Final Project: Data Cleaning and Simulations!!

Welcome to our final project our team is ****Zane, Kenadi, and AJ**** !!

We wanted to study what factors affect song popularity and wanted to run a simulation to predict popularity based on these factors. Before we start this was our main hypothesis.

We looked at the data and saw many different measures such as available markets, duration, and spotify metrics (loudness, liveness, danceability, energy, acousticness, valence) Valence is the general positive or negative feel of the song (0.0 to 1.0)!!

Ex) a song of 0 valence is very negative -- a song of 1 valence is very happy sounding.

Hypothesis: We predict more popular artists will have a larger fan bases and will be more popular. We also predict that Spotify's danceability/energy predictors are positively correlated the popularity, yet this is influenced by other data such as marketing and genre which is not found in our data set.

Here is our cleaning of the data

In [1]:
```
#!pip install --upgrade altair
```

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
df=pd.read_csv('spotify_data_1986_2023.csv') #import dataset
print(df.columns)
```

```
Index(['Unnamed: 0.1', 'Unnamed: 0', 'track_id', 'track_name', 'popularity',
       'available_markets', 'disc_number', 'duration_ms', 'explicit',
       'track_number', 'href', 'album_id', 'album_name', 'album_release_dat
e',
       'album_type', 'album_total_tracks', 'artists_names', 'artists_ids',
       'principal_artist_id', 'principal_artist_name', 'artist_genres',
       'principal_artist_followers', 'acousticness', 'analysis_url',
       'danceability', 'energy', 'instrumentalness', 'key', 'liveness',
       'loudness', 'mode', 'speechiness', 'tempo', 'time_signature', 'valenc
e',
       'year', 'duration_min'],
      dtype='object')
```

Remove Irrelevant Columns

In [5]:
```
dropping = ['Unnamed: 0.1', 'Unnamed: 0', 'href', 'artists_ids', 'principal_
df = df.drop(dropping, axis = 1)
df
```

Out[5]:

	track_id	track_name	popularity	
0	2A6yzRGMgSQCUapR2ptm6A	True Colors	73	AR;AU;AT;BE;BO;BR;BG;CA;
1	3gKwVWwKmeuFtPubICbOGc	Paul Revere	61	AR;AU;AT;BE;BO;BR;BG;CA;
2	2tY1gxCKslfXLFpFofYmJQ	Brass Monkey	68	AR;AU;AT;BE;BO;BR;BG;CA;
3	31dqpLUModJWNbxrXu6TWd	Shot in the Dark	66	AR;AU;AT;BE;BO;BR;BG;CA;
4	00vYs0qZA40Z8AAaN7xmMO	Manic Monday	63	AE;BH;EG;GB;IE;IQ;JO;
...	...	...	...	
11445	4nrPB8O7Y7wsOCJdgXkthe	Shakira: Bzrp Music Sessions, Vol. 53	89	AR;AU;AT;BE;BO;BR;BG;CA;
11446	7Lkxvfl2rkNYWS4kBDCQtN	Las Morras	81	AR;AU;AT;BE;BO;BR;BG;CA;
11447	6UoKX6uLJwhsnyTp5k5StP	The Painter	75	AR;AU;AT;BE;BO;BR;BG;CA;
11448	4ZYAU4A2YBtINdqOUtc7T2	Red Ruby Da Sleeze	78	AR;AU;AT;BE;BO;BR;BG;CA;
11449	5vZoQQ1hH5L2s4Y8G86ksg	Angels (Don't Always Have Wings)	74	AR;AU;AT;BE;BO;BR;BG;CA;

11450 rows × 30 columns

Look at the amount of letters and amount of words in the track name

First let's drop the duplicates

In [6]:
```
df.drop_duplicates(subset='track_name', inplace=True) #first drop duplicates
df[df['track_name'] == "Gonna Make You Sweat (Everybody Dance Now) (feat. Fr
```

Out[6]:

	track_id	track_name	popularity	
1246	7eheoW4hxrH51ww2QePVwF	Gonna Make You Sweat (Everybody Dance Now) (fe...	71	AR;AU;AT;BE;BO;BR;BG;CA;CL,

1 rows × 30 columns

Now let's start looking at the names, there are some special cases to see that we don't want counted when we sum the letters

In [7]:
```
special_cases = df[
    df['track_name'].str.contains(r'\([A-Za-z0-9 ].*\)$', regex=True) | #has
    df['track_name'].str.contains(r'-\s*[A-Za-z].*$', regex=True) | #has a c
    df['track_name'].str.contains(r':\s*[A-Za-z].*$', regex=True) #has a col
]

# Show only the track_name column
print(special_cases['track_name'])

print(len(special_cases))
```

```
26                                  Battery (Remastered)
27                        Master of Puppets (Remastered)
31                              Only You (And You Alone)
34       Holding Out for a Hero - From "Footloose" Soun...
47                       Walk This Way (feat. Aerosmith)
                               ...
11429                         Pull Up (feat. 21 Savage)
11430                                   MONTAGEM - PR FUNK
11444          AMERICA HAS A PROBLEM (feat. Kendrick Lamar)
11445               Shakira: Bzrp Music Sessions, Vol. 53
11449                    Angels (Don't Always Have Wings)
Name: track_name, Length: 1696, dtype: object
1696
```

So it looks like there are 1696 cases, wow that is a lot!

Let's try to remove all of the irrelevant ones while keeping things like "(Live)" or parts of the title like "Only You (And You Alone)"

In [8]:
```
### clean track names
import re

df['clean_track_name'] = df['track_name'].str.replace(
    r'\s*(\((?:feat\.?|featuring|with)[^)]*\)|\([^)]*(revisited|remaster(?:e
    '',
    regex=True,
```

```
    flags=re.IGNORECASE, #replaces my key words regardless of if it is capit
).str.strip()

# note for str.replace:
# \s*                          → optional spaces before extra info
# \( ... \)                    → matches parentheses
# (?:feat\.?|featuring|with)  → only removes parentheses if they start with
# [^)]*                        → match everything inside the parentheses unti
# \([^)]*(revisited|remaster(?:ed)?)[^)]*  → removes parentheses containing
# [-–—].*                       → match a dash (-), en dash (–), or em dash (
# :.*                           → match a colon and everything after it (at t
# $                             → only match at the end of the string
# flags=re.IGNORECASE           → makes regex case-insensitive (so "Remastere
# .str.strip()                  → removes extra spaces left at the beginning

#move clean tarck name so it is next to track name
clean_col = df.pop('clean_track_name')
df.insert(2, 'clean_track_name', clean_col)

### Count letters can make new column!

# Count letters + numbers (all languages)
letter_num_count = df['clean_track_name'].apply(lambda x: sum(c.isalnum() fo

# Remove existing column if it already is there - this will let me edit the
if 'letters_track_name' in df.columns:
    df.drop(columns='letters_track_name', inplace=True)

# Insert next to clean track name column so I can see them side by side
df.insert(3, 'letters_track_name', letter_num_count)

#look at it to check :)
pd.set_option('display.max_rows', 50)
df[['track_name', 'clean_track_name', "letters_track_name"]].sample(50, ranc
```

Out[8]:

	track_name	clean_track_name	letters_track_name
5634	A Boy Brushed Red Living In Black And White	A Boy Brushed Red Living In Black And White	35
9756	Unbothered	Unbothered	10
1348	Is She Weird	Is She Weird	10
4171	Unpretty	Unpretty	8
1893	Black or White	Black or White	12
4594	Yo No Soy Esa Mujer	Yo No Soy Esa Mujer	15
3820	Quizás Si, Quizás No	Quizás Si, Quizás No	16
8452	r - Cali	r	1
11004	No Se Va - EN VIVO	No Se Va	6
6809	の	の	5
8955	Delusions of Saviour	Delusions of Saviour	18
33	Talk Dirty To Me	Talk Dirty To Me	13
11127	PAINTING PICTURES	PAINTING PICTURES	16
3383	Jump On It	Jump On It	8
7498	The Resistance	The Resistance	13
2651	Y Es Que La Quiero	Y Es Que La Quiero	14
5266	Such Great Heights - Remastered	Such Great Heights	16
7166	Evacuate The Dancefloor - Radio Edit	Evacuate The Dancefloor	21
3785	Da Art of Storytellin' (Pt. 1)	Da Art of Storytellin' (Pt. 1)	21
10181	Another Life	Another Life	11
9658	Blue Tacoma	Blue Tacoma	10
3692	A Spoonful of Sugar - From "Mary Poppins" / So...	A Spoonful of Sugar	16
5032	I Can See Clearly Now	I Can See Clearly Now	17
4151	Un Desengaño	Un Desengaño	11
5181	Don't Forget Me	Don't Forget Me	12
5440	Figure.09	Figure.09	8
10969	Vegas (From the Original Motion Picture Soundt...	Vegas (From the Original Motion Picture Soundt...	48
556	Moonglow	Moonglow	8

	track_name	clean_track_name	letters_track_name
8073	Inténtalo (feat. América Sierra & El Bebeto)	Inténtalo	9
1378	The Ghetto	The Ghetto	9
2984	Secret Garden	Secret Garden	12
3973	Así Fue - En Vivo [Desde el Instituto Nacional...	Así Fue	6
3994	the city	the city	7
1140	Seven Year Ache	Seven Year Ache	13
1466	Tiempo De Vals	Tiempo De Vals	12
3968	Feelin' Good Again	Feelin' Good Again	15
10350	GIVE HEAVEN SOME HELL	GIVE HEAVEN SOME HELL	18
9678	El Color de Tus Ojos	El Color de Tus Ojos	16
9137	Party Monster	Party Monster	12
7615	Rolling in the Deep	Rolling in the Deep	16
2684	Hang on to Your Love	Hang on to Your Love	16
9271	walk away as the door slams	walk away as the door slams	22
1369	Something's Gotta Give	Something's Gotta Give	19
10530	BookBag 2.0 (feat. Polo G)	BookBag 2.0	9
2317	Take The Power Back	Take The Power Back	16
2018	Drivin' My Life Away	Drivin' My Life Away	16
321	Welcome To The Jungle	Welcome To The Jungle	18
5605	Ain't No Use in Tryin'	Ain't No Use in Tryin'	16
10333	Romantic Lover	Romantic Lover	13
7207	Help I'm Alive	Help I'm Alive	11

Okay so it is not perfect, there are some weird exceptions but that is okay. I got most of the stuff out. Let me look at the full data set again just for clarity.

In [9]:
```
df.sample(10, random_state=46)
```

Out[9]:

	track_id	track_name	clean_track_name	letters_track_nan
10915	3k3NWokhRRkEPhCzPmV8TW	Ojitos Lindos	Ojitos Lindos	
3950	1fotoYONO343JjbC8XvPSl	Moment Of Truth	Moment Of Truth	
4714	6uRH1qMz30ZBwwUG0IYE5s	Dance With Me	Dance With Me	
6885	4gzeYkzuzxuzAUTsGcdjqA	It Won't Be Like This For Long	It Won't Be Like This For Long	2
5544	3xrn9i8zhNZsTtcoWgQEAd	Since U Been Gone	Since U Been Gone	
8017	24LS4lQShWyixJ0ZrJXfJ5	Sweet Nothing (feat. Florence Welch)	Sweet Nothing	
10395	3hLuHKzG1cmlRpq53ZVWd8	The Good Ones	The Good Ones	
7749	725NSblej5lP3GfhLC7So3	115	115	
8224	4sebUbjqbcgDSwG6PbSGl0	Come a Little Closer	Come a Little Closer	
3530	6cKWDVak6o362TElLvwtmU	Suavecito Suavecito	Suavecito Suavecito	

10 rows × 32 columns

Looks gorgeous :)

Calculate how many countries the song is available in

In [10]:

```
# count the markets by taking everything between semicolons and making it a
market_count = df['available_markets'].str.split(';').str.len()

# Remove existing column if it exists
if 'market_count' in df.columns:
    df.drop(columns='market_count', inplace=True)

# Insert next to markets for readibility
df.insert(df.columns.get_loc('available_markets') + 1, 'market_count', marke
```

In [11]:

```
# Check dataset
df.sample(10, random_state=14)
```

Out[11]:

	track_id	track_name	clean_track_name	letters_track
443	7KA6U0WOHdGxWWLGPYN2Sb	On the Turning Away	On the Turning Away	
827	1Xf1lWBSml62NG1du3Ro14	Just The Way You Are	Just The Way You Are	
877	64IOxX7fXk89bMG2831w4G	Goodbye Time	Goodbye Time	
6849	4LloVtxNZpeh7q7xdi1DQc	Free Fallin' - Live at the Nokia Theatre, Los ...	Free Fallin'	
2083	00QAndVDVfNqNWYdWAhEan	Who Wants To Live Forever - Remastered 2011	Who Wants To Live Forever	
9823	4MXhiYIRDMGAuvZc5IFTwC	ASTROTHUNDER	ASTROTHUNDER	
7694	0Uybrtb766jul6WpkjqbID	Hard Times	Hard Times	
7203	76LGCP0g9nVknR7HD2Jjyp	Not The American Average	Not The American Average	
5387	5TpaWJKnuyA4MjzAbFXSTQ	Damn! (feat. Lil' Jon) - Club Mix	Damn! (feat. Lil' Jon)	
4454	4oPNN7syJYSjzDhRerF966	Untitled (How Does It Feel)	Untitled (How Does It Feel)	

10 rows × 33 columns

Count how many artists made/are on the track

In [12]:
```
df['artists_names'].sample(5, random_state=16)
```

Out[12]:
```
4930                                                  Trapt
5167                                                   Beck
6827      Angel Y Khriz;Gocho "El Lápiz De Platino";John...
10428     DJ Scheme;Ski Mask The Slump God;Danny Towers;...
322                                                    INXS
Name: artists_names, dtype: object
```

Looks like I can do the same thing I did for markets

In [13]:
```
df.columns.get_loc('artists_names')
```

Out[13]:
```
15
```

In [14]:
```
# count the artists by taking everything between semicolons and making it a
artist_numb = df['artists_names'].str.split(';').str.len()

# Remove existing column if it exists
if 'artist_numb' in df.columns:
```

```
    df.drop(columns='artist_numb', inplace=True)

# Insert next to markets for readibility
df.insert(df.columns.get_loc('artists_names') + 1, 'artist_numb', artist_num
```

In [15]:
```
df.iloc[:, 16:21].sample(5, random_state=16) #look at the added column to ch
```

Out[15]:

	artist_numb	principal_artist_name	artist_genres	principal_artist_followers	a
4930	1	Trapt	alternative metal;nu metal;post-grunge	1072711.0	
5167	1	Beck	alternative rock;anti-folk;permanent wave;rock	1466809.0	
6827	3	Angel Y Khriz	latin hip hop;reggaeton	1008284.0	
10428	4	DJ Scheme	viral rap	254790.0	
322	1	INXS	australian rock;dance rock;funk rock;new roman...	2457267.0	

Now calculate how many genres the main artist produces in

In [16]:
```
df['artist_genres'].sample(5, random_state=16)
```

Out[16]:
```
4930                alternative metal;nu metal;post-grunge
5167        alternative rock;anti-folk;permanent wave;rock
6827                               latin hip hop;reggaeton
10428                                            viral rap
322      australian rock;dance rock;funk rock;new roman...
Name: artist_genres, dtype: object
```

Again, looks like I can do the same thing I did for markets

In [17]:
```
df.columns.get_loc('artist_genres')
```

Out[17]: 18

In [18]:
```
# count the genres by taking everything between semicolons and making it a s
genres_numb = df['artist_genres'].str.split(';').str.len()

# Remove existing column if it exists
if 'genres_numb' in df.columns:
    df.drop(columns='genres_numb', inplace=True)
```

```
# Insert next to markets for readibility
df.insert(df.columns.get_loc('artist_genres') + 1, 'genres_numb', genres_num
```

In [19]:
```
df.iloc[:, 19:24].sample(5, random_state=30) #look at the added column to ch
```

Out[19]:

	genres_numb	principal_artist_followers	acousticness	danceability	energy
9624	2.0	50572176.0	0.0185	0.704	0.859
2879	5.0	683645.0	0.0287	0.754	0.785
398	8.0	632734.0	0.0139	0.927	0.832
11374	1.0	6957293.0	0.2800	0.415	0.573
8315	3.0	5646381.0	0.0737	0.343	0.536

Now let's look at the album release date and do some stuff with that

In [20]:
```
df['album_release_date'].sample(5, random_state=16)
```

Out[20]:
```
4930     2002-11-05 00:00:00
5167     2002-01-01 00:00:00
6827     2008-01-01 00:00:00
10428    2020-12-04 00:00:00
322      1987-01-01 00:00:00
Name: album_release_date, dtype: object
```

Let's figure out the month and day of the week they were released in. Right now we only have this format 1986-10-14 00:00:00.

In [21]:
```
df.columns.get_loc('album_release_date')
```

Out[21]:
```
12
```

In [22]:
```
# change column is datetime
df['album_release_date'] = pd.to_datetime(df['album_release_date'])

# Create new columns
df['release_month'] = df['album_release_date'].dt.month
df['release_weekday'] = df['album_release_date'].dt.weekday

#move columns to be next to the original album release date one
month_col = df.pop('release_month')
df.insert(df.columns.get_loc('album_release_date') + 1, 'release_month', mon

weekday_col = df.pop('release_weekday')
df.insert(df.columns.get_loc('album_release_date') + 2, 'release_weekday', w
```

In [23]:
```
df.iloc[:, 10:16].sample(5, random_state=30) #look at the added column to ch
```

Out[23]:

	track_number	album_name	album_release_date	release_month	release_
9624	1	Finesse (Remix) [feat. Cardi B]	2017-12-20	12	
2879	4	Labcabincalifornia (Deluxe Edition)	1995-01-01	1	
398	13	The Whispers: Greatest Hits	1987-01-01	1	
11374	36	One Thing At A Time	2023-03-03	3	
8315	8	PARTYNEXTDOOR	2013-07-01	7	

Okay cool. Lastly, release year is at the end of the data set right now but it kinda goes with these things so I'm gonna move it.

In [24]:

```
release_year = df.pop('year')
df.insert(df.columns.get_loc('album_release_date') + 3, 'year', release_year
```

In [25]:

```
df.iloc[:, 10:16].sample(5, random_state=30) #check again
```

Out[25]:

	track_number	album_name	album_release_date	release_month	release_
9624	1	Finesse (Remix) [feat. Cardi B]	2017-12-20	12	
2879	4	Labcabincalifornia (Deluxe Edition)	1995-01-01	1	
398	13	The Whispers: Greatest Hits	1987-01-01	1	
11374	36	One Thing At A Time	2023-03-03	3	
8315	8	PARTYNEXTDOOR	2013-07-01	7	

Okay, now I feel like our data set has some good information! Let's just check for N/As to clean it up completely.

In [26]:

```
df.isna().sum()
```

```
Out[26]: track_id                       0
track_name                     0
clean_track_name               0
letters_track_name             0
popularity                     0
available_markets              1
market_count                   1
disc_number                    0
duration_ms                    0
explicit                       0
track_number                   0
album_name                     0
album_release_date             0
release_month                  0
release_weekday                0
year                           0
album_type                     0
album_total_tracks             0
artists_names                  0
artist_numb                    0
principal_artist_name          0
artist_genres                105
genres_numb                  105
principal_artist_followers     0
acousticness                   5
danceability                   5
energy                         5
instrumentalness               5
key                            5
liveness                       5
loudness                       5
mode                           5
speechiness                    5
tempo                          5
time_signature                 5
valence                        5
duration_min                   0
dtype: int64
```

For genres I think I will just put the median because I don't want to drop all those rows but for the others I think I will drop them for ease

```
In [27]: print(df['genres_numb'].median())
df['genres_numb'] = df['genres_numb'].fillna(df['genres_numb'].median())
```

```
3.0
```

I am just gonna drop the rest now

```
In [28]: df = df.dropna()
```

Let's check

```
In [29]: df.isna().sum()
```

Out[29]:
```
track_id                     0
track_name                   0
clean_track_name             0
letters_track_name           0
popularity                   0
available_markets            0
market_count                 0
disc_number                  0
duration_ms                  0
explicit                     0
track_number                 0
album_name                   0
album_release_date           0
release_month                0
release_weekday              0
year                         0
album_type                   0
album_total_tracks           0
artists_names                0
artist_numb                  0
principal_artist_name        0
artist_genres                0
genres_numb                  0
principal_artist_followers   0
acousticness                 0
danceability                 0
energy                       0
instrumentalness             0
key                          0
liveness                     0
loudness                     0
mode                         0
speechiness                  0
tempo                        0
time_signature               0
valence                      0
duration_min                 0
dtype: int64
```

BEAUTIFUL!!!!

Run a Multiple Linear Regression to see which variables might affect song popularity!

In [30]:
```
print(df.columns)
```

```
Index(['track_id', 'track_name', 'clean_track_name', 'letters_track_name',
       'popularity', 'available_markets', 'market_count', 'disc_number',
       'duration_ms', 'explicit', 'track_number', 'album_name',
       'album_release_date', 'release_month', 'release_weekday', 'year',
       'album_type', 'album_total_tracks', 'artists_names', 'artist_numb',
       'principal_artist_name', 'artist_genres', 'genres_numb',
       'principal_artist_followers', 'acousticness', 'danceability', 'energ
y',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'time_signature', 'valence', 'duration_min'],
      dtype='object')
```

All of my columns need to be numeric variables for this to work

In [31]:
```
df.dtypes
```

Out[31]:
```
track_id                              object
track_name                            object
clean_track_name                      object
letters_track_name                     int64
popularity                             int64
available_markets                     object
market_count                         float64
disc_number                            int64
duration_ms                            int64
explicit                                bool
track_number                           int64
album_name                            object
album_release_date            datetime64[ns]
release_month                          int64
release_weekday                        int64
year                                   int64
album_type                            object
album_total_tracks                     int64
artists_names                         object
artist_numb                            int64
principal_artist_name                 object
artist_genres                         object
genres_numb                          float64
principal_artist_followers           float64
acousticness                         float64
danceability                         float64
energy                               float64
instrumentalness                     float64
key                                  float64
liveness                             float64
loudness                             float64
mode                                 float64
speechiness                          float64
tempo                                float64
time_signature                       float64
valence                              float64
duration_min                         float64
dtype: object
```

In [32]:
```
irrelevant = ['track_id', 'track_name', 'clean_track_name', 'available_marke

df_regression = df.drop(columns=irrelevant) #drop them

#convert my categorical columns to numeric
df_regression['explicit'] = df_regression['explicit'].astype(int)
df_regression['album_type'] = df_regression['album_type'].astype('category')

pred = df_regression.drop(columns=['popularity']) #drop target and keep all
y = df_regression['popularity'] #target
```

In [33]:
```
#check that my "numeric" columns actually are
pred.dtypes.head(50)
```

Out[33]:
```
letters_track_name                int64
market_count                    float64
disc_number                       int64
duration_ms                       int64
explicit                          int64
track_number                      int64
release_month                     int64
release_weekday                   int64
year                              int64
album_type                         int8
album_total_tracks                int64
artist_numb                       int64
genres_numb                     float64
principal_artist_followers      float64
acousticness                    float64
danceability                    float64
energy                          float64
instrumentalness                float64
key                             float64
liveness                        float64
loudness                        float64
mode                            float64
speechiness                     float64
tempo                           float64
time_signature                  float64
valence                         float64
duration_min                    float64
dtype: object
```

BEAUTIFUL!!!!

Now I get to run my model FINALLY

In [34]:
```
#one package I could use
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(pred, y)
```

Out[34]:

▾ LinearRegression

```
LinearRegression()
```

In [35]:

```
import statsmodels.api as sm

# Add a constant column for the intercept
pred_sm = sm.add_constant(pred)

# Fit the OLS regression
model = sm.OLS(y, pred_sm).fit()

# Print the full summary
print(model.summary2())
```

```
                         Results: Ordinary least squares
============================================================================
=====
Model:                    OLS                  Adj. R-squared:         0.506
Dependent Variable:       popularity           AIC:                    6665
9.1416
Date:                     2025-12-10 13:51     BIC:                    6685
4.1402
No. Observations:         10119                Log-Likelihood:         -3330
3.
Df Model:                 26                   F-statistic:            398.9
Df Residuals:             10092                Prob (F-statistic):     0.00
R-squared:                0.507                Scale:                  42.39
3
----------------------------------------------------------------------------
-----
                             Coef.   Std.Err.    t     P>|t|    [0.025    0.
975]
----------------------------------------------------------------------------
-----
const                      -875.2074  16.7820 -52.1515 0.0000 -908.1035 -84
2.3113
letters_track_name           -0.0404   0.0104  -3.8753 0.0001   -0.0609   -
0.0200
market_count                  0.0305   0.0011  26.5829 0.0000    0.0282
0.0327
disc_number                  -0.1517   0.4035  -0.3760 0.7069   -0.9427
0.6392
duration_ms                   0.0000   0.0000   0.9626 0.3358   -0.0000
0.0000
explicit                     -0.1675   0.1895  -0.8838 0.3768   -0.5390
0.2040
track_number                 -0.0786   0.0158  -4.9835 0.0000   -0.1095   -
0.0477
release_month                 0.1007   0.0183   5.5097 0.0000    0.0649
0.1366
release_weekday               0.2183   0.0379   5.7628 0.0000    0.1441
0.2926
year                          0.4687   0.0083  56.7773 0.0000    0.4525
0.4849
album_type                    0.4337   0.1281   3.3868 0.0007    0.1827
0.6847
album_total_tracks           -0.0015   0.0084  -0.1766 0.8598   -0.0179
0.0149
artist_numb                   0.1398   0.0876   1.5962 0.1105   -0.0319
0.3115
genres_numb                  -0.0818   0.0355  -2.3056 0.0212   -0.1514   -
0.0123
principal_artist_followers    0.0000   0.0000  20.0420 0.0000    0.0000
0.0000
acousticness                  0.7609   0.3353   2.2691 0.0233    0.1036
1.4183
danceability                  2.1622   0.5200   4.1581 0.0000    1.1429
3.1814
energy                        0.8040   0.5807   1.3845 0.1662   -0.3343
1.9423
```

```
instrumentalness                 0.1241    0.4307    0.2881 0.7733    -0.7202
0.9684
key                             -0.0245    0.0184   -1.3296 0.1837    -0.0605
0.0116
liveness                        -1.3275    0.4552   -2.9165 0.0035    -2.2197   -
0.4353
loudness                         0.1274    0.0289    4.4123 0.0000     0.0708
0.1840
mode                            -0.7832    0.1426   -5.4939 0.0000    -1.0626   -
0.5037
speechiness                     -6.6990    0.8265   -8.1051 0.0000    -8.3192   -
5.0789
tempo                           -0.0040    0.0022   -1.7707 0.0766    -0.0083
0.0004
time_signature                  -0.1837    0.1863   -0.9863 0.3240    -0.5488
0.1814
valence                         -0.3764    0.3482   -1.0810 0.2797    -1.0590
0.3062
duration_min                     0.0000    0.0000    1.2872 0.1981    -0.0000
0.0000
----------------------------------------------------------------------------
-----
Omnibus:                   391.996        Durbin-Watson:           0.829
Prob(Omnibus):             0.000          Jarque-Bera (JB):        438.736
Skew:                      0.510          Prob(JB):                0.000
Kurtosis:                  3.032          Condition No.:           113323675328
00962
============================================================================
=====
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly
specified.
[2] The smallest eigenvalue is 3.24e-14. This might indicate that
there are strong multicollinearity problems or that the design
matrix is singular.
```

In [36]:

```
#look at it all without it truncating
summary_df = pd.DataFrame({
    'coef': model.params,
    'std_err': model.bse,
    't': model.tvalues,
    'p_value': model.pvalues
})

pd.set_option('display.float_format', '{:.6f}'.format) #sets it to not go ir
print(summary_df.to_string())
```

	coef	std_err	t	p_value
const	-875.207385	16.782011	-52.151521	0.000000
letters_track_name	-0.040449	0.010438	-3.875262	0.000107
market_count	0.030465	0.001146	26.582909	0.000000
disc_number	-0.151736	0.403504	-0.376047	0.706890
duration_ms	0.000001	0.000001	0.962634	0.335754
explicit	-0.167513	0.189534	-0.883818	0.376816
track_number	-0.078553	0.015763	-4.983506	0.000001
release_month	0.100728	0.018282	5.509739	0.000000
release_weekday	0.218341	0.037888	5.762796	0.000000
year	0.468707	0.008255	56.777259	0.000000
album_type	0.433719	0.128062	3.386797	0.000710
album_total_tracks	-0.001480	0.008378	-0.176633	0.859800
artist_numb	0.139821	0.087593	1.596248	0.110465
genres_numb	-0.081814	0.035486	-2.305554	0.021156
principal_artist_followers	0.000000	0.000000	20.041984	0.000000
acousticness	0.760938	0.335347	2.269103	0.023283
danceability	2.162157	0.519986	4.158102	0.000032
energy	0.803984	0.580719	1.384463	0.166247
instrumentalness	0.124082	0.430733	0.288072	0.773297
key	-0.024465	0.018401	-1.329601	0.183680
liveness	-1.327505	0.455177	-2.916458	0.003548
loudness	0.127407	0.028875	4.412330	0.000010
mode	-0.783161	0.142552	-5.493875	0.000000
speechiness	-6.699014	0.826517	-8.105109	0.000000
tempo	-0.003962	0.002237	-1.770659	0.076648
time_signature	-0.183711	0.186267	-0.986281	0.324019
valence	-0.376430	0.348239	-1.080954	0.279743
duration_min	0.000000	0.000000	1.287185	0.198059

Oh wow so quite a few of them are significant predictors and my R squared is decently high. Seems promising to me!

List of variables with a p value less than 0.05 -- statistically significant contributors!!!!

- letters_track_name
- market_count
- track_number
- release_month
- release_weekday
- year
- album_type
- genres_numb
- principal_artist_followers
- acousticness

- danceability
- liveness
- loudness
- mode
- speechiness

Let's check for multicollinearity just to make sure

In [37]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# add constant/beta 0/intercept added
pred_vif = sm.add_constant(pred)

# Compute VIF for each column
vif_df = pd.DataFrame({
    "feature": pred_vif.columns,
    "VIF": [variance_inflation_factor(pred_vif.values, i) #values convert to
            for i in range(pred_vif.shape[1])] #shape takes all the columns
}) #so basically go through all calumns and calculate the variance

print(vif_df)
```

	feature	VIF
0	const	67225.254413
1	letters_track_name	1.034509
2	market_count	1.050163
3	disc_number	1.256947
4	duration_ms	inf
5	explicit	1.609970
6	track_number	1.166390
7	release_month	1.138891
8	release_weekday	1.142766
9	year	1.953558
10	album_type	1.242710
11	album_total_tracks	1.472709
12	artist_numb	1.088994
13	genres_numb	1.201645
14	principal_artist_followers	1.208237
15	acousticness	1.837259
16	danceability	1.703249
17	energy	3.412144
18	instrumentalness	1.211604
19	key	1.020806
20	liveness	1.055523
21	loudness	2.832940
22	mode	1.057183
23	speechiness	1.448800
24	tempo	1.089438
25	time_signature	1.066449
26	valence	1.746087
27	duration_min	inf

Looks pretty good to me!

Some explanations about the VIF stuff

- What is VIF?
  - Variance Inflation Factor (VIF) is a statistical measure used in regression analysis to identify multicollinearity
  - A VIF of 1 means no correlation, while a VIF exceeding 10 can signal serious multicollinearity
  - A model with a low VIF is more stable because its predictors are less influenced by each other
- Infinite VIF
  - duration_ms: VIF = infinity !!!
  - duration_ms is perfectly collinear with one or more other predictors or it has near-zero variance
    - this make sense because duration_ms and duration_min are the same numbers but converted unit-wise
- Most other predictors with VIF < 5 are Totally acceptable to use in our simulation! - - We know they are not colinear
  - liveness

- mode
- speechiness
- tempo
- time_signature

Next Steps

Possible Ideas (won't do all of them)

- for each predictor, find the range of values so we know what boundaries to simulate within
- simulate a whole bunch of songs and see trends, what makes something more popular
    - take like top 10% or something and see what they have in common
    - graphs of course
    - maybe map two features against eachother
    - maybe we could add a random luck factor for going viral
        - likelihoood it goes viral on tiktok or it gets put in a show/movie or some things like that
- maybe have another simulation where you input your own song and it generates a popularity score
    - then you can change one variable and see how it affects the score
    - could make a distribution for teh same song but from 1986 to 2023 to see the change over time
- could do an agent where there are certain probabilities that someone shares a song

Start OOP

Goal: start by simulating the features of one song, then simulate the popularity of that one song, then we can run that simulation a whole bunch of times and start making some visulaizations

Maybe let's start by finding the ranges/probabilities of the data we already have so that we can simulate new songs that are realistic

In [38]:
```
print(pred.columns)
```

```
Index(['letters_track_name', 'market_count', 'disc_number', 'duration_ms',
       'explicit', 'track_number', 'release_month', 'release_weekday', 'yea
r',
       'album_type', 'album_total_tracks', 'artist_numb', 'genres_numb',
       'principal_artist_followers', 'acousticness', 'danceability', 'energ
y',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'time_signature', 'valence', 'duration_min'],
      dtype='object')
```

In [39]:

```
continuous_cols = ['letters_track_name', 'market_count', 'disc_number', 'dur
       'explicit', 'track_number', 'release_month', 'release_weekday', 'year
       'album_type', 'album_total_tracks', 'artist_numb', 'genres_numb',
       'principal_artist_followers', 'acousticness', 'danceability', 'energy
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'time_signature', 'valence', 'duration_min']
cont_min = pred[continuous_cols].min()
cont_max = pred[continuous_cols].max()
print(cont_max)
print(cont_min)
```

```
letters_track_name                 76.000000
market_count                      184.000000
disc_number                        10.000000
duration_ms                   2238733.000000
explicit                            1.000000
track_number                       48.000000
release_month                      12.000000
release_weekday                     6.000000
year                             2023.000000
album_type                          2.000000
album_total_tracks                176.000000
artist_numb                        40.000000
genres_numb                        15.000000
principal_artist_followers  114675033.000000
acousticness                        0.996000
danceability                        0.988000
energy                              1.000000
instrumentalness                    1.000000
key                                11.000000
liveness                            0.982000
loudness                            0.522000
mode                                1.000000
speechiness                         0.944000
tempo                             220.099000
time_signature                      5.000000
valence                             0.994000
duration_min                       37.312217
dtype: float64
letters_track_name                0.000000
market_count                      1.000000
disc_number                       1.000000
duration_ms                   33493.000000
explicit                          0.000000
track_number                      1.000000
release_month                     1.000000
release_weekday                   0.000000
year                           1986.000000
album_type                        0.000000
album_total_tracks                1.000000
artist_numb                       1.000000
genres_numb                       1.000000
principal_artist_followers      100.000000
acousticness                      0.000000
danceability                      0.000000
energy                            0.000020
instrumentalness                  0.000000
key                               0.000000
liveness                          0.000000
loudness                        -47.070000
mode                              0.000000
speechiness                       0.000000
tempo                             0.000000
time_signature                    0.000000
valence                           0.000000
duration_min                      0.558217
dtype: float64
```

In [40]:

```python
## prediction function
def Predict_Popularity(predictor_list, model):
    """
    Predicts track popularity using a list of predictor values
    corresponding to the significant predictors in the reduced model.

    Parameters
    ----------
    predictor_list : list
        A list of values in the correct order of the significant predictors.
    model : fitted statsmodels OLS model

    Returns
    -------
    float : predicted popularity score
    """

    # Significant predictors in correct order
    predictor_names = [
        "letters_track_name",
        "market_count",
        "track_number",
        "release_month",
        "release_weekday",
        "year",
        "album_type",
        "genres_numb",
        "principal_artist_followers",
        "acousticness",
        "danceability",
        "liveness",
        "loudness",
        "mode",
        "speechiness"
    ]

    # Build input dataframe
    input_data = pd.DataFrame([dict(zip(predictor_names, predictor_list))])

    # Predict
    prediction = model.predict(input_data)

    return float(prediction.iloc[0])
```

In [41]:

```python
significant_predictors = [
    "letters_track_name",
    "market_count",
    "track_number",
    "release_month",
    "release_weekday",
    "year",
    "genres_numb",
    "principal_artist_followers",
    "acousticness",
    "danceability",
```

```
    "liveness",
    "loudness",
    "mode",
    "speechiness"
]
```

I decided to redo the cleaning / OLS model because I wanted to reduce the amount of explanatory variables we were using

I only used the variables that were deemed significant from the earlier OLS model!

In [42]:
```
df_clean = df[significant_predictors].copy()

# Convert numeric columns cleanly
df_clean = df_clean.apply(pd.to_numeric, errors="coerce")

# Remove any NaN rows
df_clean = df_clean.dropna()

# Align y
y_clean = df.loc[df_clean.index, "popularity"]

# Fit reduced model
X_reduced = sm.add_constant(df_clean)
model_reduced = sm.OLS(y_clean, X_reduced).fit()

print(model_reduced.summary())
```

```
                            OLS Regression Results
============================================================================
==
Dep. Variable:             popularity   R-squared:                       0.5
06
Model:                            OLS   Adj. R-squared:                  0.5
05
Method:                 Least Squares   F-statistic:                     73
7.8
Date:                Wed, 10 Dec 2025   Prob (F-statistic):               0.
00
Time:                        13:52:01   Log-Likelihood:                -3331
6.
No. Observations:               10119   AIC:                          6.666e+
04
Df Residuals:                   10104   BIC:                          6.677e+
04
Df Model:                          14
Covariance Type:            nonrobust
============================================================================
==================
                                 coef    std err          t      P>|t|
[0.025      0.975]
----------------------------------------------------------------------------
------------------
const                       -885.2613     14.748    -60.027      0.000    -9
14.170    -856.353
letters_track_name            -0.0381      0.010     -3.669      0.000
-0.058      -0.018
market_count                   0.0305      0.001     26.700      0.000
0.028       0.033
track_number                  -0.0908      0.015     -6.102      0.000
-0.120      -0.062
release_month                  0.1017      0.018      5.569      0.000
0.066       0.137
release_weekday                0.2143      0.038      5.662      0.000
0.140       0.289
year                           0.4735      0.007     64.512      0.000
0.459       0.488
genres_numb                   -0.0851      0.035     -2.422      0.015
-0.154      -0.016
principal_artist_followers  7.894e-08    3.9e-09     20.222      0.000     7.
13e-08    8.66e-08
acousticness                   0.6054      0.293      2.064      0.039
0.030       1.180
danceability                   1.8533      0.418      4.429      0.000
1.033       2.674
liveness                      -1.2798      0.452     -2.832      0.005
-2.166      -0.394
loudness                       0.1337      0.021      6.330      0.000
0.092       0.175
mode                          -0.7934      0.141     -5.628      0.000
-1.070      -0.517
speechiness                   -6.8152      0.734     -9.287      0.000
-8.254      -5.377
============================================================================
```

```
==
Omnibus:                      394.854   Durbin-Watson:                   0.8
34
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              442.3
14
Skew:                           0.512   Prob(JB):                     8.97e-
97
Kurtosis:                       3.030   Cond. No.                     4.62e+
09
==============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
[2] The condition number is large, 4.62e+09. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

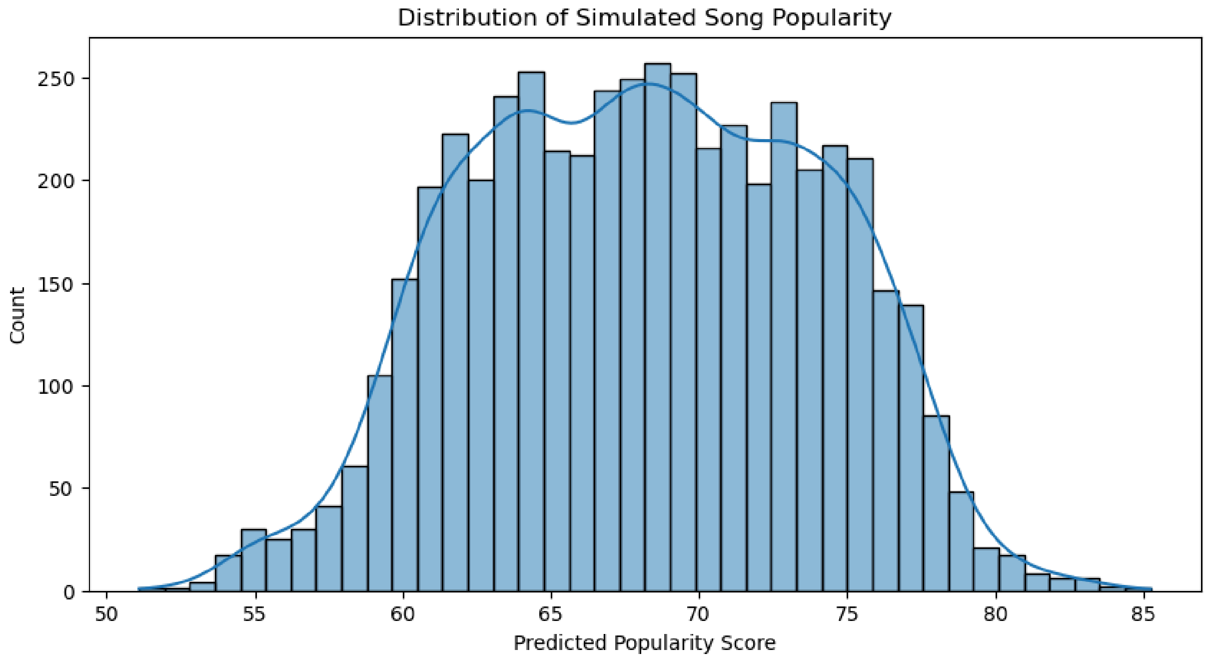
In [43]:
```python
def simulate_song_features(df, predictors):
    """
    Randomly generate one simulated song by sampling from the empirical
    distributions of each predictor.
    """
    sim = {}

    for col in predictors:
        sim[col] = np.random.choice(df[col].values)

    return sim
```

In [44]:
```python
def predict_popularity(sim_features, model):
    """
    Takes a simulated dict of features and returns predicted popularity.
    """
    df_input = pd.DataFrame([sim_features])
    df_input = sm.add_constant(df_input, has_constant="add")
    return model.predict(df_input)[0]
```

In [45]:
```python
def simulate_one_song(df, predictors, model):
    song = simulate_song_features(df, predictors)
    popularity = predict_popularity(song, model)
    return song, popularity
```

In [46]:
```python
sim_results = []

N = 5000  # number of simulated songs

for i in range(N):
    song, popularity = simulate_one_song(df_clean, significant_predictors, m
    song["predicted_popularity"] = popularity
    sim_results.append(song)

sim_df = pd.DataFrame(sim_results)
```

In [47]:
```
plt.figure(figsize=(10,5))
sns.histplot(sim_df["predicted_popularity"], bins=40, kde=True)
plt.title("Distribution of Simulated Song Popularity")
plt.xlabel("Predicted Popularity Score")
plt.ylabel("Count")
plt.show()
```

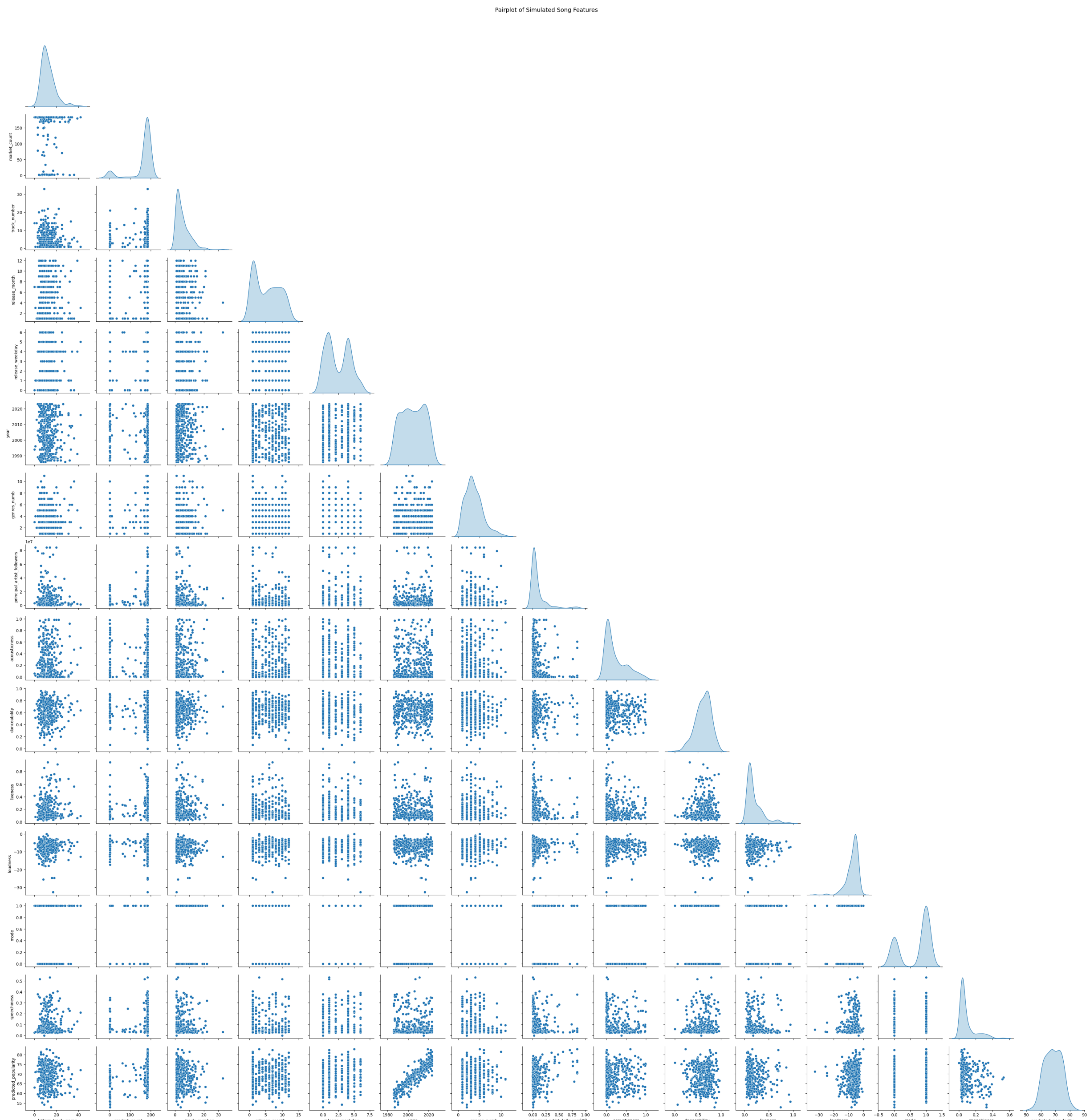


I took a data visualizations course and wanted to see an interactive graph of the variabes in comparison!

First let's see how danceability affects popularity

In [48]:
```
import altair as alt
```

In [49]:
```
sim_sample = sim_df.sample(400)

sns.pairplot(sim_sample, diag_kind="kde", corner=True)
plt.suptitle("Pairplot of Simulated Song Features", y=1.02, fontsize=14)
plt.show()
```



Which variables might be correlated?

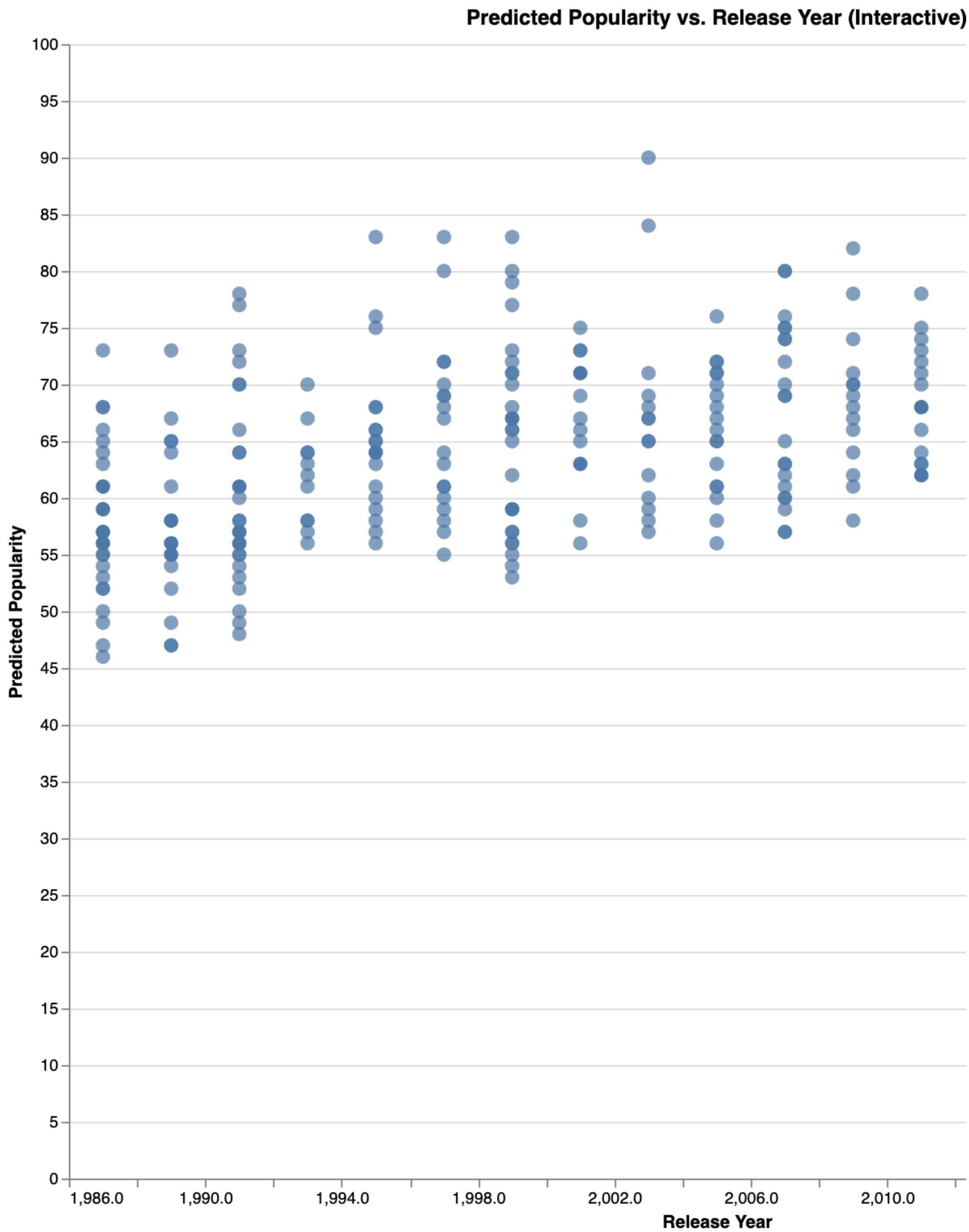
Using a pairplot we can easily see what variables have a linear relationship with the predicted popularity.

The clearest linear plot I see is with the year -- I created a zoomed in scatter plot of **popularity vs year** below using altair.

In [50]:
```
df_sample = df.sample(400)
alt.data_transformers.disable_max_rows()

# Brush for zoom + pan
brush = alt.selection_interval(bind="scales")


# Create interactive scatterplot
```

```python
chart = (
    alt.Chart(df_sample)
    .mark_circle(size=80)
    .encode(
        x=alt.X("year:Q", bin=alt.Bin(step=2), title="Release Year"),
        y=alt.Y("popularity:Q", title="Predicted Popularity"),
        tooltip=[
            "track_name:N",
            "year:Q",
            "popularity:Q",
            "danceability:Q",
            "acousticness:Q"
        ]
    )
    .add_params( brush)
    .properties(
        width=800,
        height=700,
        title="Predicted Popularity vs. Release Year (Interactive)"
    )
)

chart
```

Out[50]:



popularity vs followers

above you can see I wanted to see how followers affect popularity, there is not much correlation

In [51]:

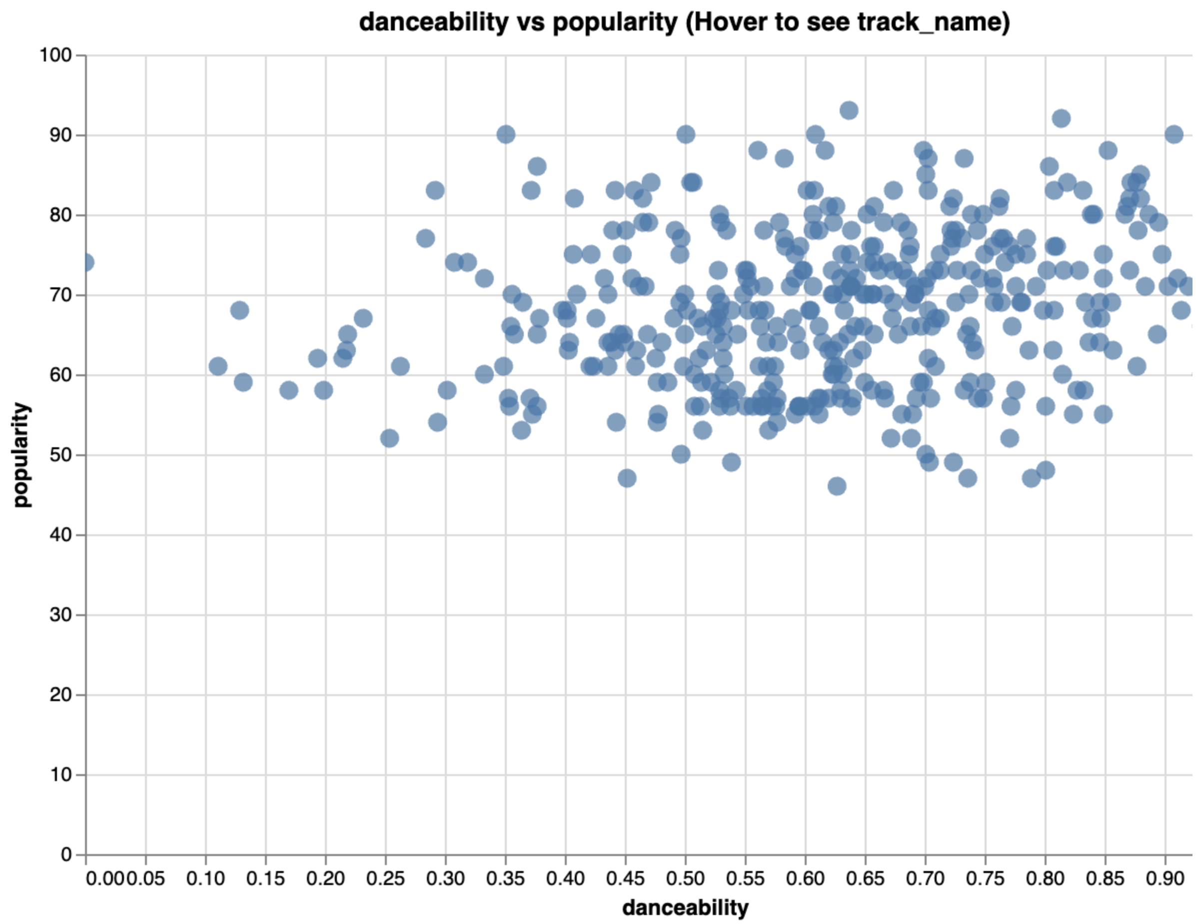
```
def interactive_scatter(df, x_var, y_var, hover_var="track_name"):
    """
    Creates an interactive Altair scatterplot comparing two variables,
```

```
    showing song name (or any hover variable) on hover.

    Parameters
    ----------
    df : DataFrame
        Your cleaned dataframe.
    x_var : str
        Column name for x-axis.
    y_var : str
        Column name for y-axis.
    hover_var : str
        Variable to show when hovering (default = track_name).
    """

    chart = (
        alt.Chart(df)
        .mark_circle(size=90)
        .encode(
            x=alt.X(x_var, title=x_var),
            y=alt.Y(y_var, title=y_var),
            tooltip=[hover_var, x_var, y_var]
        )
        .interactive()
        .properties(
            width=600,
            height=400,
            title=f"{x_var} vs {y_var} (Hover to see {hover_var})"
        )
    )

    return chart
```

In [52]:

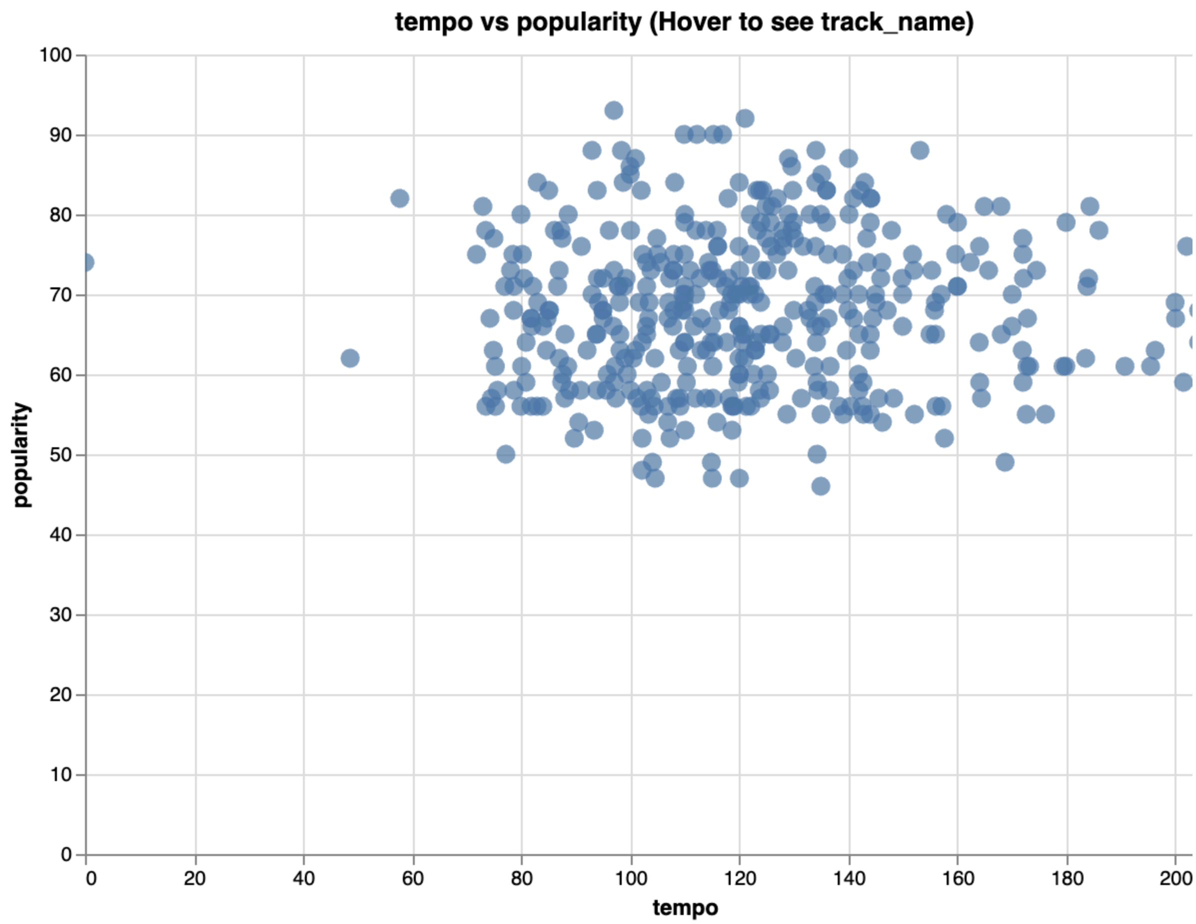
```
interactive_scatter(df_sample, "danceability", "popularity", hover_var="trac
```

Out[52]:



In [53]:
```
interactive_scatter(df_sample, "tempo", "popularity", hover_var="track_name"
```

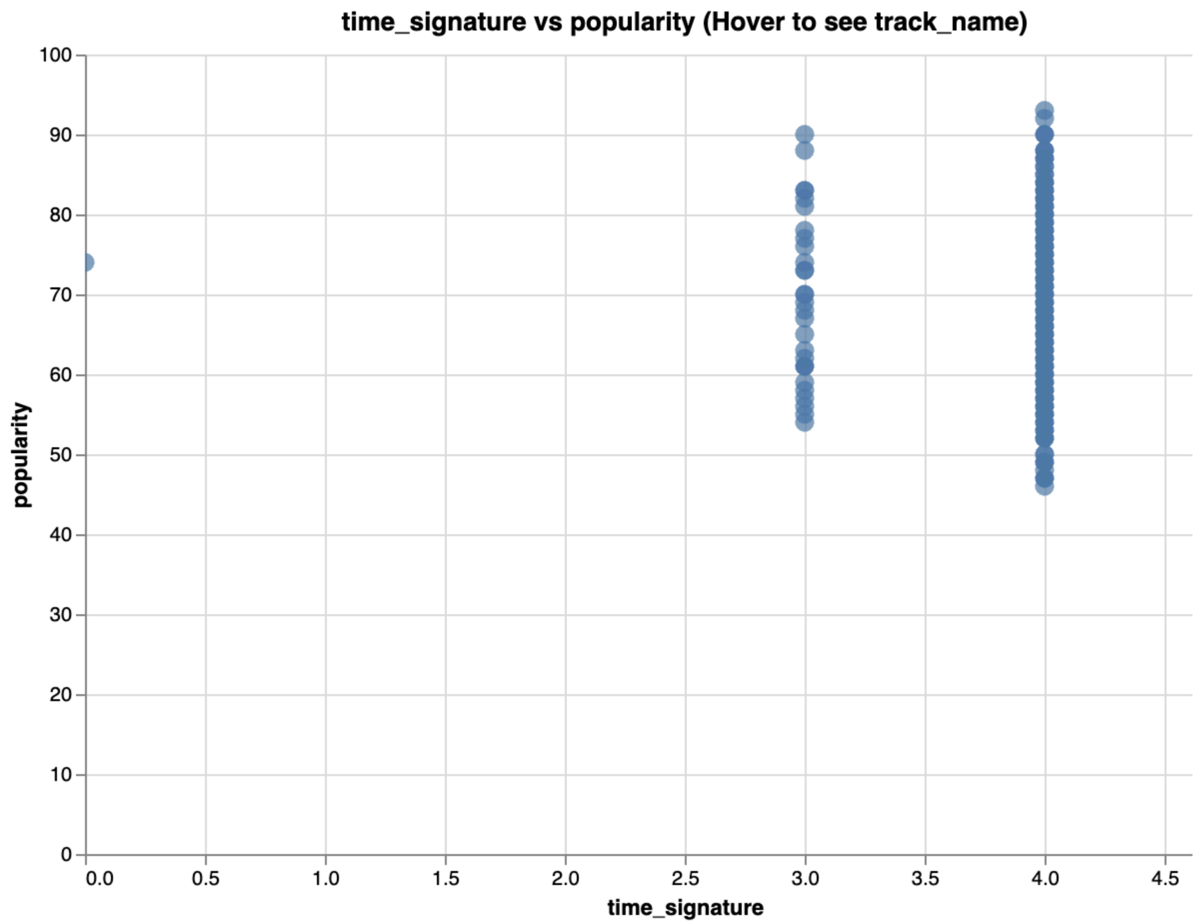
Out[53]:



In [54]:

```
interactive_scatter(df_sample, "time_signature", "popularity", hover_var="tr
```

Out[54]:



Some key outliers are "brown noise" and "Box fan sound" which do not have any time signature or tempo, and therefore have no danceability. I also would argue that some songs such as "Always forever" by Cults are not ranked the best in terms of danceability, I think I could dance to that song!

Another Simulations with Added Noise and Virality Variable + New Way of Analyzing Current Variables

In [55]:

```
df['album_type'] = df['album_type'].astype('category').cat.codes.astype('int
```

In [56]:

```
predictor_cols = [
    'letters_track_name', 'market_count', 'disc_number', 'duration_ms',
    'explicit', 'track_number', 'release_month', 'release_weekday', 'year',
    'album_type', 'album_total_tracks', 'artist_numb', 'genres_numb',
    'principal_artist_followers', 'acousticness', 'danceability', 'energy',
    'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
    'speechiness', 'tempo', 'time_signature', 'valence', 'duration_min'
]

DEFAULT_N_SIMS = 10000
TOP_PCT = 0.10  # top 10%
```

```python
RANDOM_SEED = 42

# Viral parameters
VIRAL_ON = True
VIRAL_PROB = 0.01  # 1% chance
VIRAL_BOOST_MEAN = 20.0
VIRAL_BOOST_STD = 5.0

# Noise settings (per-column multiplier of observed std; can be scalar or di
NOISE_SCALE = 1.0  # will multiply each column's std by this for Gaussian no


# Helper: compute ranges & stats
#Returns a DataFrame with min, max, mean, std, and dtype for each predictor
def get_predictor_stats(df, cols):
    stats = []
    for c in cols:
        s = df[c]
        stats.append({
            'col': c,
            'min': s.min(),
            'max': s.max(),
            'mean': s.mean(),
            'std': s.std(ddof=0),  # population std to match simulation scal
            'dtype': s.dtype
        })
    return pd.DataFrame(stats).set_index('col')


#Sample continuous variable using normal around mean, switch to uniform if s
def sample_continuous(col_stats, n):
    mean, std, mn, mx = col_stats['mean'], col_stats['std'], col_stats['min'
    if pd.isna(std) or std == 0:
        return np.random.uniform(mn, mx, size=n)
    else:
        samp = np.random.normal(loc=mean, scale=std * NOISE_SCALE, size=n)
        # clip to observed min/max
        return np.clip(samp, mn, mx)

#Sample integer-like columns by sampling uniform in [min, max] and rounding.
def sample_integer(col_stats, n):
    mn, mx = int(col_stats['min']), int(col_stats['max'])
    if mn == mx:
        return np.full(n, mn, dtype=int)
    # sample integers uniformly
    return np.random.randint(mn, mx + 1, size=n)

#for things like acousticness/danceability in [0,1] - sample Beta-like behav
def sample_bounded_ratio(col_stats, n):

    mn, mx, mean, std = col_stats['min'], col_stats['max'], col_stats['mean'
    if std == 0 or pd.isna(std):
        return np.clip(np.full(n, mean), mn, mx)
    s = np.random.normal(loc=mean, scale=std * NOISE_SCALE, size=n)
    return np.clip(s, mn, mx)
```

Similar Simulation

In [57]:
```
def simulate_songs(df, model, predictor_cols, n_sims=DEFAULT_N_SIMS, seed=RA
                   viral_on=VIRAL_ON, viral_prob=VIRAL_PROB,
                   viral_boost_mean=VIRAL_BOOST_MEAN, viral_boost_std=VIRAL_
    """
    Simulate synthetic songs and predict popularity using statsmodels `model
    Returns the simulated DataFrame with `predicted_popularity` and `predict
    """
    np.random.seed(seed)
    stats = get_predictor_stats(df, predictor_cols)
    sim = pd.DataFrame(index=range(n_sims))

    for col in predictor_cols:
        cs = stats.loc[col]
        dtype = cs['dtype']
        # Identify how to sample:
        # - many features are bounded 0..1 (acousticness, danceability, etc.
        # - some are integer flags/counts (explicit, disc_number, track_numb
        # - duration_ms, tempo, loudness continuous
        if dtype in (np.int8, np.int16, np.int32, np.int64) or col in [
                'disc_number', 'explicit', 'track_number', 'release_month',
                'album_type', 'album_total_tracks', 'artist_numb', 'key', 't
            ]:
            # integer-like sampling
            sim[col] = sample_integer(cs, n_sims)
        else:
            # float columns - decide if bounded ratio or general continuous
            if col in ['acousticness', 'danceability', 'energy', 'instrument
                sim[col] = sample_bounded_ratio(cs, n_sims)
            elif col in ['key', 'time_signature']:
                sim[col] = sample_integer(cs, n_sims)
            else:
                sim[col] = sample_continuous(cs, n_sims)

    # Ensure types mirror original where appropriate
    for col in predictor_cols:
        if df[col].dtype in (np.int8, np.int16, np.int32, np.int64):
            sim[col] = sim[col].round().astype(int)

    # Prepare dataframe for model prediction. statsmodels expects same colum
    # If model was fit with a constant (safer to add), use sm.add_constant
    sim_for_pred = sim.copy()
    sim_for_pred = sm.add_constant(sim_for_pred, has_constant='add')

    # Use model.predict
    try:
        sim['predicted_popularity'] = model.predict(sim_for_pred)
    except Exception as e:
        raise RuntimeError("Failed to predict with provided model. Make sure

    # Optionally add viral luck factor
    sim['predicted_popularity_with_viral'] = sim['predicted_popularity'].cop
    if viral_on:
```

```
        # determine viral events
        viral_events = np.random.rand(n_sims) < viral_prob
        viral_boosts = np.random.normal(loc=viral_boost_mean, scale=viral_bo
        viral_boosts = np.where(viral_events, viral_boosts, 0.0)
        sim['viral_boost'] = viral_boosts
        sim['predicted_popularity_with_viral'] += sim['viral_boost']
    else:
        sim['viral_boost'] = 0.0

    # Clip popularity to realistic bounds if you prefer (e.g., 0-100)
    sim['predicted_popularity'] = sim['predicted_popularity'].clip(lower=0,
    sim['predicted_popularity_with_viral'] = sim['predicted_popularity_with_

    return sim, stats


# --- Analysis functions ------------------------------------------------------
def analyze_top(sim, pct=TOP_PCT, target_col='predicted_popularity_with_vira
    """
    Return top fraction of rows and summary statistics comparing top vs rest
    """
    n_top = int(len(sim) * pct)
    top = sim.nlargest(n_top, target_col)
    rest = sim.drop(top.index)

    summary = pd.DataFrame({
        'top_mean': top[predictor_cols].mean(),
        'rest_mean': rest[predictor_cols].mean(),
        'top_median': top[predictor_cols].median(),
        'rest_median': rest[predictor_cols].median(),
    })
    summary['mean_diff'] = summary['top_mean'] - summary['rest_mean']
    return top, rest, summary.sort_values('mean_diff', ascending=False)
```

Plot Functions!

In [58]:

```
# --- Plotting ----------------------------------------------------------------
def plot_histogram(sim, col='predicted_popularity_with_viral', bins=50, save
    plt.figure(figsize=(8,5))
    plt.hist(sim[col], bins=bins)
    plt.title(f'Histogram of {col}')
    plt.xlabel('Predicted popularity')
    plt.ylabel('Count')
    if savepath:
        plt.savefig(savepath, bbox_inches='tight')
    plt.show()


def plot_scatter(sim, x, y, target_col='predicted_popularity_with_viral', hu
    plt.figure(figsize=(7,6))
    # color by popularity if desired
    sc = plt.scatter(sim[x], sim[y], c=sim[target_col], alpha=0.6)
    plt.colorbar(sc, label=target_col)
    plt.xlabel(x)
    plt.ylabel(y)
```

```
    plt.title(f'{x} vs {y} colored by {target_col}')
    if savepath:
        plt.savefig(savepath, bbox_inches='tight')
    plt.show()


def plot_correlation_heatmap(df_sim, cols=None, savepath=None):
    if cols is None:
        cols = predictor_cols + ['predicted_popularity_with_viral']
    corr = df_sim[cols].corr()
    plt.figure(figsize=(12,10))
    sns.heatmap(corr, annot=False, cmap='vlag', center=0)
    plt.title('Correlation matrix (simulated data)')
    if savepath:
        plt.savefig(savepath, bbox_inches='tight')
    plt.show()


def plot_top_vs_rest_feature(sim, feature, top_idx=None, pct=TOP_PCT, savepa
    top, rest, _ = analyze_top(sim, pct=pct)
    plt.figure(figsize=(8,5))
    sns.kdeplot(rest[feature], label='rest', fill=True)
    sns.kdeplot(top[feature], label='top', fill=True)
    plt.title(f'Distribution of {feature} - top {int(pct*100)}% vs rest')
    plt.legend()
    if savepath:
        plt.savefig(savepath, bbox_inches='tight')
    plt.show()
```

Run the Simulation

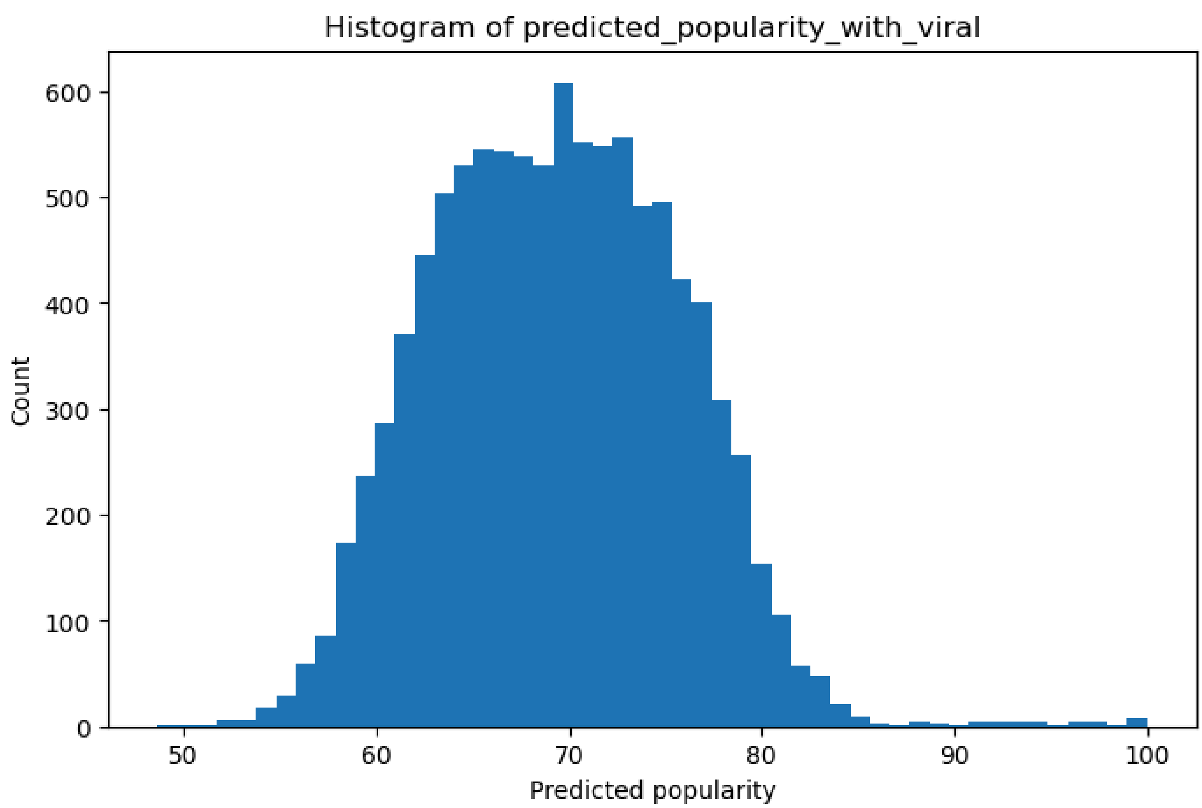
In [59]:

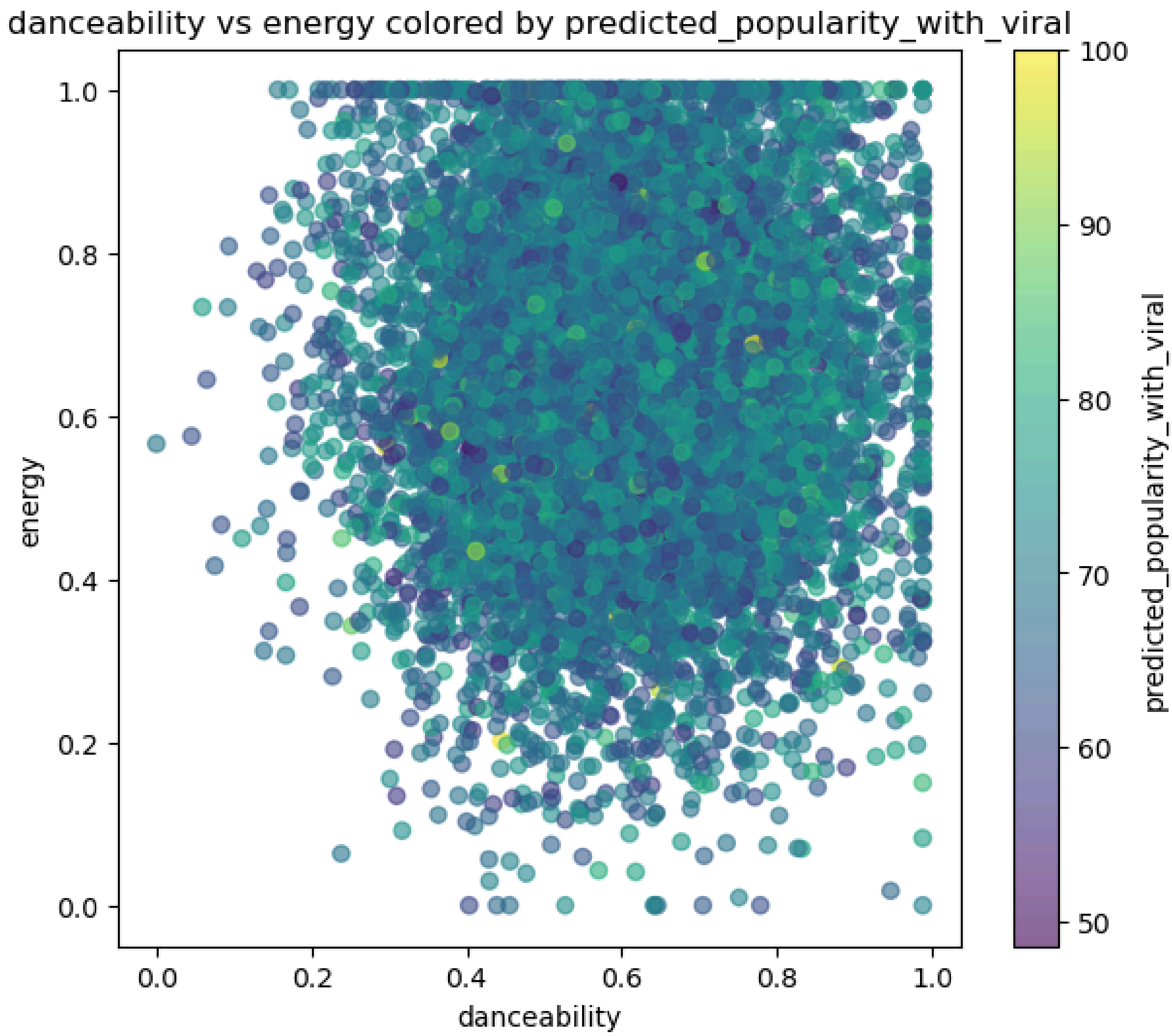
```
if __name__ == "__main__":
    # Run simulation (adjust n_sims as needed)
    sim_df, stats_df = simulate_songs(df=df, model=model, predictor_cols=pre
    print("Simulated shape:", sim_df.shape)
    print("Predictor stats:\n", stats_df)

    # Basic analysis
    top, rest, summary = analyze_top(sim_df, pct=TOP_PCT)
    print(f"Top {int(TOP_PCT*100)}% count:", len(top))
    display_cols = ['top_mean', 'rest_mean', 'mean_diff']
    print("Top vs Rest summary (top mean - rest mean):")
    print(summary[display_cols].head(20))

    # Plots
    plot_histogram(sim_df, col='predicted_popularity_with_viral')
    # Two-feature scatter examples (change to features you'd like)
    plot_scatter(sim_df, 'danceability', 'energy')
    plot_scatter(sim_df, 'principal_artist_followers', 'liveness')
    plot_correlation_heatmap(sim_df)

    # Visual comparisons for top vs rest for interesting features
    for feat in ['danceability', 'energy', 'acousticness', 'genres_numb', 'p
        if feat in sim_df.columns:
            plot_top_vs_rest_feature(sim_df, feat)
```

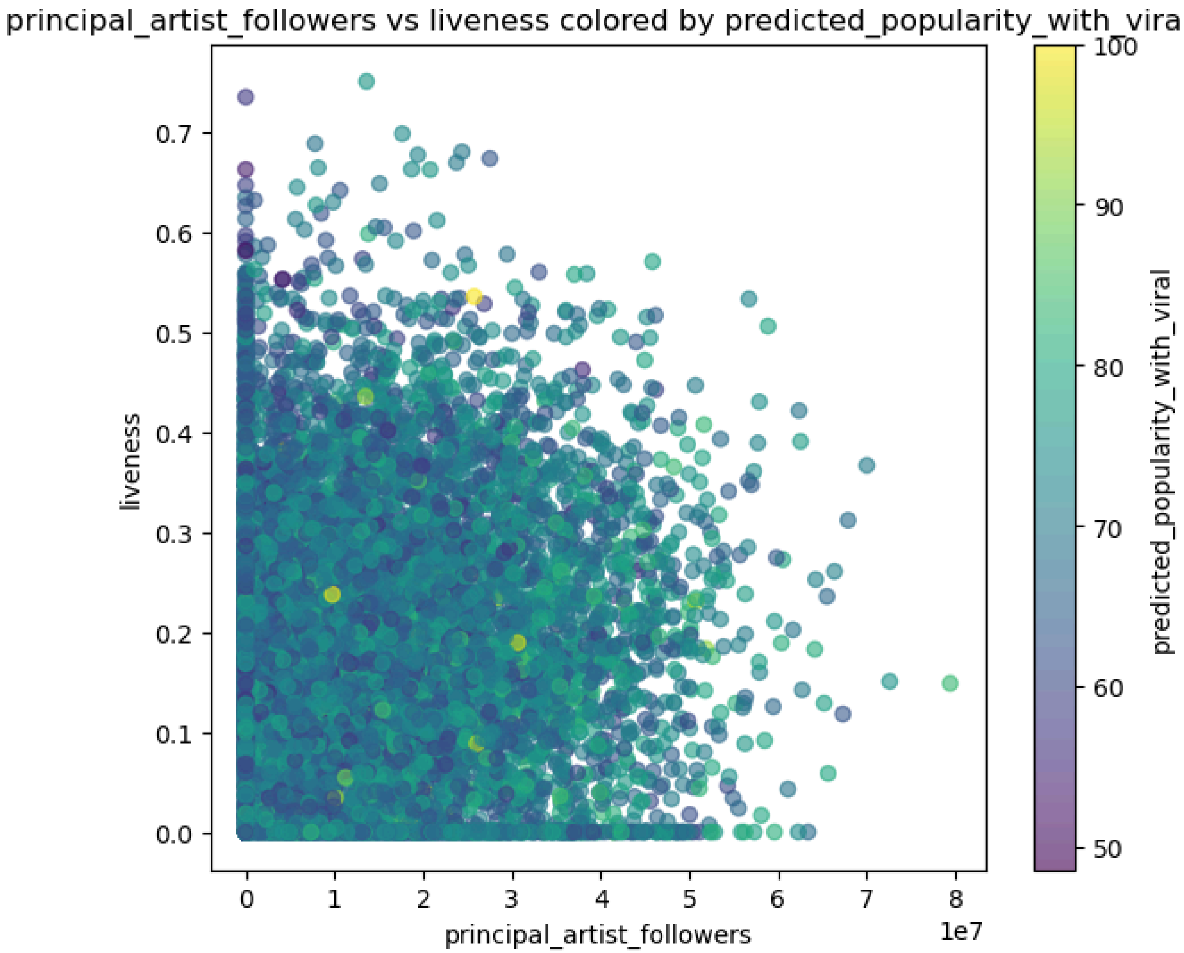
```python
# Save simulated dataset if you want
# sim_df.to_csv('simulated_songs.csv', index=False)
```

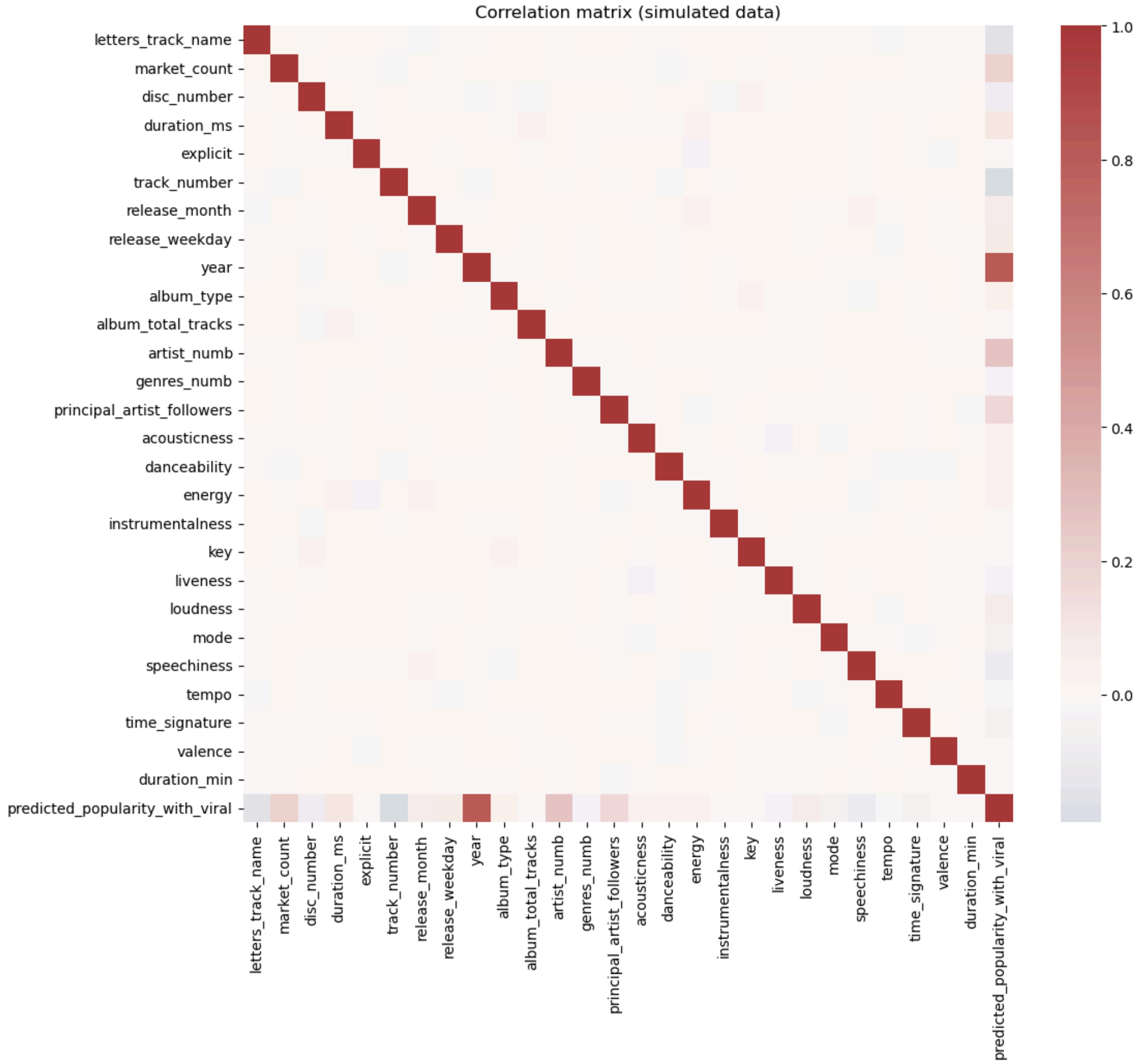
```
Simulated shape: (10000, 30)
Predictor stats:
                                   min              max           mean  \
col
letters_track_name                   0               76      12.510821
market_count                  1.000000       184.000000     159.008301
disc_number                          1               10       1.016405
duration_ms                      33493          2238733  228975.991303
explicit                         False             True       0.250519
track_number                         1               48       5.309220
release_month                        1               12       5.328194
release_weekday                      0                6       2.364463
year                              1986             2023    2004.258326
album_type                           0                2       0.208914
album_total_tracks                   1              176      14.248641
artist_numb                          1               40       1.249926
genres_numb                   1.000000        15.000000       3.776065
principal_artist_followers 100.000000 114675033.000000 9507767.987449
acousticness                  0.000000         0.996000       0.225428
danceability                  0.000000         0.988000       0.611739
energy                        0.000020         1.000000       0.652720
instrumentalness              0.000000         1.000000       0.045690
key                           0.000000        11.000000       5.265441
liveness                      0.000000         0.982000       0.183409
loudness                    -47.070000         0.522000      -7.521953
mode                          0.000000         1.000000       0.679020
speechiness                   0.000000         0.944000       0.089384
tempo                         0.000000       220.099000     121.007037
time_signature                0.000000         5.000000       3.929044
valence                       0.000000         0.994000       0.539131
duration_min                  0.558217        37.312217       3.816267

                                       std    dtype
col
letters_track_name                6.307239    int64
market_count                     57.877488  float64
disc_number                       0.179841    int64
duration_ms                   64891.284072    int64
explicit                          0.433312     bool
track_number                      4.434761    int64
release_month                     3.778310    int64
release_weekday                   1.826225    int64
year                             10.958820    int64
album_type                        0.563434     int8
album_total_tracks                9.375143    int64
artist_numb                       0.771115    int64
genres_numb                       1.999470  float64
principal_artist_followers 17923152.799975  float64
acousticness                      0.261618  float64
danceability                      0.162452  float64
energy                            0.205885  float64
instrumentalness                  0.165405  float64
key                               3.554011  float64
liveness                          0.146094  float64
loudness                          3.772877  float64
mode                              0.466853  float64
```

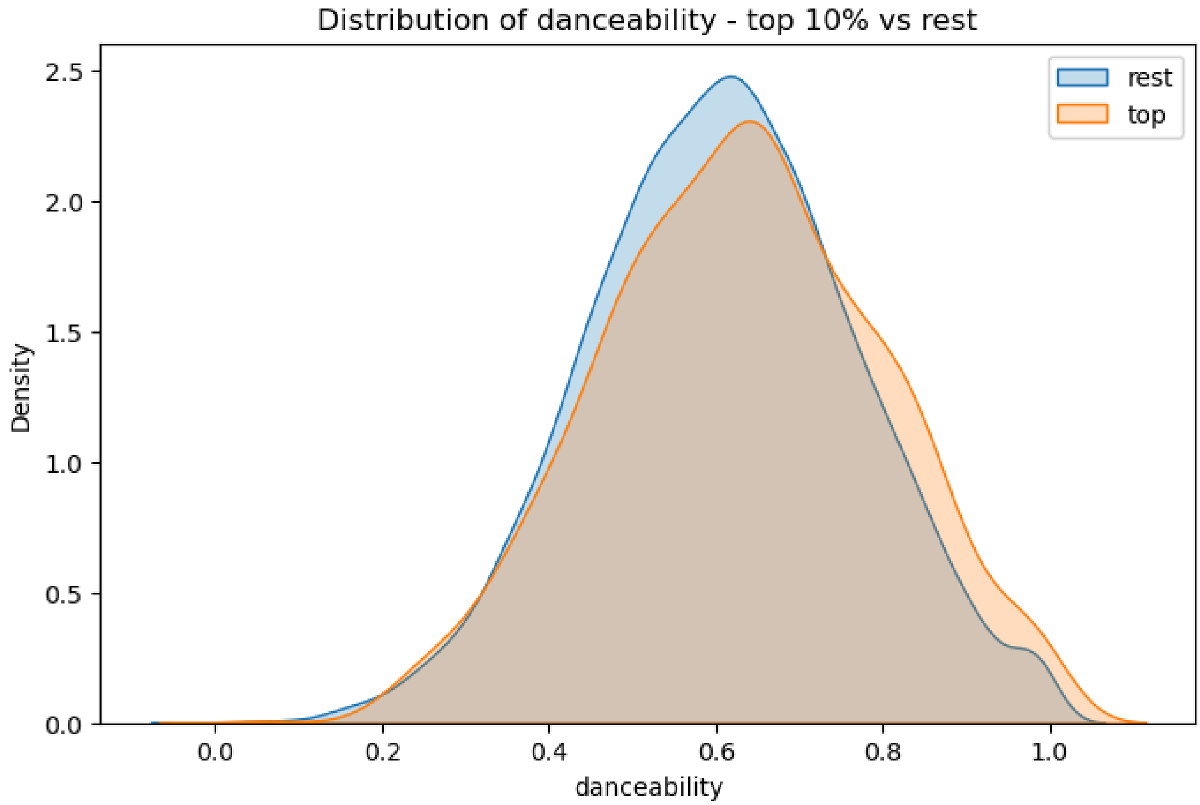
```
speechiness                          0.094261  float64
tempo                               30.195821  float64
time_signature                       0.358850  float64
valence                              0.245603  float64
duration_min                         1.081521  float64
Top 10% count: 1000
Top vs Rest summary (top mean - rest mean):
                                  top_mean        rest_mean       mean_diff
principal_artist_followers 18167389.212643 12326918.396237 5840470.816406
duration_ms                  1304507.636000  1132531.212778  171976.423222
market_count                     161.025743      144.572784       16.452959
year                            2017.930000     2003.088000       14.842000
artist_numb                       26.686000       19.736000        6.950000
album_total_tracks                90.062000       88.821556        1.240444
release_month                      6.948000        6.421222        0.526778
loudness                          -7.083061       -7.601161        0.518101
release_weekday                    3.375000        2.971778        0.403222
duration_min                      18.661000       18.464667        0.196333
album_type                         1.095000        0.971889        0.123111
danceability                       0.628153        0.606131        0.022022
acousticness                       0.267073        0.249835        0.017238
energy                             0.661337        0.647949        0.013388
instrumentalness                   0.095568        0.091874        0.003695
explicit                           0.496000        0.492667        0.003333
valence                            0.532927        0.539260       -0.006332
liveness                           0.183195        0.191905       -0.008710
speechiness                        0.084891        0.100936       -0.016045
mode                               0.598108        0.633682       -0.035575
```

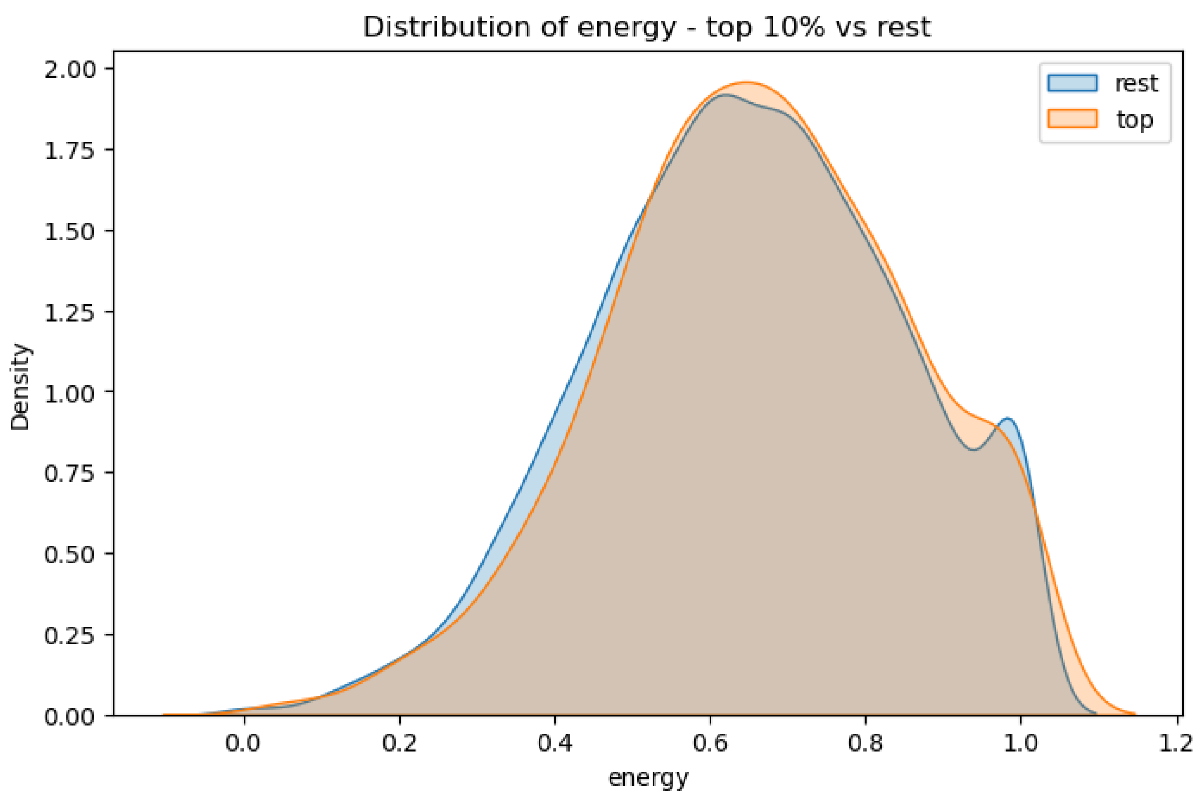


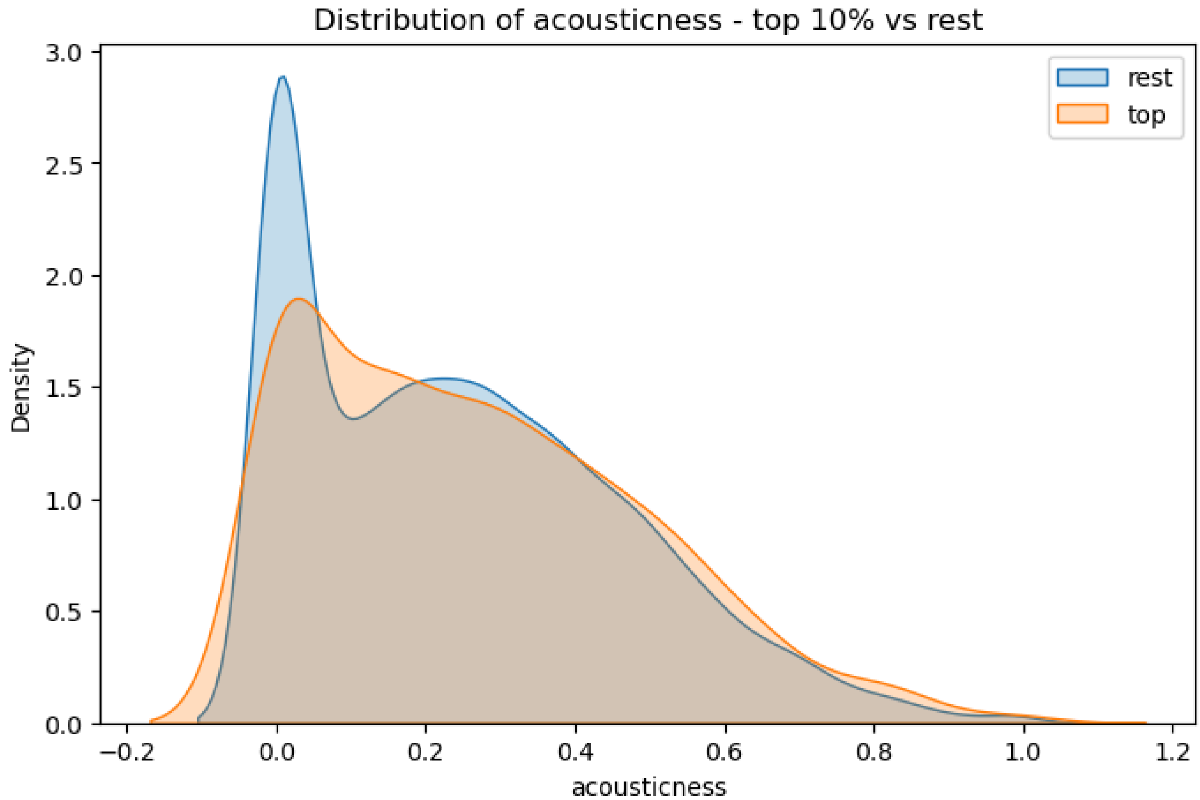

danceability vs energy colored by predicted_popularity_with_viral
energy
danceability
predicted_popularity_with_viral
0.0
0.2
0.4
0.6
0.8
1.0
50
60
70
80
90
100

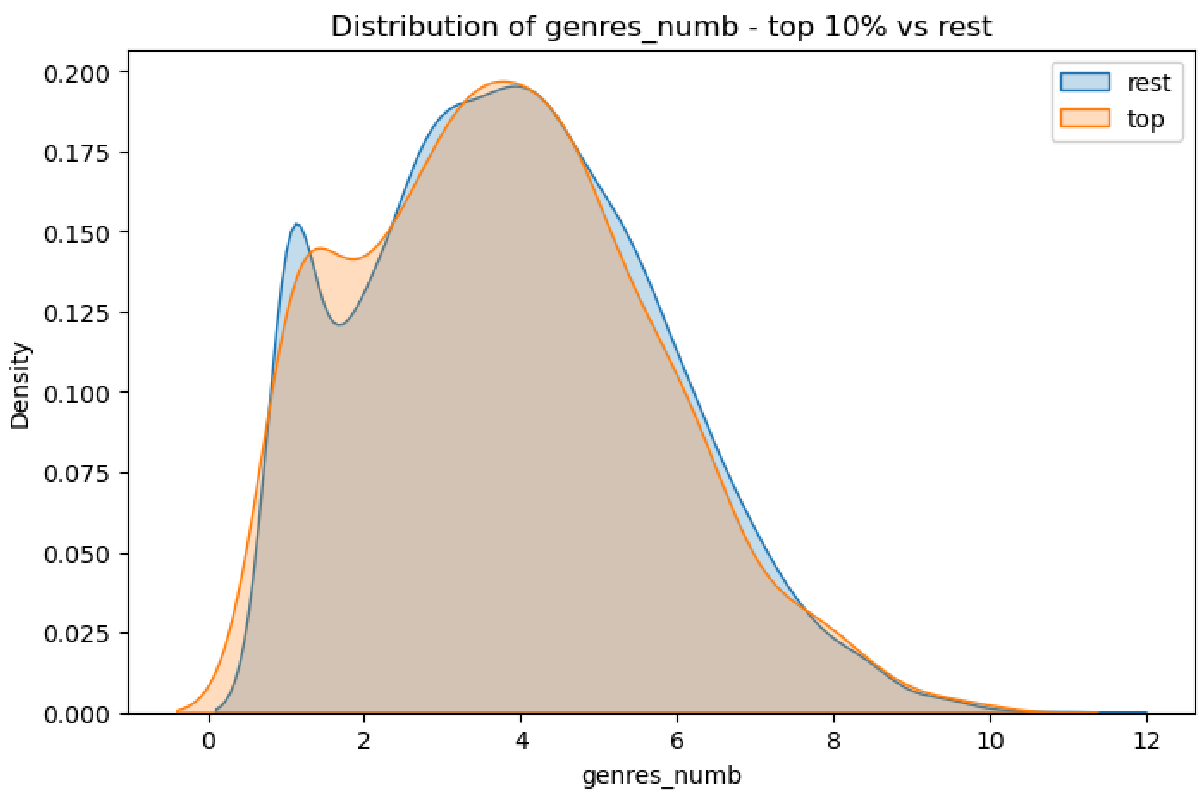

principal_artist_followers vs liveness colored by predicted_popularity_with_viral
liveness
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
principal_artist_followers
0
1
2
3
4
5
6
7
8
1e7
predicted_popularity_with_viral
50
60
70
80
90
100

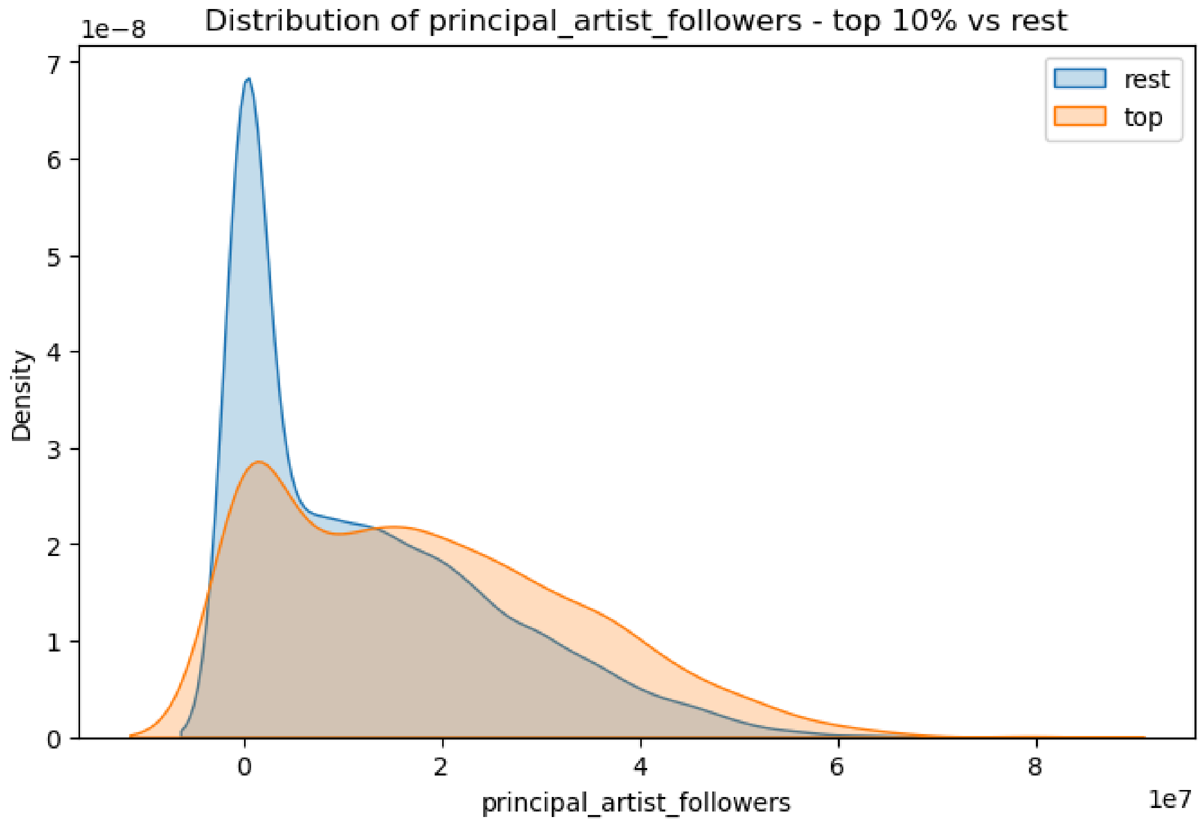

Correlation matrix (simulated data)
letters_track_name
market_count
disc_number
duration_ms
explicit
track_number
release_month
release_weekday
year
album_type
album_total_tracks
artist_numb
genres_numb
principal_artist_followers
acousticness
danceability
energy
instrumentalness
key
liveness
loudness
mode
speechiness
tempo
time_signature
valence
duration_min
predicted_popularity_with_viral
1.0
0.8
0.6
0.4
0.2
0.0

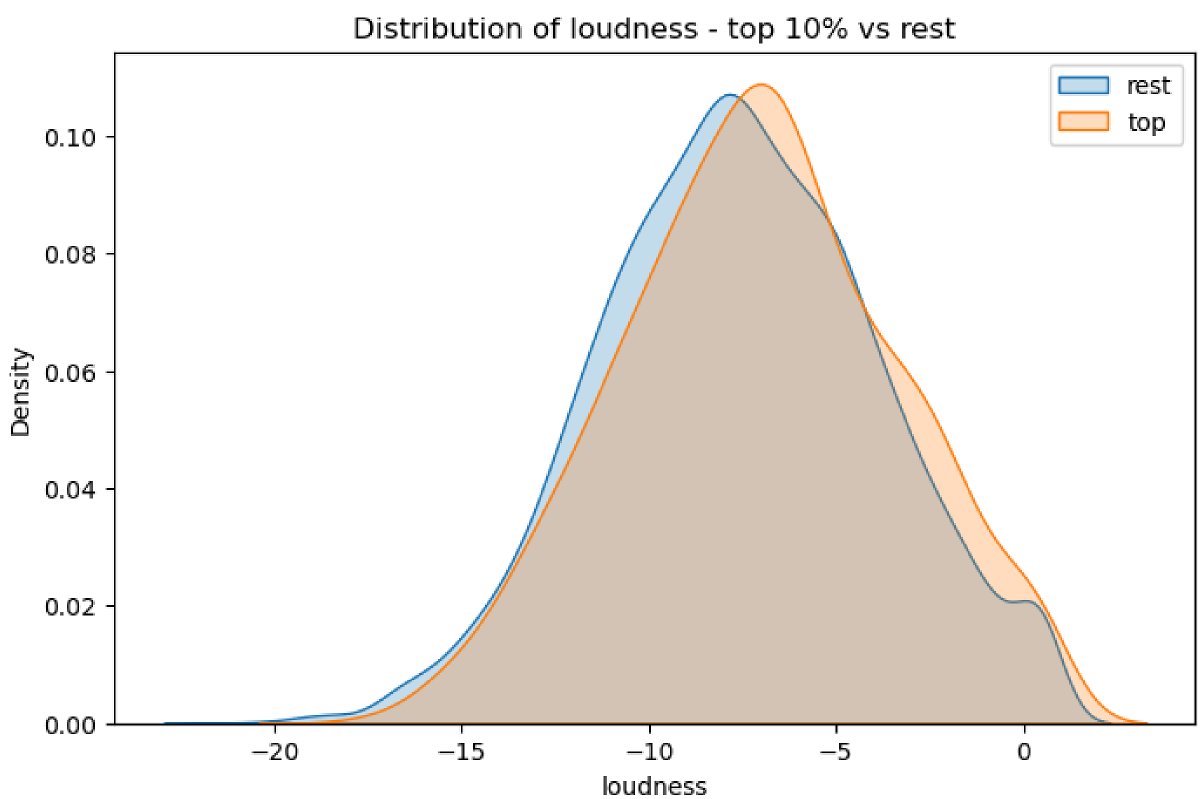

Distribution of danceability - top 10% vs rest
rest
top
Density
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
danceability


Distribution of energy - top 10% vs rest
rest
top
Density
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0
1.2
energy


Distribution of acousticness - top 10% vs rest
rest
top
Density
acousticness
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Distribution of genres_numb - top 10% vs rest
rest
top
Density
genres_numb
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
2
4
6
8
10
12





Very cool!

Key Conclusions

Overall we found that lots of people tend to listen to music in time_signature 3-4 and some of the best predictors for popularity are artist followers, and the year released. our simulations found that other variables such as time_signature and danceability can also play a role, yet the effect is less due to smaller values of time signature, and the fact that our data was too broad.

Future simulation projects should focus on more specific genres or audiences that we can target, which could provide more meaningful results instead of being so broad like our project.

We hypothesize that popularity is also affected by other variables such as advertisements, and managing teams, which funnel a lot of money in to create a persona for the artist creating the song.

More work can be done to run simulations on how we choose the music we listen to.