who appear in more than 150␣
 ↪playlists,\nGeneration domination = by which generation group the artist is␣
 ↪listened the most', fontsize = 13, pad=30)

ax.set_xlabel('Number of playlists', fontsize = 12)
ax.set_ylabel('Number of streams', fontsize = 12)

#limit axis to get rid of outliers and better analyse arists who appear in␣
 ↪fewer playlists
ax.set_ylim(0, 10000)
ax.set_xlim(0, 150)

ax.xaxis.grid(color='lightgray', linestyle='--', linewidth=1);
ax.yaxis.grid(color='lightgray', linestyle='--', linewidth=1);

scatter = plt.scatter(x,y,c = c, cmap =colours)
plt.legend(handles = scatter.legend_elements()[0], labels = classes,␣
 ↪shadow=True, title='Legend', borderpad = 1)

plt.show();
```



Percentage distribution of generation domination across all artists

[141]:
```
#assign generation domination information to df subset
generation_type = {}
for i in range(len(df.groupby(['artist_name', 'age_group'])['Unnamed: 0'].
↪count().sort_values().groupby(level=0).tail(1).index)):
    generation_type[a[i][0]] = a[i][1]

#assign value back to dataframe
overall["generation_domination"] = overall["artist_name"].apply(lambda x:␣
↪generation_type.get(x))
```

[142]:
```
#calculate percentage distribution across all artists
overall.groupby('generation_domination').agg({'generation_domination':␣
↪'count'}).rename(columns={'generation_domination': 'percentage_share'})\
    / len(overall)
```

[142]:
```
                       percentage_share
generation_domination
boomer                         0.043818
```

```
generation_x                    0.087637
generation_z                    0.009390
millennials                     0.857590
unknown                         0.001565
```

Percentage distribution of gender domination across successful artists

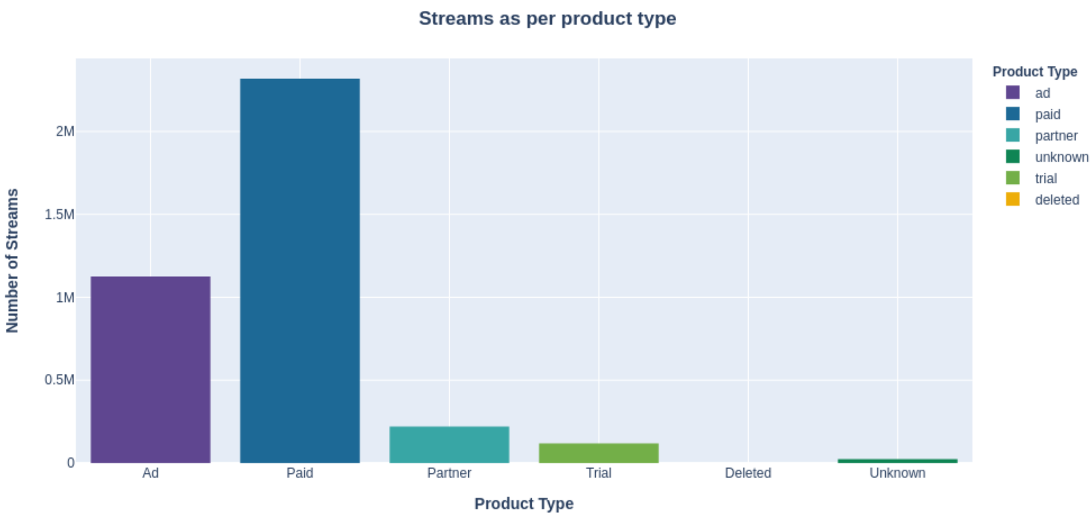
[143]:
```
#calculate percentage distribution across artists who appear in one of the␣
↪top-4 playlists
overall[overall['artist_name'].isin(artist_filter)].
↪groupby('generation_domination').agg({'generation_domination': 'count'})\
    .rename(columns={"generation_domination": "percentage_share"})/␣
↪len(overall[overall['artist_name'].isin(artist_filter)])
```

[143]:
```
                      percentage_share
generation_domination
millennials                        1.0
```

1.4.15 Product Type

[144]:
```
# Replacing NAN with unknowns
df['user_product_type'] = df['user_product_type'].replace(np.nan, 'unknown')
```

[145]:
```
#Visual for product type against streams
fig = px.histogram(df, x="user_product_type", color='user_product_type',
                   labels={'user_product_type': "<b>Product Type</b>"},
                   color_discrete_sequence=px.colors.qualitative.Prism
                  )
fig.
↪update_xaxes(type='category',ticktext=["Ad",'Paid','Partner','Trial','Deleted','Unknown'],
                 tickvals=["0", "1",'2','3','4','5'], showgrid=True)
fig.update_layout(title={'text': '<b>Streams as per product type</b>','x':0.5},
                  yaxis_title_text='<b>Number of Streams</b>',
                 xaxis_title_text='<b>Product Type</b>',
                  xaxis={'categoryorder':'array',
                         'categoryarray':
↪["ad",'paid','partner','trial','deleted','unknown']}
                 )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```



[ ]:
```

```

1.4.16 Yearly Split

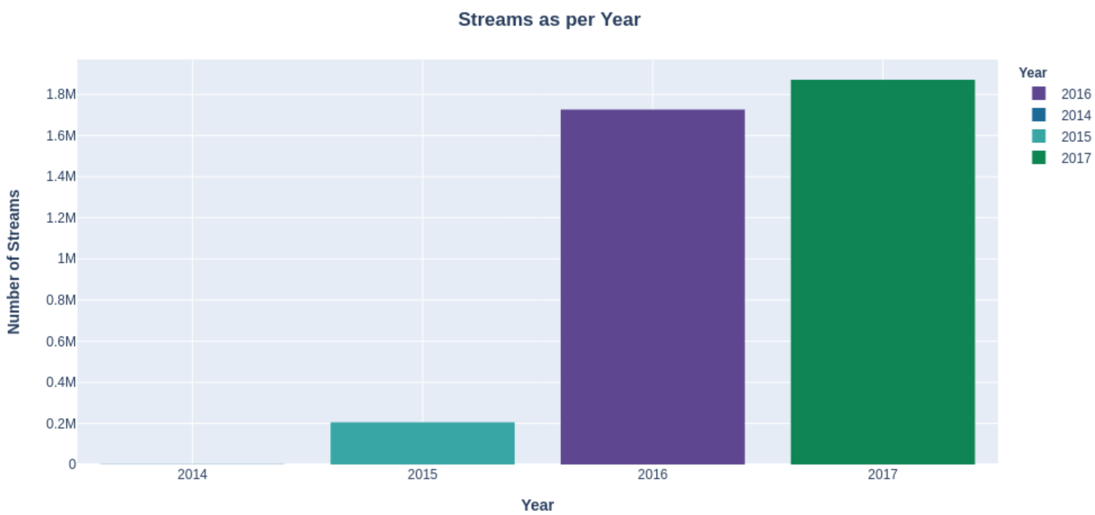
[146]:
```
#NOT CONTRIBUTING TO THE PROBLEM
```

[147]:
```
#Yearly split
df['year'].unique()
year_streams = df.groupby(['year'])['log_time'].agg(['count'])
year_streams
```

[147]:
```
        count
year
2014     1102
2015   205293
2016  1727360
2017  1871744
```

[148]:
```
#Split of streaming data as per year
fig = px.histogram(df, x="year", color='year',
                   labels={'year': "<b>Year</b>"},
                   color_discrete_sequence=px.colors.qualitative.Prism
                  )
fig.update_xaxes(type='category',ticktext=["2014",'2015','2016','2017'],
                 tickvals=["0", "1",'2','3'], showgrid=True)
fig.update_layout(title={'text': '<b>Streams as per Year</b>','x':0.5},
                  yaxis_title_text='<b>Number of Streams</b>',
                 xaxis_title_text='<b>Year</b>',
```

```
                    xaxis={'categoryorder':'array',
                           'categoryarray':["2014",'2015','2016','2017']}
                   )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```



1.4.17 Seasons

[149]:
```
#convert date into datetime format to use dt operator
df['date'] = pd.to_datetime(df['date'])
df['month'] = df['date'].dt.month
```

[150]:
```
df.month.unique()
```

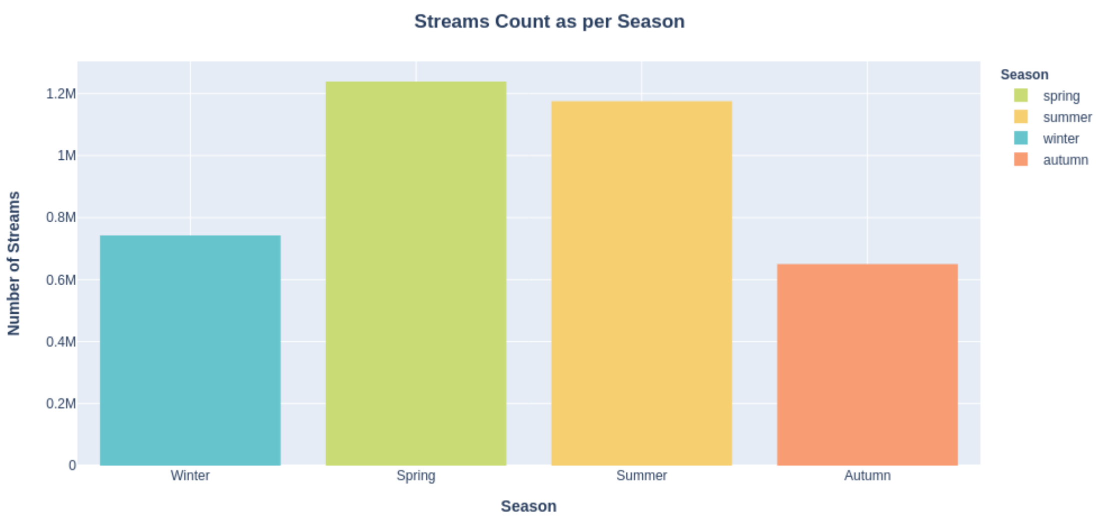
[150]:
```
array([ 5,  6,  4,  2,  3,  7,  8,  9,  1, 10, 11, 12])
```

[151]:
```
#use mathematical logic to convert month into season:
#[1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 1]
df['season_id'] = df['month'].replace({1: 1, 2 : 1, 3: 2, 4: 2, 5:2, 6:3, 7:3,␣
 ↪8:3, 9:4, 10:4,11:4, 12:1})
```

[152]:
```
#create season name
df['season_name'] = df['season_id']
df['season_name'] = df['season_name'].replace({1: 'winter', 2 : 'spring', 3:␣
 ↪'summer', 4: 'autumn'})
```

[153]:
```
#Split of streaming data as per season
fig = px.histogram(df, x="season_name", color='season_name',
                   labels={'season_name': "<b>Season</b>"},
```

```
                    color_discrete_map={"winter": 'rgb(102, 197, 204)',
                                        'spring': 'rgb(201, 219, 116)',
                                        'summer': 'rgb(246, 207, 113)',
                                        'autumn':'rgb(248, 156, 116)' }
                   )
fig.update_xaxes(type='category',ticktext=['Winter','Spring','Summer','Autumn'],
                 tickvals=["0", "1",'2','3'], showgrid=True)
fig.update_layout(title={'text': '<b>Streams Count as per Season</b>','x':0.5},
                  yaxis_title_text='<b>Number of Streams</b>',
                 xaxis_title_text='<b>Season</b>',
                  xaxis={'categoryorder':'array', 'categoryarray':
 ↪['winter','spring','summer','autumn']} )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```



[154]:
```
#group by artist and season and use max function to identify which season␣
 ↪dominates for the particular artist
a = df.groupby(['artist_name', 'season_name'])['Unnamed: 0'].count().
 ↪sort_values().groupby(level=0).tail(1).index
df["season_domination"] = ''

#assign value of which season dominates for a particular artist back to the␣
 ↪dataframe
season_type = {}
for i in range(len(df.groupby(['artist_name', 'season_name'])['Unnamed: 0'].
 ↪count().sort_values().groupby(level=0).tail(1).index)):
    season_type[a[i][0]] = a[i][1]

#assign value back to dataframe using apply lambda
```

```
df["season_domination"] = df["artist_name"].apply(lambda x: season_type.get(x))
```

Percentage distribution of season domination across all artists

[155]:
```
#assign season domination information to df subset
season_type = {}
for i in range(len(df.groupby(['artist_name', 'season_name'])['Unnamed: 0'].
 ↪count().sort_values().groupby(level=0).tail(1).index)):
    season_type[a[i][0]] = a[i][1]

#assign value back to dataframe
overall["season_domination"] = overall["artist_name"].apply(lambda x:␣
 ↪season_type.get(x))
```

[156]:
```
#calculate percentage distribution across all artists
overall.groupby('season_domination').agg({'season_domination': 'count'}).
 ↪rename(columns={'season_domination': 'percentage_share'})\
    / len(overall)
```

[156]:
```
                   percentage_share
season_domination
autumn                     0.079812
spring                     0.388106
summer                     0.370892
winter                     0.161189
```

Percentage distribution of season domination across successful artists

[157]:
```
#calculate percentage distribution across artists who appear in one of the␣
 ↪top-4 playlists
overall[overall['artist_name'].isin(artist_filter)].
 ↪groupby('season_domination').agg({'season_domination': 'count'})\
    .rename(columns={"season_domination": "percentage_share"})/␣
 ↪len(overall[overall['artist_name'].isin(artist_filter)])
```

[157]:
```
                   percentage_share
season_domination
autumn                     0.057143
spring                     0.500000
summer                     0.314286
winter                     0.128571
```

[158]:
```
#assign season domination information to df subset
season_type = {}
for i in range(len(df.groupby(['artist_name', 'season_name'])['Unnamed: 0'].
 ↪count().sort_values().groupby(level=0).tail(1).index)):
    season_type[a[i][0]] = a[i][1]
```

```
#assign value back to dataframe
overall["season_domination"] = overall["artist_name"].apply(lambda x:␣
↪season_type.get(x))
```

[159]:
```
#calculate percentage distribution across all artists
overall.groupby('season_domination').agg({'season_domination': 'count'}).
↪rename(columns={'season_domination': 'percentage_share'})\
    / len(overall)
```

[159]:
```
                   percentage_share
season_domination
autumn                     0.079812
spring                     0.388106
summer                     0.370892
winter                     0.161189
```

[160]:
```
##RUN ONLY ONCE!

#Season domination added to artists_new df
season_dom = df.drop_duplicates(['artist_name'],keep = 'last')
artists_new = artists_new.
↪merge(season_dom[['artist_name','season_domination']], on='artist_name')
```

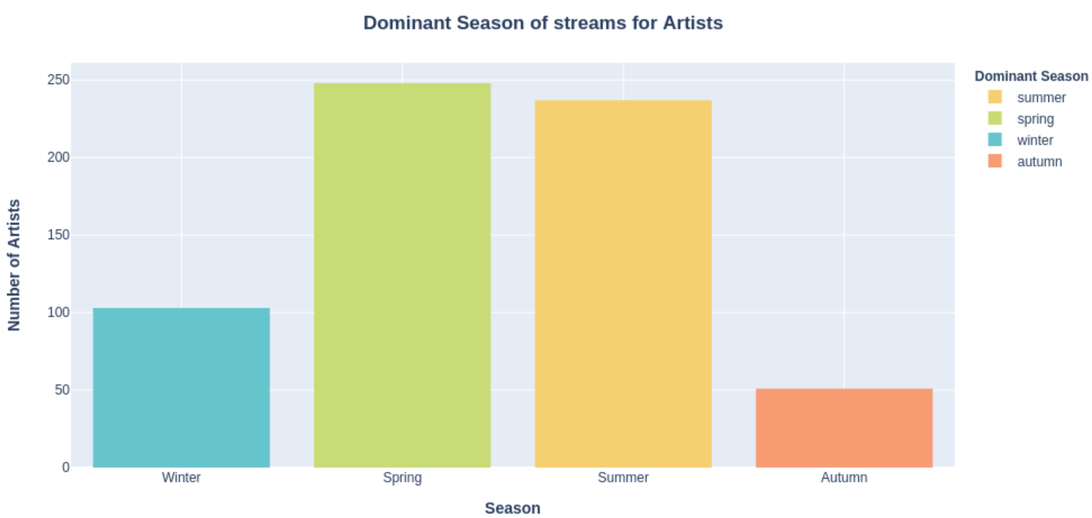
[161]:
```
artists_new
```

[161]:
```
             artist_name   count  number_songs  success  playlists  listeners  \
0           charlie puth  447873            38        1       1747     367023
1               dua lipa  315663            50        1        892     260778
2           lukas graham  311271            22        1       1211     247580
3            cheat codes  255820            16        1       1218     225658
4             anne-marie  247934            28        1        757     220413
..                   …       …             …        …          …          …
634     rebecka karlsson       1             1        0          0          1
635  los tres paraguayos       1             1        0          0          1
636               deuspi       1             1        0          1          1
637           vince pope       1             1        0          1          1
638           los romeos       1             1        0          0          1

     passion_score  avg_stream_time  repeat_count gender_domination  \
0         1.220286       185.767816         23424            female
1         1.210466       178.106221         11671            female
2         1.257254       207.311259         27625              male
3         1.133662       184.465644         11889            female
4         1.124861       182.480559          9965            female
..             …                …             …                 …
634       1.000000       189.000000             0              male
```

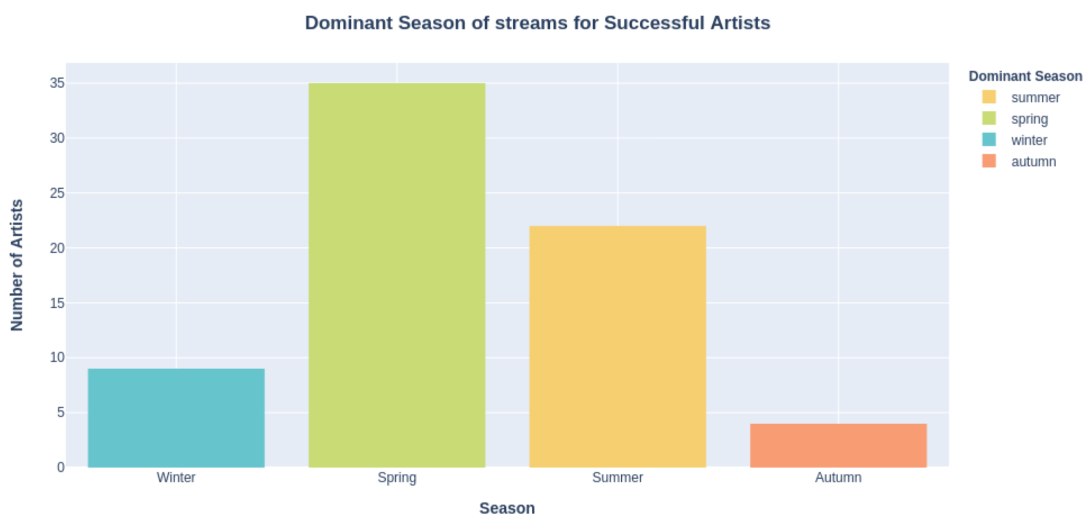
```
635       1.000000     172.000000            0           female
636       1.000000     217.000000            0             male
637       1.000000      83.000000            0             male
638       1.000000     203.000000            0             male

    generation_domination season_domination
0             millennials            summer
1             millennials            summer
2             millennials            spring
3             millennials            summer
4             millennials            spring
..                    …                …
634          generation_x            summer
635           millennials            summer
636          generation_x            summer
637           millennials            spring
638          generation_x            summer

[639 rows x 12 columns]
```

[162]:

```
#Dominating season as per artists
fig = px.histogram(artists_new, x="season_domination",␣
 ↪color='season_domination',
                   labels={'season_domination': "<b>Dominant Season</b>"},
                               color_discrete_map={"winter": 'rgb(102, 197,␣
 ↪204)',
                                      'spring': 'rgb(201, 219, 116)',
                                      'summer': 'rgb(246, 207, 113)',
                                      'autumn':'rgb(248, 156, 116)' })
fig.update_xaxes(type='category',ticktext=['Winter','Spring','Summer','Autumn'],
                 tickvals=["0", "1",'2','3'], showgrid=True)
fig.update_layout(title={'text': '<b>Dominant Season of streams for Artists</
 ↪b>','x':0.5},
                  yaxis_title_text='<b>Number of Artists</b>',
                 xaxis_title_text='<b>Season</b>',
                  xaxis={'categoryorder':'array', 'categoryarray':
 ↪['winter','spring','summer','autumn']} )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```

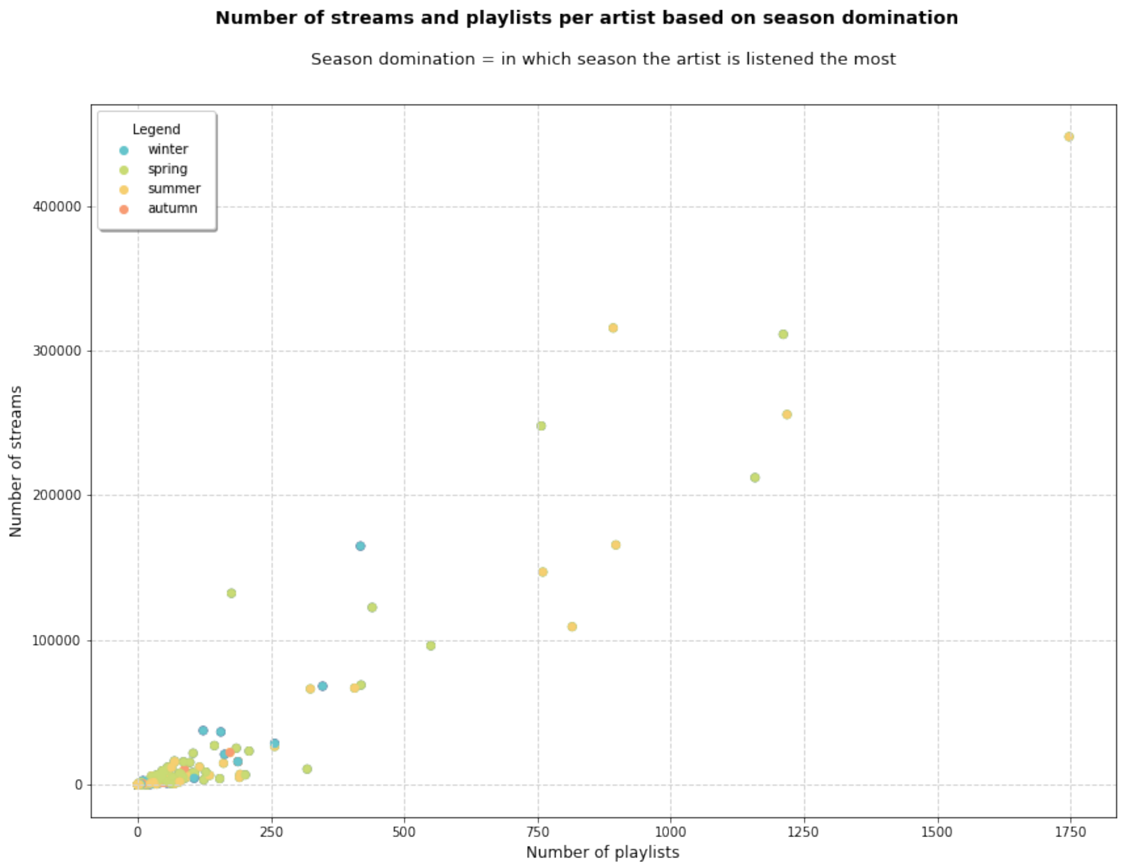


[163]:
```python
#Dominating season as per artists (SUCCESSFUL ONLY)
fig = px.histogram(artists_new[artists_new['success']==1],␣
↪x="season_domination", color='season_domination',
                   labels={'season_domination': "<b>Dominant Season</b>"},
                         color_discrete_map={"winter": 'rgb(102, 197,␣
↪204)',
                                             'spring': 'rgb(201, 219, 116)',
                                             'summer': 'rgb(246, 207, 113)',
                                             'autumn':'rgb(248, 156, 116)' })
fig.update_xaxes(type='category',ticktext=['Winter','Spring','Summer','Autumn'],
                 tickvals=["0", "1",'2','3'], showgrid=True)
fig.update_layout(title={'text': '<b>Dominant Season of streams for Successful␣
↪Artists</b>','x':0.5},
                  yaxis_title_text='<b>Number of Artists</b>',
                 xaxis_title_text='<b>Season</b>',
                  xaxis={'categoryorder':'array', 'categoryarray':
↪['winter','spring','summer','autumn']} )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```



[164]:
```
#number of streams
y = df.groupby(['artist_name'])['Unnamed: 0'].count()

#number of playlists the artists appears
x = df.groupby('artist_name')['playlist_id'].nunique()

#color datapoints based on value of season domination column
c = []
data = df.groupby('artist_name')['season_domination'].unique()

for i in range(len(data)):
    if(data[i] == 'winter'):
            c.append(0)
    if(data[i] == 'spring'):
            c.append(1)
    if(data[i] == 'summer'):
            c.append(2)
    if(data[i] == 'autumn'):
            c.append(3)


classes = ['winter','spring', 'summer', 'autumn']
colours = ListedColormap(['#66C5CC', '#C9DB74',
                          '#F6CF71', '#F89C74'])


#plot figure
fig = plt.figure(figsize=(14,10))
```

```
ax = fig.add_subplot(111)
ax.scatter(x, y, c=c)

plt.suptitle('Number of streams and playlists per artist based on season␣
↪domination', fontsize = 14, weight = 'bold')
plt.title('Season domination = in which season the artist is listened the␣
↪most', fontsize = 13, pad=30)

ax.set_xlabel('Number of playlists', fontsize = 12)
ax.set_ylabel('Number of streams', fontsize = 12)

#limit axis to get rid of outliers and better analyse arists who appear in␣
↪fewer playlists
#ax.set_ylim(0, 10000)
#ax.set_xlim(0, 150)

ax.xaxis.grid(color='lightgray', linestyle='--', linewidth=1);
ax.yaxis.grid(color='lightgray', linestyle='--', linewidth=1);

scatter = plt.scatter(x,y,c = c, cmap =colours)
plt.legend(handles = scatter.legend_elements()[0], labels = classes,␣
↪shadow=True, title='Legend', borderpad = 1)

plt.show();
```

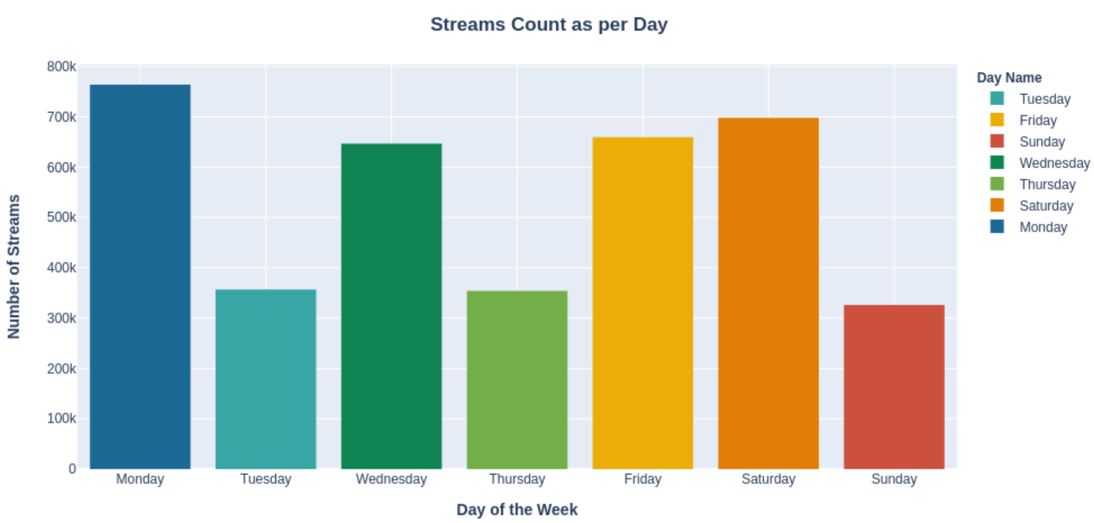


[165]:
```
# df
```

1.4.18 Day of the week

[166]:
```
#Day of the week streams in order
fig = px.histogram(df, x="weekday_name", color='weekday_name',
                   labels={'weekday_name': "<b>Day Name</b>"},
                   color_discrete_map = {'Monday':px.colors.qualitative.
 ↪Prism[1],
                                         'Tuesday':px.colors.qualitative.
 ↪Prism[2],
                                         'Wednesday':px.colors.qualitative.
 ↪Prism[3],
                                         'Thursday':px.colors.qualitative.
 ↪Prism[4],
                                         'Friday':px.colors.qualitative.
 ↪Prism[5],
                                         'Saturday':px.colors.qualitative.
 ↪Prism[6],
```

```
                                              'Sunday':px.colors.qualitative.
 ↪Prism[7]}
                        )
fig.update_xaxes(type='category',ticktext=['Monday','Tuesday','Wednesday',
                                          'Thursday','Friday','Saturday',
                                          'Sunday'],
                 tickvals=["0", "1",'2','3','4','5','6'], showgrid=True)
fig.update_layout(title={'text': '<b>Streams Count as per Day</b>','x':0.5},
                  yaxis_title_text='<b>Number of Streams</b>',
                 xaxis_title_text='<b>Day of the Week</b>',
                  xaxis={'categoryorder':'array', 'categoryarray':
 ↪['Monday','Tuesday','Wednesday',
                                          'Thursday','Friday','Saturday',
                                          'Sunday']} )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```

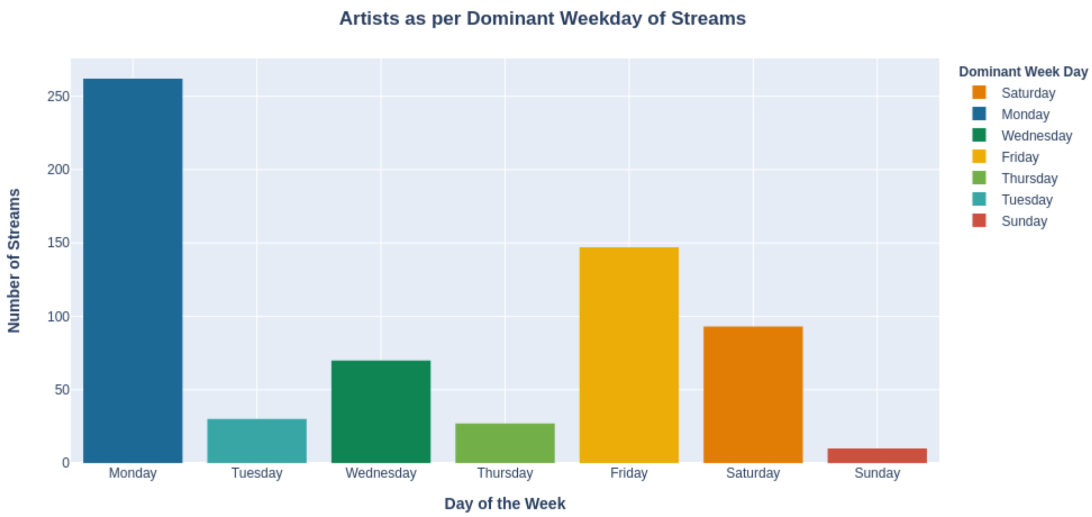


[167]:
```
#group by artist and season and use max function to identify which season␣
 ↪dominates for the particular artist
a = df.groupby(['artist_name', 'weekday_name'])['Unnamed: 0'].count().
 ↪sort_values().groupby(level=0).tail(1).index
df["weekday_domination"] = ''

#assign value of which season dominates for a particular artist back to the␣
 ↪dataframe
weekday_type = {}
for i in range(len(df.groupby(['artist_name', 'weekday_name'])['Unnamed: 0'].
 ↪count().sort_values().groupby(level=0).tail(1).index)):
    weekday_type[a[i][0]] = a[i][1]
```

```
#assign value back to dataframe using apply lambda
df["weekday_domination"] = df["artist_name"].apply(lambda x: weekday_type.
↪get(x))
```

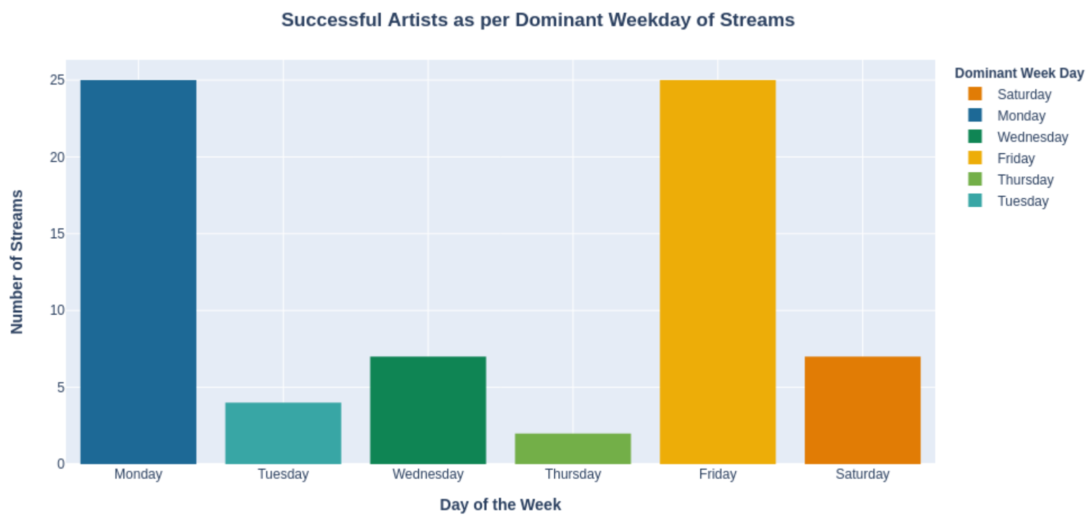
[168]:
```
##RUN ONLY ONCE!

#Weekday domination added to artists_new df
weekday_dom = df.drop_duplicates(['artist_name'],keep = 'last')
artists_new = artists_new.
↪merge(weekday_dom[['artist_name','weekday_domination']], on='artist_name')
```

[169]:
```
#Dominating season as per artists
fig = px.histogram(artists_new, x="weekday_domination",␣
↪color='weekday_domination',
                   labels={'weekday_domination': "<b>Dominant Week Day</b>"},
                                color_discrete_map = {'Monday':px.colors.
↪qualitative.Prism[1],
                                              'Tuesday':px.colors.qualitative.
↪Prism[2],
                                              'Wednesday':px.colors.qualitative.
↪Prism[3],
                                              'Thursday':px.colors.qualitative.
↪Prism[4],
                                              'Friday':px.colors.qualitative.
↪Prism[5],
                                              'Saturday':px.colors.qualitative.
↪Prism[6],
                                           'Sunday':px.colors.qualitative.
↪Prism[7]}
                  )
fig.update_xaxes(type='category',ticktext=['Monday','Tuesday','Wednesday',
                                           'Thursday','Friday','Saturday',
                                           'Sunday'],
                 tickvals=["0", "1",'2','3','4','5','6'], showgrid=True)
fig.update_layout(title={'text': '<b>Artists as per Dominant Weekday of␣
↪Streams</b>','x':0.5},
                  yaxis_title_text='<b>Number of Streams</b>',
                 xaxis_title_text='<b>Day of the Week</b>',
                  xaxis={'categoryorder':'array', 'categoryarray':
↪['Monday','Tuesday','Wednesday',
                                           'Thursday','Friday','Saturday',
                                           'Sunday']} )
```



[170]:
```
#Dominating season as per artists (SUCCESSFUL ONLY)
fig = px.histogram(artists_new[artists_new['success']==1],␣
↪x="weekday_domination", color='weekday_domination',
                   labels={'weekday_domination': "<b>Dominant Week Day</b>"},
                            color_discrete_map = {'Monday':px.colors.
↪qualitative.Prism[1],
                                                  'Tuesday':px.colors.qualitative.
↪Prism[2],
                                                  'Wednesday':px.colors.qualitative.
↪Prism[3],
                                                  'Thursday':px.colors.qualitative.
↪Prism[4],
                                                  'Friday':px.colors.qualitative.
↪Prism[5],
                                                  'Saturday':px.colors.qualitative.
↪Prism[6],
                                               'Sunday':px.colors.qualitative.
↪Prism[7]}
                  )
fig.update_xaxes(type='category',ticktext=['Monday','Tuesday','Wednesday',
                                           'Thursday','Friday','Saturday',
                                           'Sunday'],
                 tickvals=["0", "1",'2','3','4','5','6'], showgrid=True)
fig.update_layout(title={'text': '<b>Successful Artists as per Dominant Weekday␣
↪of Streams</b>','x':0.5},
                  yaxis_title_text='<b>Number of Streams</b>',
                 xaxis_title_text='<b>Day of the Week</b>',
```

```
                      xaxis={'categoryorder':'array', 'categoryarray':
↪['Monday','Tuesday','Wednesday',
                                              'Thursday','Friday','Saturday',
                                              'Sunday']} )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```



1.4.19 Time of Day

[171]:
```
#checking hours
df['hour'].unique()
```

[171]:
```
array([12, 14, 10,  2,  9, 19, 15,  7, 17, 11, 18,  4, 16,  8,  6, 23, 21,
       13, 20,  5,  0,  3, 22,  1])
```

[172]:
```
#Specifying conditions and creating dayphase
dayphase_conditions = [(df['hour'].isin([5,6,7,8,9,10,11])),
                    (df['hour'].isin([12,13,14,15,16,17])),
                    (df['hour'].isin([18,19,20,21])),
                    (df['hour'].isin([22,23,0,1,2,3,4]))]

dayphase_values = ['morning','afternoon','evening','night']

df['dayphase'] = np.select(dayphase_conditions, dayphase_values)
```

[173]:
```
df['dayphase'].unique()
```

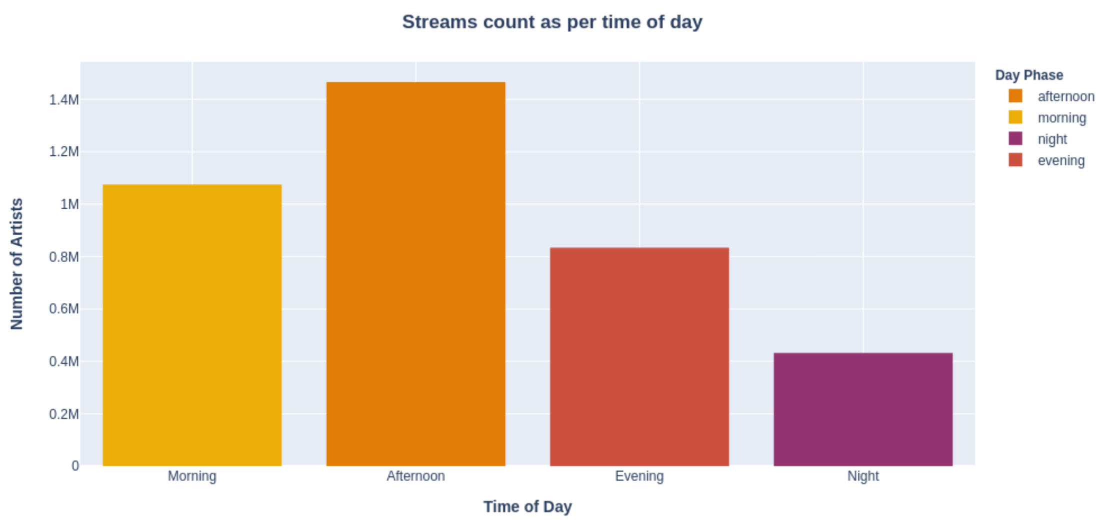
[173]:
```
array(['afternoon', 'morning', 'night', 'evening'], dtype=object)
```

[174]:
```
#Counting dayphase
dayphase = df.groupby(['dayphase'])['log_time'].agg(['count'])
dayphase
```

[174]:

	count
dayphase	
afternoon	1465562
evening	833365
morning	1074758
night	431814

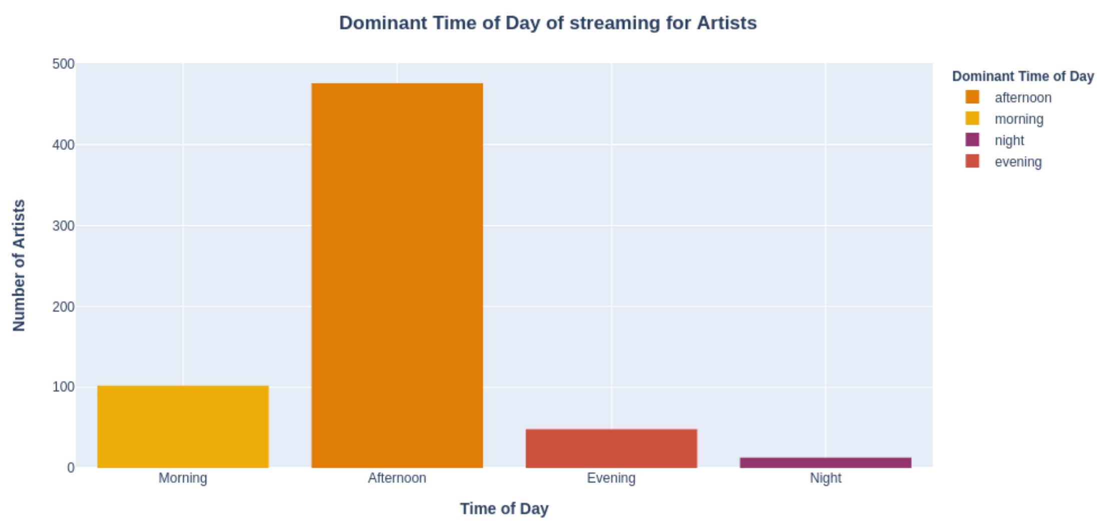
[175]:
```
#Day of the week streams in order
fig = px.histogram(df, x="dayphase", color='dayphase',
                   labels={'dayphase': "<b>Day Phase</b>"},
                  color_discrete_map={"morning": px.colors.qualitative.Prism[5],
                                      'afternoon': px.colors.qualitative.
 ↪Prism[6],
                                      'evening': px.colors.qualitative.Prism[7],
                                      'night':px.colors.qualitative.Prism[8]})
fig.
 ↪update_xaxes(type='category',ticktext=['Morning','Afternoon','Evening','Night'],
                 tickvals=["0", "1",'2','3'], showgrid=True)
fig.update_layout(title={'text': '<b>Streams count as per time of day</b>','x':
 ↪0.5},
                  yaxis_title_text='<b>Number of Artists</b>',
                 xaxis_title_text='<b>Time of Day</b>',
                  xaxis={'categoryorder':'array',
                         'categoryarray':
 ↪['morning','afternoon','evening','night']} )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```



[176]:
```python
#Creating dominant dayphase variable for each artist
artists_dayphase_count = df.groupby(['artist_name', 'dayphase'])['log_time'].
 ↪agg(['count']) #Counting different dayphases per artists
artists_dayphase_count.reset_index(inplace=True) #resetting index
artists_dayphase_count = artists_dayphase_count.pivot_table(values='count',␣
 ↪index=artists_dayphase_count['artist_name'], columns='dayphase',␣
 ↪aggfunc='first') #making a pivot table to get the different dayphases as␣
 ↪columns
artists_dayphase_count = artists_dayphase_count.fillna(0) #Filling nans with 0
artists_dayphase_count['dominant_dayphase'] = artists_dayphase_count.
 ↪idxmax(axis=1) #creating a dominant dayphase column with the maximum value␣
 ↪for each of the dayphase columns
artists_dayphase_count.reset_index( inplace=True) #resetting index
artists_dayphase_count = artists_dayphase_count[['artist_name',␣
 ↪'dominant_dayphase']].copy() #taking the only values we need
```

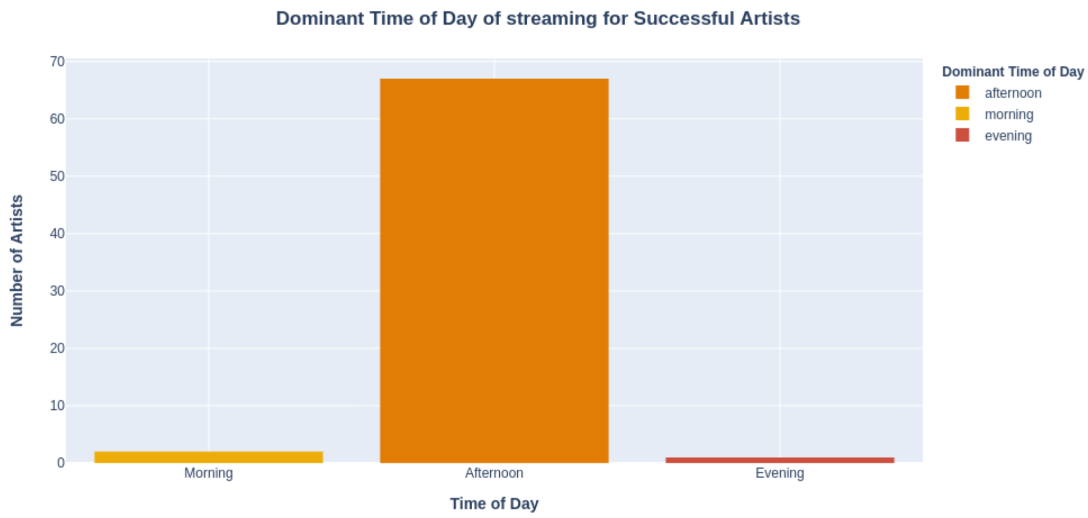
[177]:
```python
# adding to artists_new df
artists_new = artists_new.
 ↪merge(artists_dayphase_count[['artist_name','dominant_dayphase']],␣
 ↪on='artist_name')
```

[178]:
```python
#Dominant day phase for each artist - Visual
fig = px.histogram(artists_new, x="dominant_dayphase",␣
 ↪color='dominant_dayphase',
                   labels={'dominant_dayphase': "<b>Dominant Time of Day</b>"},
                               color_discrete_map={"morning": px.colors.
 ↪qualitative.Prism[5],
                                      'afternoon': px.colors.qualitative.
 ↪Prism[6],
                                      'evening': px.colors.qualitative.Prism[7],
                                      'night':px.colors.qualitative.Prism[8] })
fig.
 ↪update_xaxes(type='category',ticktext=['Morning','Afternoon','Evening','Night'],
                 tickvals=["0", "1",'2','3'], showgrid=True)
fig.update_layout(title={'text': '<b>Dominant Time of Day of streaming for␣
 ↪Artists</b>','x':0.5},
                  yaxis_title_text='<b>Number of Artists</b>',
                 xaxis_title_text='<b>Time of Day</b>',
                  xaxis={'categoryorder':'array', 'categoryarray':
 ↪['morning','afternoon','evening','night']} )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```



[179]:

```
#Dominant day phase for each artist - Visual
fig = px.histogram(artists_new[artists_new['success']==1],␣
↪x="dominant_dayphase", color='dominant_dayphase',
                   labels={'dominant_dayphase': "<b>Dominant Time of Day</b>"},
                            color_discrete_map={"morning": px.colors.
↪qualitative.Prism[5],
                                   'afternoon': px.colors.qualitative.
↪Prism[6],
                                   'evening': px.colors.qualitative.Prism[7],
                                   'night':px.colors.qualitative.Prism[8] })
fig.
↪update_xaxes(type='category',ticktext=['Morning','Afternoon','Evening','Night'],
                 tickvals=["0", "1",'2','3'], showgrid=True)
fig.update_layout(title={'text': '<b>Dominant Time of Day of streaming for␣
↪Successful Artists</b>','x':0.5},
                  yaxis_title_text='<b>Number of Artists</b>',
                 xaxis_title_text='<b>Time of Day</b>',
                  xaxis={'categoryorder':'array', 'categoryarray':
↪['morning','afternoon','evening','night']} )
fig.show() #CHANGE TO FIG.SHOW() FOR USING ON FACULTY
```



1.4.20 Regions

1.5 Successful artists

[180]:
```
#Number of successful Artists per each of the top 4 playlists
df['artist_name']=df['artist_name'].astype(str).str.lower()
artist_num_per_playlist = pd.DataFrame(df.
↪groupby(['playlist_id','playlist_name'])['artist_name'].nunique())
artist_num_per_playlist.reset_index(inplace=True)
artist_num_per_playlist=artist_num_per_playlist.rename(columns= {'playlist_id':
↪'playlist_id','artist_name':'artists'})
artist_num_per_playlist=artist_num_per_playlist.sort_values(by='artists',␣
↪ascending = False)
artists_in_top4=artist_num_per_playlist.
↪loc[(artist_num_per_playlist['playlist_id'].isin(top4['playlist_id']))]
artists_in_top4
```

[180]:

	playlist_id	playlist_name	artists
6074	6FfOZSAN3N6u7v81uS7mxZ	Hot Hits UK	41
3002	37i9dQZF1DX4JAvHpjipBk	New Music Friday	32
3046	37i9dQZF1DX5uokaTN4FTR	Massive Dance Hits	18
2777	37i9dQZF1DWVTKDs2aOkxu	The Indie List	11

[181]:
```
#Check that the below dataframe filter selects the correct number of artists

all_artists_filter[(all_artists_filter['playlist_name']=='Hot Hits␣
↪UK')]['artist_name'].astype(str).str.lower().nunique()
```

```
all_artists_filter[(all_artists_filter['playlist_name']=='New Music␣
↪Friday')]['artist_name'].astype(str).str.lower().nunique()
all_artists_filter[(all_artists_filter['playlist_name']=='Massive Dance␣
↪Hits')]['artist_name'].astype(str).str.lower().nunique()
all_artists_filter[(all_artists_filter['playlist_name']=='The Indie␣
↪List')]['artist_name'].astype(str).str.lower().nunique()

all_artists_filter[(all_artists_filter['playlist_name'].
↪isin(playlist_values))]['artist_name'].astype(str).str.lower().nunique()
#there are 70 unique artists in total in all 4 playlists

artists_in_top4 #aggregated number of artists in the top 4 playlists
```

[181]:

	playlist_id	playlist_name	artists
6074	6FfOZSAN3N6u7v81uS7mxZ	Hot Hits UK	41
3002	37i9dQZF1DX4JAvHpjipBk	New Music Friday	32
3046	37i9dQZF1DX5uokaTN4FTR	Massive Dance Hits	18
2777	37i9dQZF1DWVTKDs2aOkxu	The Indie List	11

[182]:
```
#get artist list for all the playlists
hhu_artists=df[((df['playlist_id'] == '6FfOZSAN3N6u7v81uS7mxZ') &␣
↪(df['playlist_name'] =='Hot Hits UK' ))]['artist_name'].unique().tolist()
mdh_artists=df[((df['playlist_id'] == '37i9dQZF1DX5uokaTN4FTR') &␣
↪(df['playlist_name'] == 'Massive Dance Hits'))]['artist_name'].unique().
↪tolist()
indie_list_artists=df[((df['playlist_id'] == '37i9dQZF1DWVTKDs2aOkxu') &␣
↪(df['playlist_name'] == 'The Indie List'))]['artist_name'].unique().tolist()
nmf_artists=df[((df['playlist_id'] == '37i9dQZF1DX4JAvHpjipBk') &␣
↪(df['playlist_name'] == 'New Music Friday'))]['artist_name'].unique().
↪tolist()
all_arts=hhu_artists+mdh_artists+indie_list_artists+nmf_artists
len(set(all_arts))
```

[182]: 70

[184]:
```
import datetime
import itertools
```

[185]:
```
#Getting the dataframe for the successful artists, and adding their journey in␣
↪terms of nr of playlists they've been in prior to being in the "succesful"␣
↪playlists
#also with nr playlsits they've been in before t-30 of being in successful␣
↪playlist

#will require this function for flattening the list of playlist_value_artists
flatten = itertools.chain.from_iterable
```

```
#make the list of lists from the artists belonging to each playlist
playlist_values_artists=list(set(list(flatten([hhu_artists,mdh_artists,indie_list_artists,nmf_

list_prior_nr_t120=[]
list_prior_nr_t60=[]
list_prior_nr_t30=[]
list_prior_nr=[]

for artist in playlist_values_artists: #for i in 4
    #for every artist name in playlist
    date_first_success=df[(df['playlist_id'].isin(top4['playlist_id'])) &␣
↪(df['artist_name']==artist)]['date']
    if np.size(date_first_success)!=0:
        date_first_success=min(date_first_success.values)
        nr_prior_playlists=df[(df['artist_name']==artist)&␣
↪(df['date']<date_first_success)]['playlist_name'].count()
        list_prior_nr.append(nr_prior_playlists)
        date_first_success=pd.to_datetime(date_first_success, '%Y-%m-%d')
        date_first_success_t120=date_first_success-datetime.timedelta(days=120)
        nr_prior_playlists_t120=df[(df['artist_name']==artist)& (pd.
↪to_datetime(df['date'])<date_first_success_t120)]['playlist_name'].count()
        list_prior_nr_t120.append(nr_prior_playlists_t120)
        date_first_success_t60=date_first_success-datetime.timedelta(days=60)
        nr_prior_playlists_t60=df[(df['artist_name']==artist)& (pd.
↪to_datetime(df['date'])<date_first_success_t60)]['playlist_name'].count()
        list_prior_nr_t60.append(nr_prior_playlists_t60)
        date_first_success_t30=date_first_success-datetime.timedelta(days=30)
        nr_prior_playlists_t30=df[(df['artist_name']==artist)& (pd.
↪to_datetime(df['date'])<date_first_success_t30)]['playlist_name'].count()
        list_prior_nr_t30.append(nr_prior_playlists_t30)
    else:
        print(artist) #this will show which arist from which playlist currently␣
↪doesn't have a first_date
        list_prior_nr_t120.append(0)
        list_prior_nr_t60.append(0)
        list_prior_nr_t30.append(0)
        list_prior_nr.append(0)

        break

prior_playlists =␣
↪list(zip(playlist_values_artists,list_prior_nr_t120,list_prior_nr_t60,␣
↪list_prior_nr_t30, list_prior_nr ))
```

```
playlists_priors=pd.DataFrame(prior_playlists,␣
↪columns=['artist_name','nr_prior_playlists_t120','nr_prior_playlists_t60','nr_prior_playlis
playlists_priors
```

[185]:

```
           artist_name  nr_prior_playlists_t120  nr_prior_playlists_t60  \
0          starlovers                        0                       0
1         xavier dunn                        0                       0
2      sage the gemini                       0                      13
3         matt maeson                        0                      18
4           coldabank                        0                       0
..                 …                       …                     …
65   catherine mcgrath                       0                       0
66                dave                       0                       0
67                vice                      16                      33
68          anne-marie                     557                     936
69          all tvvins                     969                    1132

    nr_prior_playlists_t30  nr_prior_playlists
0                        0                   0
1                        0                   0
2                       13                  38
3                       53                  87
4                        0                   0
..                     …                   …
65                       0                   0
66                       0                   0
67                      33                1326
68                    1668                1668
69                    1198                1198

[70 rows x 5 columns]
```

[186]:

```
#stream_count alternative method
#make the list of lists from the artists belonging to each playlist

#will require this function for flattening the list of playlist_value_artists
import itertools
flatten = itertools.chain.from_iterable

playlist_values_artists=list(set(list(flatten([hhu_artists,mdh_artists,indie_list_artists,nmf_

stream_count_t120=[]
stream_count_t60=[]
stream_count_t30=[]
stream_count_present=[]

for artist in playlist_values_artists: #for i in 4
```

```
    #for every artist name in playlist
    date_first_success=df[(df['playlist_id'].isin(top4['playlist_id'])) &␣
↪(df['artist_name']==artist)]['date']
    if np.size(date_first_success)!=0:
        date_first_success=min(date_first_success.values)
        nr_prior_streams=df[(df['artist_name']==artist)&␣
↪(df['date']<date_first_success)]['day'].count()
        stream_count_present.append(nr_prior_streams)
        date_first_success=pd.to_datetime(date_first_success, '%Y-%m-%d')
        date_first_success_t120=date_first_success-datetime.timedelta(days=120)
        nr_prior_streams_120=df[(df['artist_name']==artist)& (pd.
↪to_datetime(df['date'])<date_first_success_t120)]['day'].count()
        stream_count_t120.append(nr_prior_streams_120)
        date_first_success_t60=date_first_success-datetime.timedelta(days=60)
        nr_prior_streams_60=df[(df['artist_name']==artist)& (pd.
↪to_datetime(df['date'])<date_first_success_t60)]['day'].count()
        stream_count_t60.append(nr_prior_streams_60)
        date_first_success_t30=date_first_success-datetime.timedelta(days=30)
        nr_prior_streams_t30=df[(df['artist_name']==artist)& (pd.
↪to_datetime(df['date'])<date_first_success_t30)]['day'].count()
        stream_count_t30.append(nr_prior_streams_t30)
    else:
        print(artist) #this will show which arist from which playlist currently␣
↪doesn't have a first_date
        stream_count_present.append(0)
        stream_count_t120.append(0)
        stream_count_t60.append(0)
        stream_count_t30.append(0)
        break

prior_streams = list(zip(playlist_values_artists,stream_count_t120,␣
↪stream_count_t60, stream_count_t30,stream_count_present ))

streams_priors=pd.DataFrame(prior_streams,␣
↪columns=['artist_name','stream_count_t120','stream_count_t60','stream_count_t30','stream_co
streams_priors
```

[186]:

```
           artist_name  stream_count_t120  stream_count_t60  stream_count_t30  \
0           starlovers                  0                 0                 0
1          xavier dunn                  0                 0                 0
2      sage the gemini                  0                21                21
3          matt maeson                  0                36               100
4            coldabank                  0                 0                 0
..                 …                  …                 …                 …
65   catherine mcgrath                  0                 0                 0
66                dave                  4                 4                 5
```

```
67            vice          57      185     185
68      anne-marie         919     1623    3108
69      all tvvins        3798     4410    4732

    stream_count_present
0                      0
1                      0
2                     75
3                    175
4                      0
..                   …
65                     0
66                     5
67                  1815
68                  3108
69                  4732

[70 rows x 5 columns]
```

[187]:
```
# streams_priors_index=streams_priors.set_index('artist_name')
```

[188]:
```
# streams_priors_index
```

[189]:
```
# streams_priors_index.plot(kind='barh')
# plt.title('Top 10 Features')
# plt.grid() #adding grid
# # save_fig('top_feature')
```

[190]:
```
# table = pd.pivot_table(streams_priors_index,index=['artist_name'])
# table=table.reset_index(drop=True)
```

[191]:
```
# table
```

[192]:
```
# fig = px.line(streams_priors, x=streams_priors.columns, y="artist_name",␣
↪color='artist_name')
# fig.show()
```

1.6 Playlist features

[193]:
```
# you could divide up the work in the group by getting different people to␣
↪calculate different features

def playlist_avg_stream_counts(data):
    playlist_streams = data.groupby('artist_name')['playlist_id'].nunique()
    artist_users = data.groupby('artist_name')['artist_name'].agg(['count'])
```

```
    avg_streams = pd.merge(left=artist_users,right=playlist_streams,␣
 ↪left_index=True, right_index=True, how='left')
    avg_streams = avg_streams.rename(columns={'count':'streams','playlist_id':
 ↪'unique_playlist_count'})
    avg_streams['playlist_avg_stream'] = avg_streams['streams']/
 ↪avg_streams['unique_playlist_count']
    return(avg_streams)

#For artists that have been played but not on a playlist, how should we fix it?␣
 ↪playlist_avg_stream==0?
def playlist_avg_number_of_users(data):
    playlist_streams = playlist_avg_stream_counts(all_artists)
    users_per_artist = data.groupby('artist_name')['customer_id'].nunique()
    avg_user = pd.merge(left=users_per_artist,right=playlist_streams,␣
 ↪left_index=True, right_index=True, how='left')
    avg_user = avg_user.rename(columns= {'customer_id':'number_of_users'})
    avg_user['user_per_playlist'] = avg_user['number_of_users']/
 ↪avg_user['unique_playlist_count']
    return (avg_user)

def playlist_avg_passion_score(data):
    artists_playlists_features = playlist_avg_number_of_users(data)
    artists_playlists_features['passion_score_per_playlist'] =␣
 ↪(artists_playlists_features['streams']/
 ↪artists_playlists_features['number_of_users'])/
 ↪artists_playlists_features['unique_playlist_count']
    return(artists_playlists_features)
#take a sample of the data and test the functions to ensure we get the correct␣
 ↪data, 5 -10 artists should suffice

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

[194]:
```
avg_streams = playlist_avg_stream_counts(df)
avg_user = playlist_avg_number_of_users(df)
playlists = playlist_avg_passion_score(df)
```

[195]:
```
# #Gender per artists
# artists_gender_split = df.groupby(['artist_name','gender']).size().
 ↪unstack(fill_value=0)
```

[196]:
```
# #Run once
# artists_new = pd.merge(left= artists_new, right= artists_gender_split,␣
 ↪on='artist_name', how = 'left').copy()
# artists_new = artists_new.rename(columns= {'female':'female_streams', 'male':
 ↪'male_streams', 'count':'streams'})
```

[197]:
```
artists_new
```

[197]:
```
              artist_name   count  number_songs  success  playlists  listeners  \
0             charlie puth  447873            38        1       1747     367023
1                 dua lipa  315663            50        1        892     260778
2             lukas graham  311271            22        1       1211     247580
3              cheat codes  255820            16        1       1218     225658
4              anne-marie  247934            28        1        757     220413
..                    …       …             …        …          …          …
634        rebecka karlsson       1             1        0          0          1
635  los tres paraguayos       1             1        0          0          1
636                 deuspi       1             1        0          1          1
637              vince pope       1             1        0          1          1
638             los romeos       1             1        0          0          1

     passion_score  avg_stream_time  repeat_count gender_domination  \
0         1.220286       185.767816         23424            female
1         1.210466       178.106221         11671            female
2         1.257254       207.311259         27625              male
3         1.133662       184.465644         11889            female
4         1.124861       182.480559          9965            female
..             …              …             …                 …
634       1.000000       189.000000             0              male
635       1.000000       172.000000             0            female
636       1.000000       217.000000             0              male
637       1.000000        83.000000             0              male
638       1.000000       203.000000             0              male

    generation_domination season_domination weekday_domination  \
0             millennials            summer           Saturday
1             millennials            summer             Monday
2             millennials            spring          Wednesday
3             millennials            summer           Saturday
4             millennials            spring             Monday
..                    …                 …                  …
634          generation_x            summer             Monday
635           millennials            summer             Monday
636          generation_x            summer           Saturday
637           millennials            spring          Wednesday
638          generation_x            summer           Saturday

    dominant_dayphase
0           afternoon
1           afternoon
2           afternoon
3           afternoon
4           afternoon
```

```
..                 …
634         afternoon
635           evening
636           morning
637           evening
638         afternoon

[639 rows x 14 columns]
```

[198]:
```python
#Featuring artists
all_artists['featuring_artists'] = all_artists.track_artists.str.count(',')
```

[199]:
```python
all_artists.sort_values(by='featuring_artists', ascending=False)
```

[199]:
```
         Unnamed: 0  Unnamed: 0.1                     Unnamed: 0.1.1  day  \
36481         36481        364819  ('small_artists_2016.csv', 364819)   10
7089           7089         70899   ('small_artists_2016.csv', 70899)   10
157879       157879       1578799  ('small_artists_2017.csv', 818141)   10
157880       157880       1578809  ('small_artists_2017.csv', 818151)   10
7104           7104         71049   ('small_artists_2016.csv', 71049)   10
…               …            …                                   …   …
1216560     1216560      12165609  ('charlie_puth_late.csv', 1455301)   10
1216561     1216561      12165619  ('charlie_puth_late.csv', 1455311)   10
1216562     1216562      12165629  ('charlie_puth_late.csv', 1455321)   10
1216563     1216563      12165639  ('charlie_puth_late.csv', 1455331)   10
3805498     3805498      38054989                             1301591   10

                  log_time  mobile                          track_id  \
36481    20160810T10:45:00   False  4d4198de27e642c7b71d3d29a6e0bc09
7089     20160510T09:30:00    True  4d4198de27e642c7b71d3d29a6e0bc09
157879   20170310T18:00:00    True  4d4198de27e642c7b71d3d29a6e0bc09
157880   20170310T07:45:00    True  4d4198de27e642c7b71d3d29a6e0bc09
7104     20160510T16:45:00    True  4d4198de27e642c7b71d3d29a6e0bc09
…                      …     …                               …
1216560  20170710T16:15:00    True  8d5f3663fc0b4696acdf97a27262cc59
1216561  20170710T07:15:00    True  8d5f3663fc0b4696acdf97a27262cc59
1216562  20170710T12:45:00    True  8d5f3663fc0b4696acdf97a27262cc59
1216563  20170710T05:45:00    True  8d5f3663fc0b4696acdf97a27262cc59
3805498  20170710T12:00:00    True  4cb959db5be04d2fa5ca4c137b651a99

                isrc           upc   artist_name  \
36481    USAT21600962  7.567991e+10  Vinyl on HBO
7089     USAT21600962  7.567991e+10  Vinyl on HBO
157879   USAT21600962  7.567991e+10  Vinyl on HBO
157880   USAT21600962  7.567991e+10  Vinyl on HBO
7104     USAT21600962  7.567991e+10  Vinyl on HBO
…                …            …            …
```

```
1216560  USAT21700928  7.567990e+10  Charlie Puth
1216561  USAT21700928  7.567990e+10  Charlie Puth
1216562  USAT21700928  7.567990e+10  Charlie Puth
1216563  USAT21700928  7.567990e+10  Charlie Puth
3805498  GBAHS1600395  1.902959e+11    Anne-Marie

                                  track_name  \
36481    Kill The Lights (with Nile Rodgers)
7089     Kill The Lights (with Nile Rodgers)
157879   Kill The Lights (with Nile Rodgers)
157880   Kill The Lights (with Nile Rodgers)
7104     Kill The Lights (with Nile Rodgers)
…                                        …
1216560                            Attention
1216561                            Attention
1216562                            Attention
1216563                            Attention
3805498                 Alarm - Cahill Remix

                                      album_name  \
36481    VINYL: THE ESSENTIALS: BEST OF SEASON 1
7089     VINYL: THE ESSENTIALS: BEST OF SEASON 1
157879   VINYL: THE ESSENTIALS: BEST OF SEASON 1
157880   VINYL: THE ESSENTIALS: BEST OF SEASON 1
7104     VINYL: THE ESSENTIALS: BEST OF SEASON 1
…                                            …
1216560                                Attention
1216561                                Attention
1216562                                Attention
1216563                                Attention
3805498                                    Alarm

                              customer_id postal_code   access country_code  \
36481    dc70b9c06f29b101fe9599a194f1b268          No  premium           GB
7089     255beb0d066633523e7d0b916599c1d8          No  premium           GB
157879   b8d7e620a307ff2bd50665044510d2b1          NE  premium           GB
157880   98869e4115a067036fc2f30a8ceb0445           1  premium           GB
7104     70e318556a8142d5b93474931febe336          No     free           GB
…                                     …           …       …            …
1216560  c1fc2b02499d5f7d3c7e0502f6143080         NaN  premium           GB
1216561  c21498d2ad618a152772863c1389976d         NaN  premium           GB
1216562  c26ee09c0562f5eabf2b731821451e4b         NaN  premium           GB
1216563  0182905135d6088100fb6df2e0da8c36         NaN  premium           GB
3805498  0192986fc253ab12b6609b3189ac809b         NaN  premium           GB

         gender  birth_year                    filename region_code  \
36481      male      1994.0  streams_20160810_GB.009.gz      GB-LND
```

```
7089     female      1969.0  streams_20160510_GB.001.gz      GB-WSX
157879   female      1992.0  streams_20170310_GB.014.gz      GB-NET
157880     male      1978.0  streams_20170310_GB.012.gz      GB-MAN
7104       male      1958.0  streams_20160510_GB.004.gz      GB-TOB
…          …         …                          …         …
1216560    male      1980.0  streams_20170710_GB.011.gz      GB-HRT
1216561  female      1960.0  streams_20170710_GB.011.gz      GB-WFT
1216562  female      1995.0  streams_20170710_GB.011.gz      GB-BIR
1216563  female      1982.0  streams_20170710_GB.000.gz      GB-STY
3805498    male      1992.0  streams_20170710_GB.000.gz         NaN

        referral_code partner_name  … offline_timestamp stream_length  \
36481             NaN          NaN  …               NaN         275.0
7089              NaN          NaN  …               NaN          45.0
157879            NaN          NaN  …               NaN          30.0
157880            NaN         boku  …               NaN         275.0
7104              NaN          NaN  …               NaN         275.0
…                …          …   …               …            …
1216560           NaN          NaN  …               NaN         540.0
1216561           NaN          NaN  …               NaN         211.0
1216562           NaN  vodafone-uk  …               NaN         211.0
1216563           NaN          NaN  …               NaN          50.0
3805498           NaN          NaN  …               NaN          81.0

        stream_cached    stream_source  \
36481             NaN  others_playlist
7089              NaN  others_playlist
157879            NaN       collection
157880            NaN       collection
7104              NaN           artist
…                …              …
1216560           NaN           artist
1216561           NaN            other
1216562           NaN       collection
1216563           NaN            other
3805498           NaN       collection

                                       stream_source_uri stream_device  \
36481                                                NaN       desktop
7089    spotify:user:spotify:playlist:3hojaDtnWmBFMGvn…        mobile
157879                                               NaN        mobile
157880                                               NaN        mobile
7104                                                 NaN        mobile
…                                                 …            …
1216560                                              NaN        mobile
1216561                                              NaN        mobile
1216562                                              NaN        mobile
```

```
1216563                                               NaN         mobile
3805498                                               NaN         mobile

        stream_os                             track_uri  \
36481     Windows  spotify:track:2lMo0dNnc09Nivs537rf0V
7089      Android  spotify:track:2lMo0dNnc09Nivs537rf0V
157879        iOS  spotify:track:2lMo0dNnc09Nivs537rf0V
157880    Android  spotify:track:2lMo0dNnc09Nivs537rf0V
7104      Android  spotify:track:2lMo0dNnc09Nivs537rf0V
…             …                                    …
1216560       iOS  spotify:track:4iLqG9SeJSnt0cSPICSjxv
1216561       iOS  spotify:track:4iLqG9SeJSnt0cSPICSjxv
1216562       iOS  spotify:track:4iLqG9SeJSnt0cSPICSjxv
1216563   Android  spotify:track:4iLqG9SeJSnt0cSPICSjxv
3805498       iOS  spotify:track:2kM7ASijHVSoMlC49EDsFj

                                            track_artists source  \
36481    Jess Glynne, DJ Cassidy, Alex Newell, Vinyl on…    NaN
7089     Jess Glynne, DJ Cassidy, Alex Newell, Vinyl on…    NaN
157879   Jess Glynne, DJ Cassidy, Alex Newell, Vinyl on…    NaN
157880   Jess Glynne, DJ Cassidy, Alex Newell, Vinyl on…    NaN
7104     Jess Glynne, DJ Cassidy, Alex Newell, Vinyl on…    NaN
…                                                     …    …
1216560                                     Charlie Puth    NaN
1216561                                     Charlie Puth    NaN
1216562                                     Charlie Puth    NaN
1216563                                     Charlie Puth    NaN
3805498                                       Anne-Marie    NaN

                   DateTime  hour minute  week  month  year        date  \
36481    2016-08-10 10:45:00    10     45    32      8  2016  2016-08-10
7089     2016-05-10 09:30:00     9     30    19      5  2016  2016-05-10
157879   2017-03-10 18:00:00    18      0    10      3  2017  2017-03-10
157880   2017-03-10 07:45:00     7     45    10      3  2017  2017-03-10
7104     2016-05-10 16:45:00    16     45    19      5  2016  2016-05-10
…                       …    …     …    …    …     …           …
1216560  2017-07-10 16:15:00    16     15    28      7  2017  2017-07-10
1216561  2017-07-10 07:15:00     7     15    28      7  2017  2017-07-10
1216562  2017-07-10 12:45:00    12     45    28      7  2017  2017-07-10
1216563  2017-07-10 05:45:00     5     45    28      7  2017  2017-07-10
3805498  2017-07-10 12:00:00    12      0    28      7  2017  2017-07-10

         weekday weekday_name             playlist_id   playlist_name  \
36481          2    Wednesday                     NaN             NaN
7089           1      Tuesday  3hojaDtnWmBFMGvnMu5Lqj  Pop Right Now!
157879         4       Friday                     NaN             NaN
157880         4       Friday                     NaN             NaN
```

```
7104            1      Tuesday                          NaN              NaN
…          …          …                      …             …
1216560         0       Monday                          NaN              NaN
1216561         0       Monday                          NaN              NaN
1216562         0       Monday                          NaN              NaN
1216563         0       Monday                          NaN              NaN
3805498         0       Monday                          NaN              NaN

        featuring_artists
36481                   3
7089                    3
157879                  3
157880                  3
7104                    3
…                   …
1216560                 0
1216561                 0
1216562                 0
1216563                 0
3805498                 0

[3805499 rows x 46 columns]
```

[200]:
```
artist_features = all_artists.groupby('artist_name', as_index =␣
 ↪False)['featuring_artists'].mean()
artist_features.sort_values(by='featuring_artists', ascending = False)
```

[200]:
```
          artist_name  featuring_artists
457         Profeetat           2.000000
601        Truls Mork           1.384615
622       Vinyl on HBO          1.134413
13              AXSHN           1.000000
654  Zbigniew Kurtycz           1.000000
..                …                 …
230             Irama           0.000000
231             Irina           0.000000
232    Isabell Otrebus          0.000000
233   Ita Purnamasari           0.000000
660         livetune+           0.000000

[661 rows x 2 columns]
```

[201]:
```
artist_features['artist_name']=artist_features['artist_name'].astype(str).str.
 ↪lower()
```

[202]:
```
#Run once
```

```
artists_new = pd.merge(left=artists_new, right = artist_features, on␣
↪='artist_name', how='left').copy()
```

[203]:
```
artists_new
```

[203]:
```
                artist_name   count  number_songs  success  playlists  listeners  \
0              charlie puth  447873            38        1       1747     367023
1                  dua lipa  315663            50        1        892     260778
2              lukas graham  311271            22        1       1211     247580
3               cheat codes  255820            16        1       1218     225658
4                anne-marie  247934            28        1        757     220413
..                      …       …           …        …         …          …
656        rebecka karlsson       1             1        0          0          1
657     los tres paraguayos       1             1        0          0          1
658                  deuspi       1             1        0          1          1
659              vince pope       1             1        0          1          1
660              los romeos       1             1        0          0          1

     passion_score  avg_stream_time  repeat_count gender_domination  \
0         1.220286       185.767816         23424            female
1         1.210466       178.106221         11671            female
2         1.257254       207.311259         27625              male
3         1.133662       184.465644         11889            female
4         1.124861       182.480559          9965            female
..             …               …             …                …
656       1.000000       189.000000             0              male
657       1.000000       172.000000             0            female
658       1.000000       217.000000             0              male
659       1.000000        83.000000             0              male
660       1.000000       203.000000             0              male

    generation_domination season_domination weekday_domination  \
0             millennials            summer           Saturday
1             millennials            summer             Monday
2             millennials            spring          Wednesday
3             millennials            summer           Saturday
4             millennials            spring             Monday
..                    …                 …                 …
656          generation_x            summer             Monday
657           millennials            summer             Monday
658          generation_x            summer           Saturday
659           millennials            spring          Wednesday
660          generation_x            summer           Saturday

    dominant_dayphase  featuring_artists
0           afternoon           0.001480
1           afternoon           0.002781
```

```
2          afternoon          0.001291
3          afternoon          0.728837
4          afternoon          0.000000
..               …                 …
656        afternoon          0.000000
657          evening          0.000000
658          morning          0.000000
659          evening          0.000000
660        afternoon          0.000000

[661 rows x 15 columns]
```

[204]:
```
artists_new.drop(['artist_name'], axis =1, inplace = True)
```

[205]:
```
artists_new = artists_new.rename(columns= {'count':'number_of_streams',␣
↪'listeners':'unique_listeners'})
```

[206]:
```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(artists_new, test_size=0.2,␣
↪random_state=42)
```

[207]:
```
#No idea why this doesn't work
train_set['success'].value_counts()
```

[207]:
```
0    473
1     55
Name: success, dtype: int64
```

[208]:
```
#No idea why this doesn't work
test_set['success'].value_counts()
```

[208]:
```
0    113
1     20
Name: success, dtype: int64
```

[209]:
```
artists_model = train_set.drop('success', axis=1)
artists_labels = train_set['success'].copy()
```

[210]:
```
artists_model
```

[210]:
```
     number_of_streams  number_songs  playlists  unique_listeners  \
533                  4             3          0                 3
552                  3             3          0                 3
613                  1             1          0                 1
61                5681             7         95              5223
430                 16             1          2                15
..                 …             …         …                 …
```

```
71            4495             22         29              3773
106           1735             18         39              1508
270             90              5         10                85
435             15              5          4                15
102           1916             34         79              1607

     passion_score  avg_stream_time  repeat_count gender_domination  \
533       1.333333       263.750000             0            female
552       1.000000       353.666667             0              male
613       1.000000        81.000000             0              male
61        1.087689       186.146277           270            female
430       1.066667       101.625000             1              male
..             …              …             …               …
71        1.191360       184.572191            93            female
106       1.150531       222.559078            21            female
270       1.058824       222.433333             5              male
435       1.000000       186.266667             0            female
102       1.192284       179.595511           172              male

    generation_domination season_domination weekday_domination  \
533           millennials            summer             Monday
552           millennials            summer          Wednesday
613          generation_z            spring             Friday
61            millennials            summer          Wednesday
430           millennials            summer             Monday
..                    …                 …                  …
71            millennials            spring             Monday
106           millennials            spring          Wednesday
270           millennials            spring             Monday
435           millennials            summer             Monday
102           millennials            summer             Friday

    dominant_dayphase  featuring_artists
533           morning           0.000000
552         afternoon           0.000000
613           morning           0.000000
61          afternoon           0.000000
430           morning           0.000000
..                …                 …
71          afternoon           0.000000
106         afternoon           0.108357
270         afternoon           0.000000
435           morning           0.000000
102         afternoon           0.000000

[528 rows x 13 columns]
```

[211]:
```
#Replacing all infinity values with NaN
artists_model= artists_model.replace([np.inf, -np.inf], np.nan)
#REplacing none with unknown_gender
artists_model['gender_domination'] = artists_model['gender_domination'].
↪replace(np.NaN,'unknown gender')
```

[212]:
```
artists_cat = artists_model.select_dtypes(include=['object']) #creating␣
↪dataframe with only catagorical values
artists_num = artists_model.select_dtypes(include=['float', 'int']) #creating␣
↪dataframe with numerical values
```

[213]:
```
artists_cat['gender_domination'] = artists_cat['gender_domination'].replace(np.
↪NaN,'Unknown')
```

```
<ipython-input-213-d3dca15802fe>:1: 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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[214]:
```
#Transformation categorical
from sklearn.impute import SimpleImputer
cat_imp = SimpleImputer(strategy='most_frequent')
cat_artists_imp = cat_imp.fit_transform(artists_cat)
cat_artists_imp
```

[214]:
```
array([['female', 'millennials', 'summer', 'Monday', 'morning'],
       ['male', 'millennials', 'summer', 'Wednesday', 'afternoon'],
       ['male', 'generation_z', 'spring', 'Friday', 'morning'],
       …,
       ['male', 'millennials', 'spring', 'Monday', 'afternoon'],
       ['female', 'millennials', 'summer', 'Monday', 'morning'],
       ['male', 'millennials', 'summer', 'Friday', 'afternoon']],
      dtype=object)
```

[215]:
```
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
artists_cat_one_hot = cat_encoder.fit_transform(cat_artists_imp)
artists_cat_one_hot
```

[215]:
```
<528x22 sparse matrix of type '<class 'numpy.float64'>'
        with 2640 stored elements in Compressed Sparse Row format>
```

[216]:
```
from sklearn.base import BaseEstimator, TransformerMixin

class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names=attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

[217]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

num_attr = list(artists_num) #Creating a list of numerical attributes
cat_attr = list(artists_cat) #Creating a list of categorical attributes

num_pipeline=Pipeline([ #Assigning numerical pipeline
    ('selector', DataFrameSelector(num_attr)), #Selecting the numerical␣
↪attributes
    ('num_imp', SimpleImputer(fill_value = 0, strategy='constant')), #Adding␣
↪the simple imputer to replace missing numerical variables with 0
    ('std_scaler',StandardScaler()), #StandardScaler
])

cat_pipeline = Pipeline([ #Assigning categorical pipeline
    ('selector', DataFrameSelector(cat_attr)), #Selecting the categorical␣
↪variables
    ('cat_imp', SimpleImputer(strategy='most_frequent')), #Adding the imputer␣
↪to replace missing categorical variables with most frequent
    ('one_hot', OneHotEncoder()), #One hot encoding the categorical variables
])

full_pipeline = ColumnTransformer([ #Putting the two pipelines together in a␣
↪final pipeline
    ('num', num_pipeline,num_attr),
    ('cat_pipe', cat_pipeline,cat_attr)
])
```

[218]:
```
artists_prepared = full_pipeline.fit_transform(artists_model)
artists_prepared
```

[218]:
```
array([[-0.18305651, -0.40849957, -0.2593627 , …,  0.        ,
         1.        ,  0.        ],
       [-0.18308623, -0.40849957, -0.2593627 , …,  0.        ,
         0.        ,  0.        ],
       [-0.18314566, -0.66254877, -0.2593627 , …,  0.        ,
         1.        ,  0.        ],
       …,
       [-0.18050066, -0.15445037, -0.18864933, …,  0.        ,
         0.        ,  0.        ],
       [-0.1827296 , -0.15445037, -0.23107735, …,  0.        ,
         1.        ,  0.        ],
       [-0.12623355,  3.52926308,  0.29927289, …,  0.        ,
         0.        ,  0.        ]])
```

[219]:
```
artists_prepared.shape
```

[219]:
```
(528, 30)
```

[ ]:
```

```

[220]:
```
#Getting all the feature names
labels = np.concatenate(cat_encoder.categories_).ravel().tolist()
cat_one_hot_attribs = (labels)
attributes = num_attr + cat_one_hot_attribs
```

[221]:
```
# Check for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif['feature'] = (attributes)
vif['VIF'] = [variance_inflation_factor(artists_prepared,i)
for i in range(len(attributes))]
```

```
/opt/anaconda/envs/Python3/lib/python3.8/site-
packages/statsmodels/stats/outliers_influence.py:193: RuntimeWarning:

divide by zero encountered in double_scalars
```

[222]:
```
vif
```

[222]:

	feature	VIF
0	number_of_streams	348.007753
1	number_songs	1.705752
2	playlists	11.952170
3	unique_listeners	300.221745
4	passion_score	1.094271
5	avg_stream_time	1.080527

```
6          repeat_count    6.981500
7     featuring_artists    1.069893
8                female         inf
9                  male         inf
10              unknown         inf
11               boomer         inf
12         generation_x         inf
13         generation_z         inf
14          millennials         inf
15               autumn         inf
16               spring         inf
17               summer         inf
18               winter         inf
19               Friday         inf
20               Monday         inf
21             Saturday         inf
22               Sunday         inf
23             Thursday         inf
24              Tuesday         inf
25            Wednesday         inf
26            afternoon         inf
27              evening         inf
28              morning         inf
29                night         inf
```

[223]:
```
artists_new_df = pd.DataFrame (artists_prepared, columns = attributes)
```

[224]:
```
artists_vif_test = artists_new_df.
 ↪drop(['number_of_streams','unique_listeners'],axis=1)
```

[225]:
```
artists_vif_test
```

[225]:
```
     number_songs  playlists  passion_score  avg_stream_time  repeat_count  \
0       -0.408500  -0.259363       0.062491         1.258126     -0.164997
1       -0.408500  -0.259363      -0.123886         2.805989     -0.164997
2       -0.662549  -0.259363      -0.123886        -1.887808     -0.164997
3        0.099599   0.412414      -0.074857        -0.077777     -0.017140
4       -0.662549  -0.245220      -0.086611        -1.532761     -0.164450
..            …          …              …                …            …
523      2.004968  -0.054294      -0.016891        -0.104874     -0.114069
524      1.496869   0.016419      -0.039720         0.549048     -0.153497
525     -0.154450  -0.188649      -0.090996         0.546884     -0.162259
526     -0.154450  -0.231077      -0.123886        -0.075704     -0.164997
527      3.529263   0.299273      -0.016374        -0.190544     -0.070807

     featuring_artists  female  male  unknown  boomer  generation_x  \
0            -0.182578     1.0   0.0      0.0     0.0           0.0
```

```
1            -0.182578     0.0   1.0      0.0     0.0            0.0
2            -0.182578     0.0   1.0      0.0     0.0            0.0
3            -0.182578     1.0   0.0      0.0     0.0            0.0
4            -0.182578     0.0   1.0      0.0     0.0            0.0
..                 …     …   …       …     …           …
523          -0.182578     1.0   0.0      0.0     0.0            0.0
524           0.486007     1.0   0.0      0.0     0.0            0.0
525          -0.182578     0.0   1.0      0.0     0.0            0.0
526          -0.182578     1.0   0.0      0.0     0.0            0.0
527          -0.182578     0.0   1.0      0.0     0.0            0.0

     generation_z  millennials  autumn  spring  summer  winter  Friday  \
0             0.0          1.0     0.0     0.0     1.0     0.0     0.0
1             0.0          1.0     0.0     0.0     1.0     0.0     0.0
2             1.0          0.0     0.0     1.0     0.0     0.0     1.0
3             0.0          1.0     0.0     0.0     1.0     0.0     0.0
4             0.0          1.0     0.0     0.0     1.0     0.0     0.0
..            …            …     …     …     …     …     …
523           0.0          1.0     0.0     1.0     0.0     0.0     0.0
524           0.0          1.0     0.0     1.0     0.0     0.0     0.0
525           0.0          1.0     0.0     1.0     0.0     0.0     0.0
526           0.0          1.0     0.0     0.0     1.0     0.0     0.0
527           0.0          1.0     0.0     0.0     1.0     0.0     1.0

     Monday  Saturday  Sunday  Thursday  Tuesday  Wednesday  afternoon  \
0       1.0       0.0     0.0       0.0      0.0        0.0        0.0
1       0.0       0.0     0.0       0.0      0.0        1.0        1.0
2       0.0       0.0     0.0       0.0      0.0        0.0        0.0
3       0.0       0.0     0.0       0.0      0.0        1.0        1.0
4       1.0       0.0     0.0       0.0      0.0        0.0        0.0
..      …       …     …       …      …        …        …
523     1.0       0.0     0.0       0.0      0.0        0.0        1.0
524     0.0       0.0     0.0       0.0      0.0        1.0        1.0
525     1.0       0.0     0.0       0.0      0.0        0.0        1.0
526     1.0       0.0     0.0       0.0      0.0        0.0        0.0
527     0.0       0.0     0.0       0.0      0.0        0.0        1.0

     evening  morning  night
0        0.0      1.0    0.0
1        0.0      0.0    0.0
2        0.0      1.0    0.0
3        0.0      0.0    0.0
4        0.0      1.0    0.0
..       …      …    …
523      0.0      0.0    0.0
524      0.0      0.0    0.0
525      0.0      0.0    0.0
```

```
526      0.0      1.0    0.0
527      0.0      0.0    0.0

[528 rows x 28 columns]
```

[226]:
```python
# Checking for multicollinearity straight from DF
import statsmodels.api as sm
def calculate_vif (data):
    vif_df = pd.DataFrame(columns=['Var', 'Vif'])
    x_var_names = data.columns
    for i in range(0, x_var_names.shape[0]):
        y= data[x_var_names[i]]
        x = data[x_var_names.drop([x_var_names[i]])]
        r_squared = sm.OLS(y,x).fit().rsquared
        vif = round(1/(1-r_squared), 2)
        vif_df.loc[i] = [x_var_names[i], vif]
    return vif_df.sort_values(by='Vif', axis = 0, ascending=False, inplace =␣
 ↪False)

calculate_vif(artists_vif_test)
```

```
<ipython-input-226-65a2d0285f9e>:10: RuntimeWarning:

divide by zero encountered in double_scalars
```

[226]:

	Var	Vif
14	spring	inf
15	summer	inf
26	morning	inf
25	evening	inf
24	afternoon	inf
23	Wednesday	inf
22	Tuesday	inf
21	Thursday	inf
20	Sunday	inf
19	Saturday	inf
18	Monday	inf
17	Friday	inf
16	winter	inf
27	night	inf
13	autumn	inf
12	millennials	inf
11	generation_z	inf
10	generation_x	inf
9	boomer	inf
8	unknown	inf

```
7                  male   inf
6                female   inf
1             playlists  5.25
4          repeat_count  4.25
0          number_songs  1.61
2         passion_score  1.09
3       avg_stream_time  1.08
5     featuring_artists  1.06
```

[227]:
```
!pip install imblearn
```

```
        .:::.     .::.
       …yy:    .yy.
       :.  .yy.    y.
           :y:   .:
           .yy  .:
            yy..:
            :y:.
            .y.
           .:.
       …:.
       :::.
```

```
• Project files and data should be stored in /project. This is shared among
everyone
  in the project.
• Personal files and configuration should be stored in /home/faculty.
• Files outside /project and /home/faculty will be lost when this server is
terminated.
• Create custom environments to setup your servers reproducibly.
```

```
Requirement already satisfied: imblearn in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (0.0)
Requirement already satisfied: imbalanced-learn in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from imblearn) (0.8.0)
Requirement already satisfied: numpy>=1.13.3 in
```

```
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from imbalanced-
learn->imblearn) (1.18.5)
Requirement already satisfied: scikit-learn>=0.24 in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from imbalanced-
learn->imblearn) (0.24.1)
Requirement already satisfied: joblib>=0.11 in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from imbalanced-
learn->imblearn) (0.16.0)
Requirement already satisfied: scipy>=0.19.1 in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from imbalanced-
learn->imblearn) (1.5.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from scikit-
learn>=0.24->imbalanced-learn->imblearn) (2.1.0)
```

[228]:

```python
#copying the labels in to have just the success column
smote_artists_labels = train_set['success'].copy()
```

[229]:

```python
# SMOTE
from imblearn.over_sampling import SMOTE #SMOTENC for handlign catagorical␣
↪variables as well
smote = SMOTE(random_state = 42)
model_train, model_test, label_train, label_test =␣
↪train_test_split(artists_vif_test, smote_artists_labels, test_size =0.2,␣
↪stratify=smote_artists_labels)
model_train_oversampled, label_train_oversampled = smote.
↪fit_resample(model_train, label_train)
smote_model = pd.DataFrame(model_train_oversampled, columns=model_train.columns)
```

[230]:

```python
label_train_oversampled.value_counts()
```

[230]:

```
1    378
0    378
Name: success, dtype: int64
```

[231]:

```python
label_train_oversampled
```

[231]:

```
0      0
1      0
2      0
3      0
4      0
      ..
751    1
752    1
753    1
754    1
```

```
755    1
Name: success, Length: 756, dtype: int64
```

[232]:

```
smote_model
```

[232]:

```
     number_songs  playlists  passion_score  avg_stream_time  repeat_count  \
0       -0.662549  -0.209863      -0.123886         0.010505     -0.164997
1       -0.662549  -0.259363      -0.123886         0.487781     -0.164997
2       -0.154450  -0.252291      -0.108775         0.814855     -0.164450
3        0.480673  -0.209863      -0.081846        -0.170821     -0.163902
4        0.099599  -0.238149      -0.050956        -0.389051     -0.162259
..            …          …              …                …             …
751      0.442383   2.146431      -0.087717        -0.369350      0.881837
752      2.611456   1.133405      -0.011010        -0.242287      0.429013
753     -0.391048  -0.168407      -0.086595        -0.350274     -0.136212
754      0.417092   4.840602      -0.033432         0.137649      4.687025
755      3.347088   2.911071      -0.032625        -0.034882      1.536526

     featuring_artists    female      male  unknown  boomer  generation_x  \
0            -0.182578  1.000000  0.000000      0.0     0.0           0.0
1            -0.182578  0.000000  1.000000      0.0     0.0           1.0
2            -0.182578  0.000000  1.000000      0.0     0.0           0.0
3            -0.182578  0.000000  1.000000      0.0     0.0           0.0
4            -0.182578  0.000000  1.000000      0.0     0.0           0.0
..                 …         …         …        …       …             …
751          -0.182578  1.000000  0.000000      0.0     0.0           0.0
752          -0.178819  1.000000  0.000000      0.0     0.0           0.0
753          -0.182578  0.000000  1.000000      0.0     0.0           0.0
754          -0.178269  0.749911  0.250089      0.0     0.0           0.0
755          -0.171445  1.000000  0.000000      0.0     0.0           0.0

     generation_z  millennials  autumn    spring    summer  winter  Friday  \
0             0.0          1.0     0.0  0.000000  1.000000     0.0     0.0
1             0.0          0.0     0.0  0.000000  1.000000     0.0     0.0
2             0.0          1.0     0.0  1.000000  0.000000     0.0     1.0
3             0.0          1.0     0.0  0.000000  1.000000     0.0     0.0
4             0.0          1.0     0.0  0.000000  0.000000     1.0     0.0
..              …            …       …         …         …       …       …
751           0.0          1.0     0.0  0.000000  0.000000     1.0     1.0
752           0.0          1.0     0.0  1.000000  0.000000     0.0     0.0
753           0.0          1.0     0.0  0.000000  1.000000     0.0     0.0
754           0.0          1.0     0.0  0.250089  0.749911     0.0     0.0
755           0.0          1.0     0.0  1.000000  0.000000     0.0     0.0

       Monday  Saturday  Sunday  Thursday  Tuesday  Wednesday  afternoon  \
0    0.000000  1.000000     0.0       0.0      0.0   0.000000        1.0
1    0.000000  1.000000     0.0       0.0      0.0   0.000000        1.0
```

```
2    0.000000  0.000000     0.0        0.0       0.0    0.000000         1.0
3    1.000000  0.000000     0.0        0.0       0.0    0.000000         0.0
4    0.000000  1.000000     0.0        0.0       0.0    0.000000         1.0
..        …         …     …       …       …        …         …
751  0.000000  0.000000     0.0        0.0       0.0    0.000000         1.0
752  1.000000  0.000000     0.0        0.0       0.0    0.000000         1.0
753  0.534346  0.465654     0.0        0.0       0.0    0.000000         1.0
754  0.250089  0.000000     0.0        0.0       0.0    0.749911         1.0
755  1.000000  0.000000     0.0        0.0       0.0    0.000000         1.0

     evening  morning  night
0        0.0      0.0    0.0
1        0.0      0.0    0.0
2        0.0      0.0    0.0
3        0.0      1.0    0.0
4        0.0      0.0    0.0
..       …       …     …
751      0.0      0.0    0.0
752      0.0      0.0    0.0
753      0.0      0.0    0.0
754      0.0      0.0    0.0
755      0.0      0.0    0.0

[756 rows x 28 columns]
```

[233]:
```
smote_model.to_csv('smote_model.csv', index = False)
```

[234]:
```
label_train_oversampled.to_csv('smote_label.csv', index = False)
```

2 Model Fitting

[235]:
```
np.random.seed(42)
```

[236]:
```
from sklearn.model_selection import train_test_split, StratifiedKFold, KFold
from sklearn.model_selection import cross_val_score,␣
↪cross_val_predict,GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.utils import class_weight
from sklearn.metrics import classification_report, roc_auc_score, f1_score,␣
↪mean_squared_error
from sklearn.metrics import accuracy_score, recall_score, precision_score,␣
↪f1_score, confusion_matrix, roc_curve, plot_confusion_matrix
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import Pipeline
from pprint import pprint
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
```

```python
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.base import BaseEstimator
```

[237]:
```python
# Cross-Validation with stratifiedKFolds

cv = StratifiedKFold(n_splits=5,shuffle=True)
def CV(x_train, y_train, clf, cv):
    aucs = []
    precisions = []
    recalls = []
    f1s = []
    accuracys=[]
    for train,test in cv.split(x_train, y_train):
        clf.fit(x_train.loc[train], y_train.loc[train])
        prediction = clf.predict(x_train.iloc[test])
        accuracy = accuracy_score(y_train[test], prediction)
        roc_auc = roc_auc_score(y_train[test], prediction)
        recall = recall_score(y_train[test], prediction)
        precision = precision_score(y_train[test], prediction)
        f1 = f1_score(y_train[test], prediction)
        accuracys.append(accuracy)
        aucs.append(roc_auc)
        precisions.append(precision)
        recalls.append(recall)
        f1s.append(f1)
        mean_accuracy = sum(accuracys)/len(accuracys)
        mean_auc = sum(aucs)/len(aucs)
        mean_precision = sum(precisions)/len(precisions)
        mean_recall = sum(recalls)/len(recalls)
        mean_f1 = sum(f1s)/len(f1s)
    print("Accuracy-Scores:", accuracys)
    print("Mean Accuracy-Score: %.4f"% (mean_accuracy))
    print("-------------------------")
    print("ROC-AUC-Scores:", aucs)
    print("Mean ROC-AUC-Score: %.4f"% (mean_auc))
    print("-------------------------")
    print("Precision scores:", precisions)
    print("Mean precision score: %.4f"% (mean_precision))
    print("-------------------------")
    print("Recall scores:", recalls)
    print("Mean recall score: %.4f"% (mean_recall))
    print("-------------------------")
    print("F1 scores:", f1s)
    print("Mean f1 score: %.4f"% (mean_f1))
```

[238]:

```
#defining the function for plotting the confusion matrix. Just assign a name␣
↪for the model
def model_evaluation(model, name,label_test, label_pred):
    print(f"\nMetrics for {name}")
    standard_model_metrics = ("Model Accuracy","Model ROC_AUC", "Model Recall",␣
↪"Model Precision", "Model F1")
    model_eval = pd.DataFrame(model, index=standard_model_metrics,␣
↪columns=[f"Score {name}"])
    model_eval.loc["Model Accuracy", f"Score {name}"] =␣
↪accuracy_score(label_test, label_pred)
    model_eval.loc["Model ROC_AUC", f"Score {name}"] =␣
↪roc_auc_score(label_test, label_pred)
    model_eval.loc["Model Recall",f"Score {name}"] =  recall_score(label_test,␣
↪label_pred)
    model_eval.loc["Model Precision",f"Score {name}"] =␣
↪precision_score(label_test, label_pred)
    model_eval.loc["Model F1", f"Score {name}"] = f1_score(label_test,␣
↪label_pred)
    return model_eval

def plot_confusion_mtx(model,name):

    disp=plot_confusion_matrix(model, model_test, label_test, normalize="true",␣
↪cmap=plt.cm.Blues,
                              values_format='.2f',
                              display_labels=["No Success", "Success"])
    disp.ax_.set_title(f"Confusion Matrix for {name}")
```

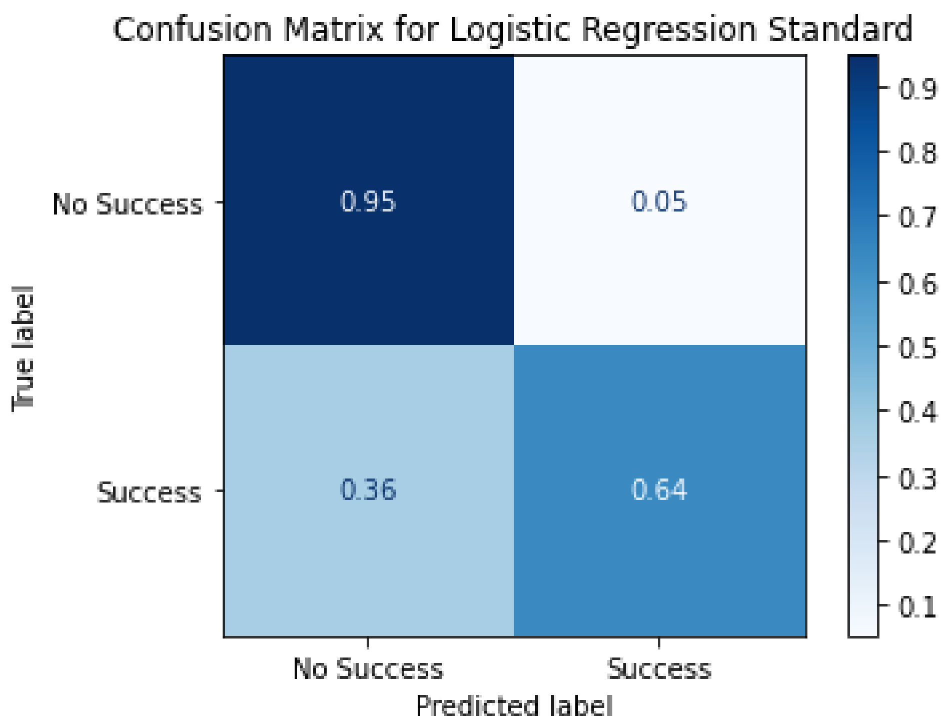
[239]:

```
log_clf = LogisticRegression(solver="liblinear", random_state=42)
log_clf.fit(smote_model, label_train_oversampled)
np.random.seed(42)
CV(smote_model, label_train_oversampled, log_clf, cv)
```

```
Accuracy-Scores: [0.8618421052631579, 0.8609271523178808, 0.847682119205298,
0.8609271523178808, 0.8344370860927153]
Mean Accuracy-Score: 0.8532
-------------------------
ROC-AUC-Scores: [0.8618421052631579, 0.860701754385965, 0.847280701754386,
0.861140350877193, 0.8348245614035087]
Mean ROC-AUC-Score: 0.8532
-------------------------
Precision scores: [0.8873239436619719, 0.8857142857142857, 0.8939393939393939,
0.8873239436619719, 0.8805970149253731]
Mean precision score: 0.8870
-------------------------
Recall scores: [0.8289473684210527, 0.8266666666666667, 0.7866666666666666,
0.8289473684210527, 0.7763157894736842]
```

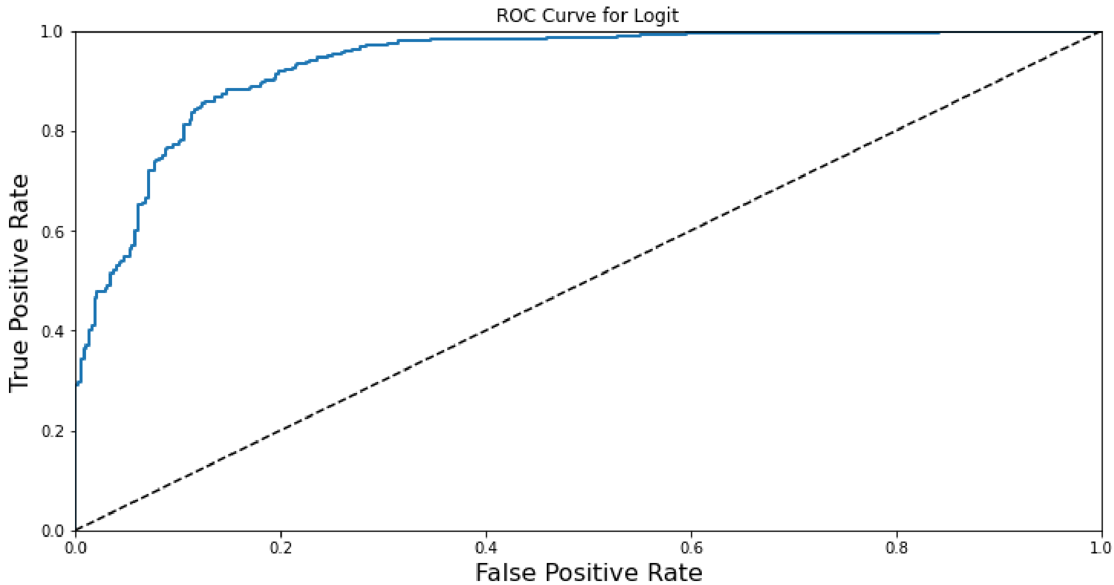
```
Mean recall score: 0.8095
-------------------------
F1 scores: [0.8571428571428571, 0.8551724137931035, 0.8368794326241135,
0.8571428571428571, 0.8251748251748251]
Mean f1 score: 0.8463
```

[240]:
```
plot_confusion_mtx(log_clf, "Logistic Regression Standard")
```



[241]:
```
#Predicting probabilities instead of labels
log_labels_prob = cross_val_predict(log_clf, smote_model,␣
↪label_train_oversampled,
                                cv=cv, method= 'decision_function')

#Defining ROC curve
fpr, tpr, thresholds = roc_curve(label_train_oversampled, log_labels_prob)

# Plotting ROC Curve for Logit
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
plt.figure(figsize=(12, 6))
```

```
plot_roc_curve(fpr, tpr)
plt.title('ROC Curve for Logit')
plt.show()
```



2.0.1 Grid Search for Logistic Regression and Model evaluation on test set

[242]:
```
import sys
import warnings

if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

[243]:
```
%%time
#Grid Search
from sklearn.model_selection import GridSearchCV
logreg = LogisticRegression()
grid_values = [{'solver': ["newton-cg", "lbfgs", "liblinear"],
                'multi_class': ['ovr', 'auto'],
                'C':[0.01,.09,1,5,10,20,24,28,29,30,100],
                'dual':[True,False],
                'max_iter':[110, 120, 130, 140,160,200]}]



logreg_grid = GridSearchCV(logreg, param_grid = grid_values, scoring =␣
 ↪'precision')
logreg_grid.fit(smote_model, label_train_oversampled)
```

```
#Predict values based on new parameters
label_pred = logreg_grid.predict(smote_model)

#showing the best estimators
print("Best estimators for the model :",logreg_grid.best_estimator_)

CV(smote_model, label_train_oversampled, logreg_grid, cv)
```

```
Best estimators for the model : LogisticRegression(C=0.09, dual=True,
max_iter=110, multi_class='ovr',
                   solver='liblinear')
Accuracy-Scores: [0.8421052631578947, 0.8278145695364238, 0.8344370860927153,
0.7814569536423841, 0.8211920529801324]
Mean Accuracy-Score: 0.8214
-------------------------
ROC-AUC-Scores: [0.8421052631578947, 0.8271929824561404, 0.8337719298245615,
0.7820175438596491, 0.8214912280701754]
Mean ROC-AUC-Score: 0.8213
-------------------------
Precision scores: [0.90625, 0.9016393442622951, 0.9166666666666666,
0.8412698412698413, 0.855072463768116]
Mean precision score: 0.8842
-------------------------
Recall scores: [0.7631578947368421, 0.7333333333333333, 0.7333333333333333,
0.6973684210526315, 0.7763157894736842]
Mean recall score: 0.7407
-------------------------
F1 scores: [0.8285714285714286, 0.8088235294117647, 0.8148148148148148,
0.7625899280575539, 0.8137931034482757]
Mean f1 score: 0.8057
CPU times: user 16min 37s, sys: 29.9 s, total: 17min 7s
Wall time: 4min 58s
```

[245]:
```
# New Model Evaluation metrics
#Predict values based on new parameters
label_pred = logreg_grid.predict(model_test)
logreg_gridsearch_eval=model_evaluation(logreg_grid, 'Logistic Regression␣
↪GridSearch',label_test, label_pred )
logreg_gridsearch_eval
```

Metrics for Logistic Regression GridSearch

[245]:

	Score Logistic Regression GridSearch
Model Accuracy	0.924528
Model ROC_AUC	0.797129
Model Recall	0.636364

```
Model Precision                              0.636364
Model F1                                     0.636364
```

2.1 AdaBoost model RandomizedSearch

[246]:
```
%%time
# random grid search

#from sklearn.model_selection import GridSearchCV

#model to be used

DTC = DecisionTreeClassifier(max_depth = 1)
ada_clf = AdaBoostClassifier(random_state=42, base_estimator = DTC)

grid_values = [{'n_estimators': [i**2 for i in range(1,50,3)],
                'learning_rate': [0.1, 0.5, 1]}]


# search across 75 different combinations, and use all available cores
ada_clf_random = RandomizedSearchCV(estimator = ada_clf, param_distributions =␣
↪grid_values, n_iter = 50, cv = 3,
                               verbose=2, random_state=42, n_jobs = -1, scoring␣
↪= 'precision')


# fit the model
ada_clf_random.fit(model_train_oversampled, label_train_oversampled)

# predict values based on new parameters
label_pred = ada_clf_random.predict(model_train_oversampled)

#showing the best estimators
print("Best estimators for the model :",ada_clf_random.best_estimator_)

CV(model_train_oversampled, label_train_oversampled, ada_clf_random, cv)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best estimators for the model :
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=0.1, n_estimators=1156, random_state=42)
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Accuracy-Scores: [0.9276315789473685, 0.9205298013245033, 0.9668874172185431,
```

```
0.9337748344370861, 0.9072847682119205]
Mean Accuracy-Score: 0.9312
-------------------------
ROC-AUC-Scores: [0.9276315789473685, 0.9207017543859649, 0.9669298245614035,
0.9337719298245614, 0.9069298245614035]
Mean ROC-AUC-Score: 0.9312
-------------------------
Precision scores: [0.9452054794520548, 0.8987341772151899, 0.9605263157894737,
0.9342105263157895, 0.8690476190476191]
Mean precision score: 0.9215
-------------------------
Recall scores: [0.9078947368421053, 0.9466666666666667, 0.9733333333333334,
0.9342105263157895, 0.9605263157894737]
Mean recall score: 0.9445
-------------------------
F1 scores: [0.9261744966442953, 0.9220779220779222, 0.9668874172185431,
0.9342105263157895, 0.9125000000000001]
Mean f1 score: 0.9324
CPU times: user 10.6 s, sys: 560 ms, total: 11.1 s
Wall time: 5min 12s
```

[247]:

```
# predict values based on new parameters
label_pred = ada_clf_random.predict(model_test)

## ADD EVALUTATION FUNCTION HERE

# New Model Evaluation metrics
ada_clf_random_eval=model_evaluation(ada_clf_random, 'Ada Boost Classifier',␣
↪label_test, label_pred)
ada_clf_random_eval
```

Metrics for Ada Boost Classifier

[247]:

	Score Ada Boost Classifier
Model Accuracy	0.830189
Model ROC_AUC	0.664115
Model Recall	0.454545
Model Precision	0.294118
Model F1	0.357143

2.2 Gradient Boost model RandomizedSearch

[248]:

```
%%time
# random grid search

#maximum number of levels in tree
```

```python
max_depth = [i for i in range(1,150,3)]

#number of features to consider at every split
max_features = ['auto', 'sqrt']

#minimum number of samples required to split a node
min_samples_split = [4, 8, 10, 12, 14, 16]

#minimum number of samples required at each leaf node
min_samples_leaf = [2, 3, 4, 5, 8]

#maximum number of leaf nodes in tree
max_leaf_nodes = [8, 10, 12, 15, 20, 25, 30, 40, 50, 55]

#create random grid
new_grid = {'max_depth': max_depth,
               'max_features': max_features,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'max_leaf_nodes': max_leaf_nodes}


#model to be used
gb_clf = GradientBoostingClassifier(random_state=42)



# search across 50 different combinations, and use all available cores
gb_clf_random = RandomizedSearchCV(estimator = gb_clf, param_distributions =
 ↪new_grid, n_iter = 50, cv = 3,
                               verbose=2, random_state=42, n_jobs = -1, scoring
 ↪= 'precision')

# fit the model
gb_clf_random.fit(model_train_oversampled, label_train_oversampled)

# predict values based on new parameters
label_pred = gb_clf_random.predict(model_train_oversampled)



## ADD EVALUTATION FUNCTION HERE
#showing the best estimators
print("Best estimators for the model :",gb_clf_random.best_estimator_)

CV(model_train_oversampled, label_train_oversampled, gb_clf_random, cv)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best estimators for the model : GradientBoostingClassifier(max_depth=148,
max_features='sqrt',
                           max_leaf_nodes=40, min_samples_leaf=2,
                           min_samples_split=4, random_state=42)
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Accuracy-Scores: [0.9407894736842105, 0.9337748344370861, 0.9470198675496688,
0.9668874172185431, 0.9801324503311258]
Mean Accuracy-Score: 0.9537
-------------------------
ROC-AUC-Scores: [0.9407894736842106, 0.9337719298245614, 0.9471052631578947,
0.9666666666666667, 0.9801754385964914]
Mean ROC-AUC-Score: 0.9537
-------------------------
Precision scores: [0.9036144578313253, 0.9333333333333333, 0.935064935064935,
0.9382716049382716, 0.9866666666666667]
Mean precision score: 0.9394
-------------------------
Recall scores: [0.9868421052631579, 0.9333333333333333, 0.96, 1.0,
0.9736842105263158]
Mean recall score: 0.9708
-------------------------
F1 scores: [0.9433962264150944, 0.9333333333333333, 0.9473684210526316,
0.9681528662420382, 0.9801324503311258]
Mean f1 score: 0.9545
CPU times: user 3.03 s, sys: 164 ms, total: 3.19 s
Wall time: 39.3 s
```

[249]:

```
# New Model Evaluation metrics
# predict values based on new parameters
label_pred = gb_clf_random.predict(model_test)
gb_clf_random_eval=model_evaluation(gb_clf_random, 'Gradient Boosting',␣
↪label_test, label_pred)
gb_clf_random_eval
```

Metrics for Gradient Boosting

[249]:

	Score Gradient Boosting
Model Accuracy	0.90566
Model ROC_AUC	0.666029
Model Recall	0.363636
Model Precision	0.571429

Model F1   0.444444

[ ]:

```

```

2.3 XGBoost

[250]:

```
!pip install xgboost
```

```
         .::::.     .::.
        …yy:    .yy.
        :.  .yy.    y.
             :y:   .:
             .yy  .:
              yy..:
              :y:.
              .y.
             .:.
        …:.
        :::.
```

• Project files and data should be stored in /project. This is shared among everyone
  in the project.
• Personal files and configuration should be stored in /home/faculty.
• Files outside /project and /home/faculty will be lost when this server is terminated.
• Create custom environments to setup your servers reproducibly.

```
Requirement already satisfied: xgboost in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (1.3.3)
Requirement already satisfied: scipy in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from xgboost) (1.5.0)
Requirement already satisfied: numpy in
/opt/anaconda/envs/Python3/lib/python3.8/site-packages (from xgboost) (1.18.5)
```

[251]:
```
import xgboost as xgb
xgb_log = xgb.XGBClassifier (objective ='binary:logistic', colsample_bytree=0.
↪3,learning_rate=0.1, max_depth=5, alpha=10, n_estimators=10,␣
↪use_label_encoder=False, random_state=42, verbosity=0)
CV(smote_model, label_train_oversampled, xgb_log, cv)
```

```
Accuracy-Scores: [0.9144736842105263, 0.9205298013245033, 0.9006622516556292,
0.8807947019867549, 0.9403973509933775]
Mean Accuracy-Score: 0.9114
-------------------------
ROC-AUC-Scores: [0.9144736842105263, 0.9207894736842105, 0.9008771929824562,
0.8806140350877193, 0.9401754385964912]
Mean ROC-AUC-Score: 0.9114
-------------------------
Precision scores: [0.8620689655172413, 0.8888888888888888, 0.875, 0.8625,
0.9135802469135802]
Mean precision score: 0.8804
-------------------------
Recall scores: [0.9868421052631579, 0.96, 0.9333333333333333,
0.9078947368421053, 0.9736842105263158]
Mean recall score: 0.9524
-------------------------
F1 scores: [0.9202453987730062, 0.923076923076923, 0.9032258064516129,
0.8846153846153847, 0.9426751592356688]
Mean f1 score: 0.9148
```

[253]:
```
#Defining paramters for search, XGBoost

param_grid = [{'max_depth':[5,7,9], 'n_estimators':[500,700,1000], 'booster':
↪['gbtree'],
              'learning_rate':[0.1,0.2,0.3], 'objective':['binary:logistic'],
              'use_label_encoder':[False], 'verbosity':[0], 'random_state':
↪[42]}]


#Initiating Search

xgb_random_search = RandomizedSearchCV(xgb_log, param_distributions=param_grid,␣
↪cv=5,n_iter=5,
                          scoring='precision',
                          return_train_score=True)

xgb_random_search.fit(smote_model, label_train_oversampled)

CV(smote_model, label_train_oversampled, xgb_random_search, cv)
```

```
Accuracy-Scores: [0.9605263157894737, 0.9072847682119205, 0.8675496688741722,
```

```
0.8874172185430463, 0.9072847682119205]
Mean Accuracy-Score: 0.9060
-------------------------
ROC-AUC-Scores: [0.9605263157894737, 0.9074561403508771, 0.8679824561403509,
0.887017543859649, 0.9068421052631579]
Mean ROC-AUC-Score: 0.9060
-------------------------
Precision scores: [0.972972972972973, 0.8860759493670886, 0.8235294117647058,
0.8470588235294118, 0.8604651162790697]
Mean precision score: 0.8780
-------------------------
Recall scores: [0.9473684210526315, 0.9333333333333333, 0.9333333333333333,
0.9473684210526315, 0.9736842105263158]
Mean recall score: 0.9470
-------------------------
F1 scores: [0.9599999999999999, 0.9090909090909091, 0.8749999999999999,
0.8944099378881987, 0.9135802469135803]
Mean f1 score: 0.9104
```

[254]:
```
print("Best estimators for the model :",xgb_random_search.best_estimator_)
```

```
Best estimators for the model : XGBClassifier(alpha=10, base_score=0.5,
booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=500, n_jobs=8, num_parallel_tree=1, random_state=42,
              reg_alpha=10, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', use_label_encoder=False,
              validate_parameters=1, verbosity=0)
```

[255]:
```
label_pred = xgb_random_search.predict(model_test)
# predict values based on new parameters
label_pred = xgb_random_search.predict(model_test)
gb_clf_random_eval=model_evaluation(xgb_random_search, 'XGB RandomSearch',␣
↪label_test, label_pred)
gb_clf_random_eval
```

```
Metrics for XGB RandomSearch
```

[255]:

	Score XGB RandomSearch
Model Accuracy	0.915094
Model ROC_AUC	0.872249
Model Recall	0.818182
Model Precision	0.5625
Model F1	0.666667

[ ]:

```

```

[ ]:

```

```

2.3.1 Random Forest Model

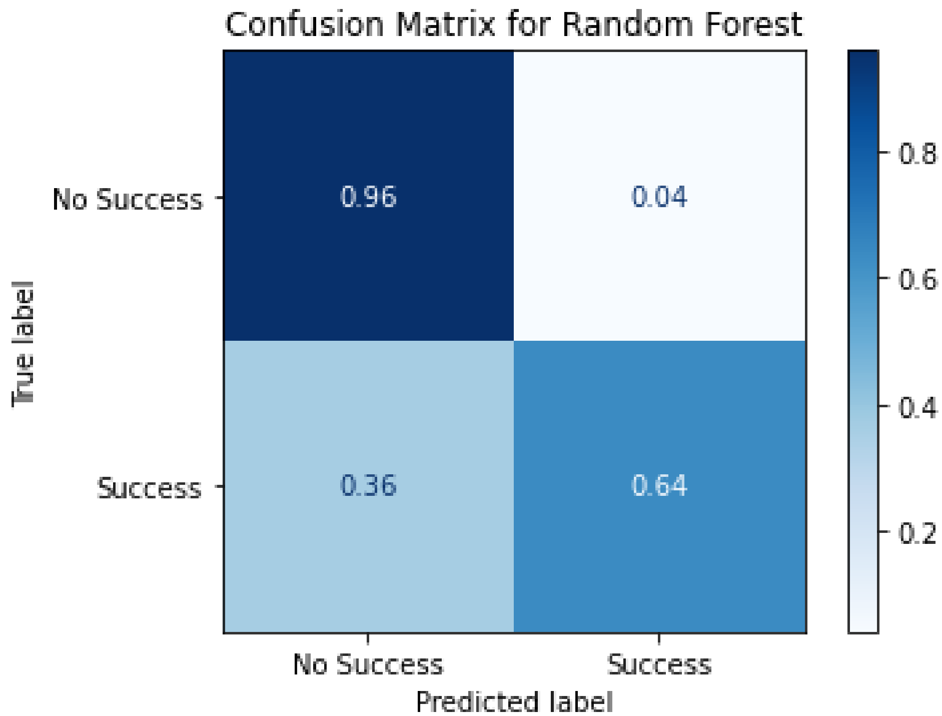
[256]:

```
#Import Random Forest
from sklearn.ensemble import RandomForestClassifier

#Defining Random Forest
forest_clf = RandomForestClassifier(n_estimators=100, random_state=42 )
forest_clf.fit(smote_model, label_train_oversampled)
np.random.seed(42)
CV(smote_model, label_train_oversampled, forest_clf, cv)
```

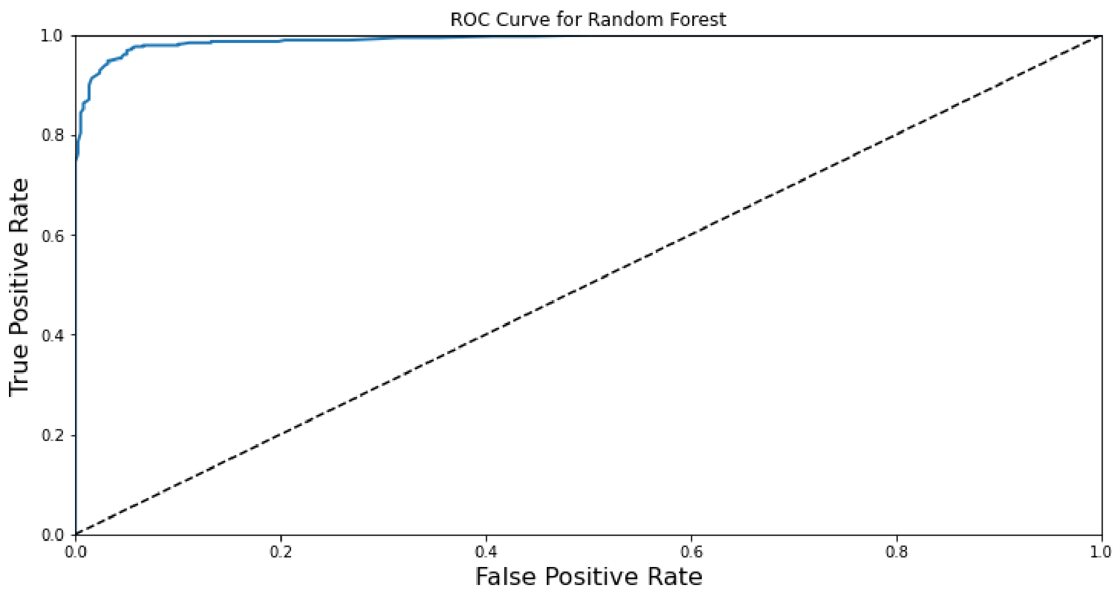
```
Accuracy-Scores: [0.9407894736842105, 0.9470198675496688, 0.9337748344370861,
0.9536423841059603, 0.9668874172185431]
Mean Accuracy-Score: 0.9484
-------------------------
ROC-AUC-Scores: [0.9407894736842106, 0.9472807017543861, 0.9338596491228071,
0.9535964912280702, 0.9667543859649123]
Mean ROC-AUC-Score: 0.9485
-------------------------
Precision scores: [0.9036144578313253, 0.9135802469135802, 0.922077922077922,
0.948051948051948, 0.9493670886075949]
Mean precision score: 0.9273
-------------------------
Recall scores: [0.9868421052631579, 0.9866666666666667, 0.9466666666666667,
0.9605263157894737, 0.9868421052631579]
Mean recall score: 0.9735
-------------------------
F1 scores: [0.9433962264150944, 0.9487179487179487, 0.9342105263157895,
0.9542483660130718, 0.967741935483871]
Mean f1 score: 0.9497
```

[257]:

```
plot_confusion_mtx(forest_clf, "Random Forest")
```



[258]:
```
#Predicting probabilities instead of labels
forest_labels_prob = cross_val_predict(forest_clf, smote_model,␣
↪label_train_oversampled,
                                cv=cv, method= 'predict_proba')[:,1]

#Defining ROC curve
fpr, tpr, thresholds = roc_curve(label_train_oversampled, forest_labels_prob)

# Plotting ROC Curve for Random Forest.
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate', fontsize=16)
    plt.ylabel('True Positive Rate', fontsize=16)
plt.figure(figsize=(12, 6))
plot_roc_curve(fpr, tpr)
plt.title('ROC Curve for Random Forest')
plt.show()
```



[259]:

```
#Randomized Search for Random Forest
param_grid = [{'n_estimators': [50,100,300,500,700],
              'criterion':['gini'], 'max_features': [10,20,30,60],
              'bootstrap': [False], 'random_state':[100]}]

forest_random_search = RandomizedSearchCV(forest_clf,␣
 ↪param_distributions=param_grid, cv=5,n_iter=5,
                          scoring='precision',
                          return_train_score=True)

forest_random_search.fit(smote_model, label_train_oversampled)

# predict values based on new parameters
label_pred = forest_random_search.predict(smote_model)

CV(smote_model, label_train_oversampled, forest_random_search, cv)
```

```
Accuracy-Scores: [0.9671052631578947, 0.9536423841059603, 0.9536423841059603,
0.9470198675496688, 0.9337748344370861]
Mean Accuracy-Score: 0.9510
-------------------------
ROC-AUC-Scores: [0.9671052631578948, 0.9538596491228071, 0.9539473684210527,
0.9469298245614035, 0.9337719298245614]
Mean ROC-AUC-Score: 0.9511
-------------------------
Precision scores: [0.9493670886075949, 0.925, 0.9146341463414634,
0.9358974358974359, 0.9342105263157895]
Mean precision score: 0.9318
```

```
-------------------------
Recall scores: [0.9868421052631579, 0.9866666666666667, 1.0, 0.9605263157894737,
0.9342105263157895]
Mean recall score: 0.9736
-------------------------
F1 scores: [0.967741935483871, 0.9548387096774195, 0.9554140127388536,
0.948051948051948, 0.9342105263157895]
Mean f1 score: 0.9521
```

[260]:
```
# predict values based on new parameters
label_pred = forest_random_search.predict(model_test)
forest_random_search_eval=model_evaluation(forest_random_search, 'Random Forest␣
↪RandomSearch', label_test, label_pred)
forest_random_search_eval
```

```
Metrics for Random Forest RandomSearch
```

[260]:

	Score Random Forest RandomSearch
Model Accuracy	0.924528
Model ROC_AUC	0.797129
Model Recall	0.636364
Model Precision	0.636364
Model F1	0.636364

Feature Importances

[261]:
```
#Final_model is the grid_search for the best model with Random Search
def feature_importance(final_model):
    feature_importances = final_model.best_estimator_.coef_[0]
    labels = smote_model.columns.tolist()
    importance_ordered = sorted(zip(feature_importances,labels), reverse=True)
    return (importance_ordered)
    return (feature_importances)
```

[262]:
```
feature_importance(logreg_grid)
```

[262]:
```
[(1.459332963763078, 'playlists'),
 (0.5497070597140369, 'repeat_count'),
 (0.5209774185642757, 'millennials'),
 (0.49606377856241923, 'afternoon'),
 (0.4859309853756733, 'spring'),
 (0.40718862475785506, 'number_songs'),
 (0.24021589559162365, 'Friday'),
 (0.13764558819835448, 'featuring_artists'),
 (0.07110081244868058, 'Saturday'),
 (0.07068401263363477, 'female'),
 (0.05694564338852501, 'Monday'),
```

```
 (0.025046746981833224, 'autumn'),
 (-0.013985103443981784, 'generation_z'),
 (-0.028540624891901907, 'Tuesday'),
 (-0.03226198567744498, 'unknown'),
 (-0.03842192473624234, 'male'),
 (-0.05315073169295868, 'night'),
 (-0.05503225883874988, 'summer'),
 (-0.06604734805881016, 'Wednesday'),
 (-0.08672656454139839, 'evening'),
 (-0.11477796128806263, 'Sunday'),
 (-0.1407026973403446, 'passion_score'),
 (-0.15889631497010664, 'Thursday'),
 (-0.19657666622107156, 'boomer'),
 (-0.31041554667927607, 'generation_x'),
 (-0.3561863801081144, 'morning'),
 (-0.36718093697334825, 'avg_stream_time'),
 (-0.45594537129880786, 'winter')]
```

[263]:

```
labels = smote_model.columns.tolist()
#Calculate the importance
model = LogisticRegression(C=0.01, max_iter=110, multi_class='ovr',␣
 ↪solver='newton-cg')
model.fit(smote_model, label_train_oversampled)
importance = pd.DataFrame(model.coef_[0], index = labels, columns =␣
 ↪["Importance"]).sort_values("Importance", ascending = False)
#Top 10 most important features
importance.head(10).style.background_gradient(sns.light_palette('#6495ED',␣
 ↪as_cmap = True))
```

[263]: <pandas.io.formats.style.Styler at 0x7f3b1c03ee50>

[264]:

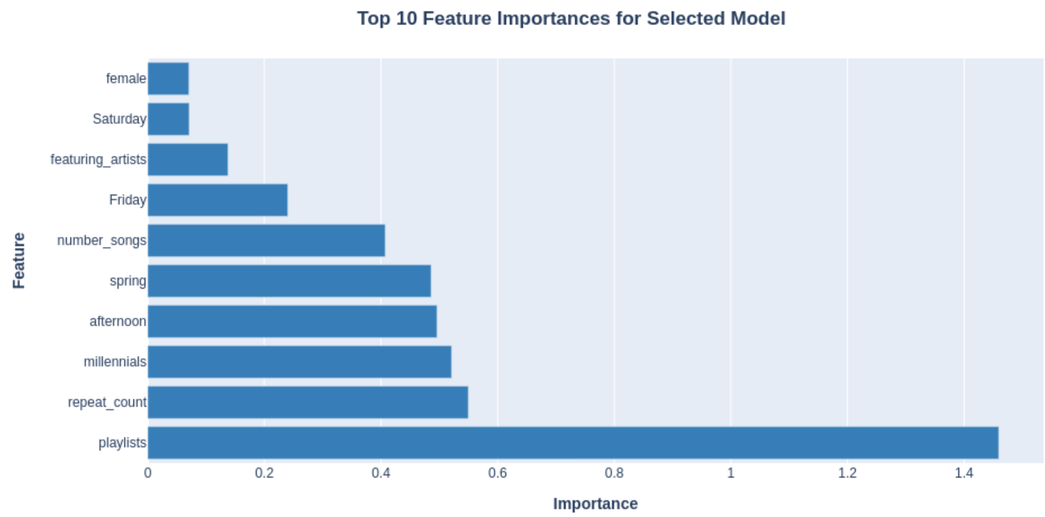
```
# #Want to explore the attributes above further by visualising them
# feature_list = feature_importances.tolist() #transforoming the feature␣
 ↪importances into a list.
# data_dict = dict(zip(labels,feature_list)) #Creating a dictionary with the␣
 ↪importance numbers and the attribute names
# df2 = pd.DataFrame(data=feature_list,index=labels) #Creating a dataframe of␣
 ↪the attributes, with attribute names being the index
# df2.columns=['importance']
```

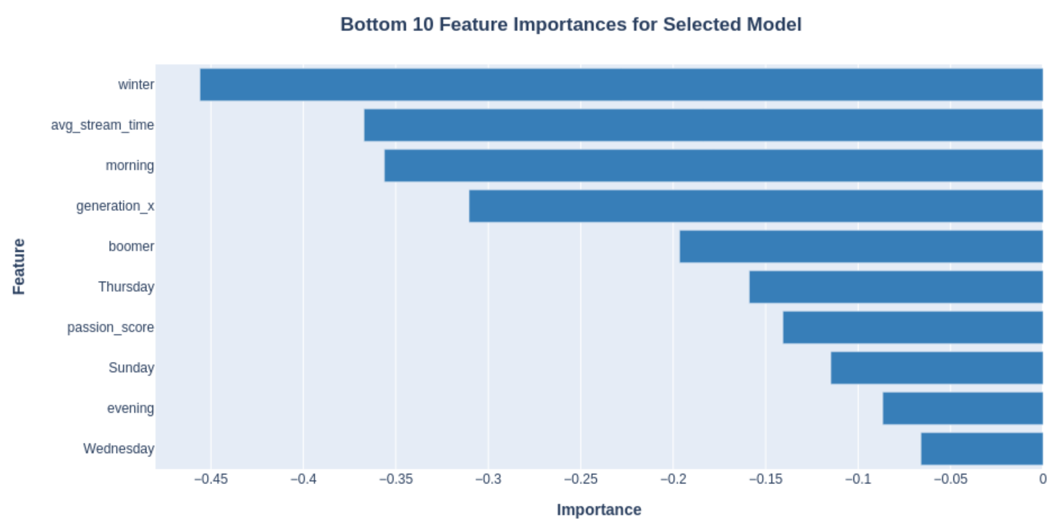
[265]:

```
#Making a plot function for the top 10 features, final_model = best␣
 ↪random_search
def feature_importance_plot(final_model,name):
    feature_importances = final_model.best_estimator_.coef_[0]
    labels = smote_model.columns.tolist()
```

```python
    feature_list = feature_importances.tolist() #transforoming the feature importances into a list.
    data_dict = dict(zip(labels,feature_list)) #Creating a dictionary with the importance numbers and the attribute names
    df2 = pd.DataFrame(data=feature_list,index=labels) #Creating a dataframe of the attributes, with attribute names being the index
    df2.columns=['importance']
    df2.reset_index(inplace=True)
    df2 = df2.sort_values(by='importance', ascending=False) #Sorting the variables

    #Plotting top 10 most important features
    fig = px.bar(df2[:10], x="importance", y="index",color_discrete_sequence=px.colors.qualitative.Set1[1:4])
    fig.update_layout(title={'text': '<b>Top 10 Feature Importances for Selected Model</b>','x':0.5},
                      yaxis_title_text='<b>Feature</b>',
                      xaxis_title_text='<b>Importance</b>')
    fig.write_image(f"./Feature Importance {name} Most important.pdf")

    #Plotting top 10 least important features
    fig1 = px.bar(df2[-10:], x="importance", y="index",
    color_discrete_sequence= px.colors.qualitative.Set1[1:4])
    fig1.update_layout(title={'text': '<b>Bottom 10 Feature Importances for Selected Model</b>','x':0.5},
                      yaxis_title_text='<b>Feature</b>',
                      xaxis_title_text='<b>Importance</b>')
    fig1.write_image(f"./Feature Importance {name} Least important.pdf")

    return(fig.show(),fig1.show())
```

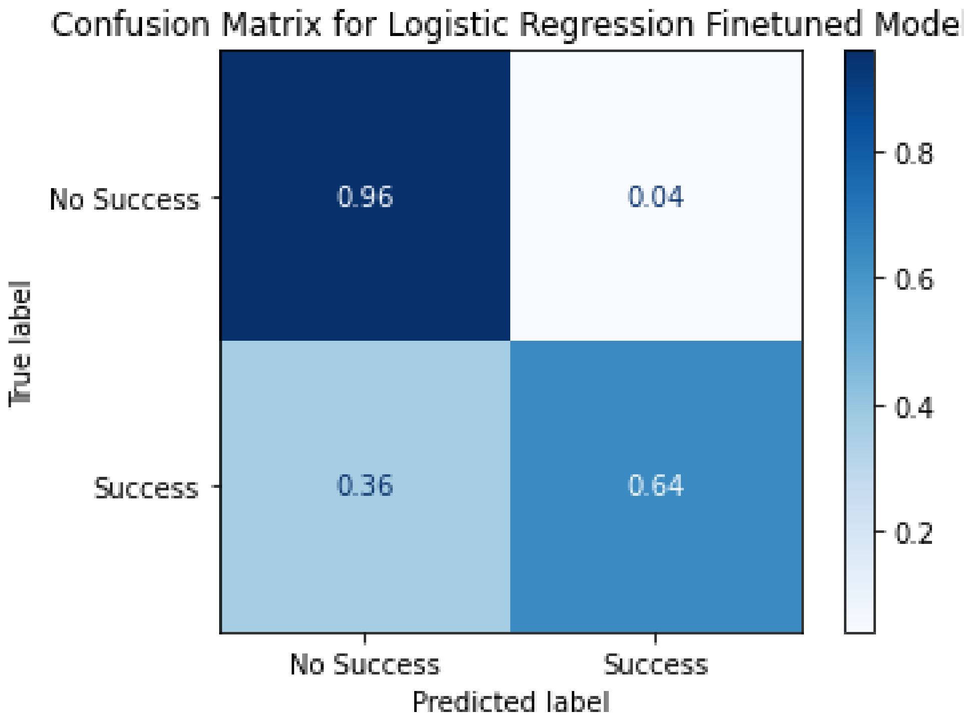
[266]:
```python
feature_importance_plot(logreg_grid, "Logistic Regression")
```





[266]: (None, None)

[267]:
```
plot_confusion_mtx(logreg_grid, "Logistic Regression Finetuned Model")
plt.savefig("./Logistic Regression RandomSearch.pdf")
```


Confusion Matrix for Logistic Regression Finetuned Model
No Success
Success
True label
0.96
0.04
0.36
0.64
No Success
Success
Predicted label
0.8
0.6
0.4
0.2

```
[ ]:
```