WOULD THEY PLAY?

Using Python and music APIs to predict the likelihood of an artist or band playing a music festival based on artist & song attributes.

BY CHRISTINE OSAZUWA

The Problem

Every year music festivals are announced with dozens to hundreds of artists. Does your favorite band fit in? Could the company's curating their list automate the process?

Given music data from the top music tech companies in the world, can you predict the likelihood of an artist playing a popular festival?

WHAT DATA WOULD I NEED?

LIST OF ALL BANDS HISTORICALLY THAT PLAYED EACH FESTIVAL FOR 5-10 YEARS

HISTORICAL DATA ON THE BANDS THAT PLAYED AT THE TIME THEY PLAYED INCLUDING:
SALES VOLUME, POPULARITY OF ARTIST, POPULARITY OF TOP TRACKS, LISTENERSHIP, RECORD LABEL, SOCIAL MEDIA METRICS, RADIO PLAYS, AVERAGE CONCERT TICKET PRICE, AWARDS WON, APPEARANCE ON BILLBOARD CHARTS, ATTRIBUTES OF TOP SONGS, GOOGLE TRENDS

EVERGREEN DATA ABOUT THE BANDS:
NUMBER OF BAND MEMBERS, PLACE OF ORIGIN, SIMILARITIES TO OTHER ARTISTS, START YEAR, BREAK UP YEAR, GENRE, IF ARTIST HAVE PLAYED THE FESTIVAL BEFORE

Acquiring & Cleaning the Data

While all the data exists for the artists, the challenge was finding and connecting data for various artists together by scraping the web & using public APIs.

The Scope

Original:

Get 5-10 years of historical data for 10-15 large world-wide festivals.

Revised:

As I started the process, I realized it would be challenging to get all the festival data I wanted, so for the sake of time I choose to limit to one festival to start:

Vans Warped Tour—a summer long tour music festival usually played by pop punk, punk, hardcore & emo bands.

Getting 10 Years of Warped Tour Data

A special issue I ran into with Warped Tour data is that the artist change daily for the festival. One area of consistency I found is that Warped Tour releases a compilation album each year with a song from most of the artists that played a significant amount of dates for the festival, so I choose to use that compilation as my baseline.

Amazon

Library/API: bottlenose

Used "ItemSearch" function to find the compilation albums, then iterated through results to get track listings from each year. Missing 2013-2015

iTunes

Library/API: python-itunes

Used "search_album" function to find compilation albums that were missing. Then iterated though results to get track listings from each year. Missing 2014.

Wikia

No Library/API, used BeautifulSoup

Wikia is a crowdsourced site with sections dedicated to specific topics. I used BS4 to pull the data for the artists for the missing date of 2014.

Getting Artist Data

This involved a lot of moving parts but thankfully, these API's play well together.

Spotify

Music Streaming Service

Library/API: spotipy

Used spotipy's "search" function to locate each artist in the Warped Tour compilation album list, then appended their Spotify ID. From there, I was able to use spotipy's "artist" feature to append artist popularity, artist genre. Finally, I used spotipy's "artist_top_tracks" to find the artists top 10 tracks by popularity on Spotify and append an average of those song's attributes.

MusicBrainz

Crowd Sourced Music Database

Library/API: musicbrainzngs

Used "search" function to locate each artist. Appended the Gracenote/MusicBrainz ID for each artist and also returned a list of band members. From there, I did a count of number of members and appended to my data list.

Last FM

Music Plays Tracker

API: pylast

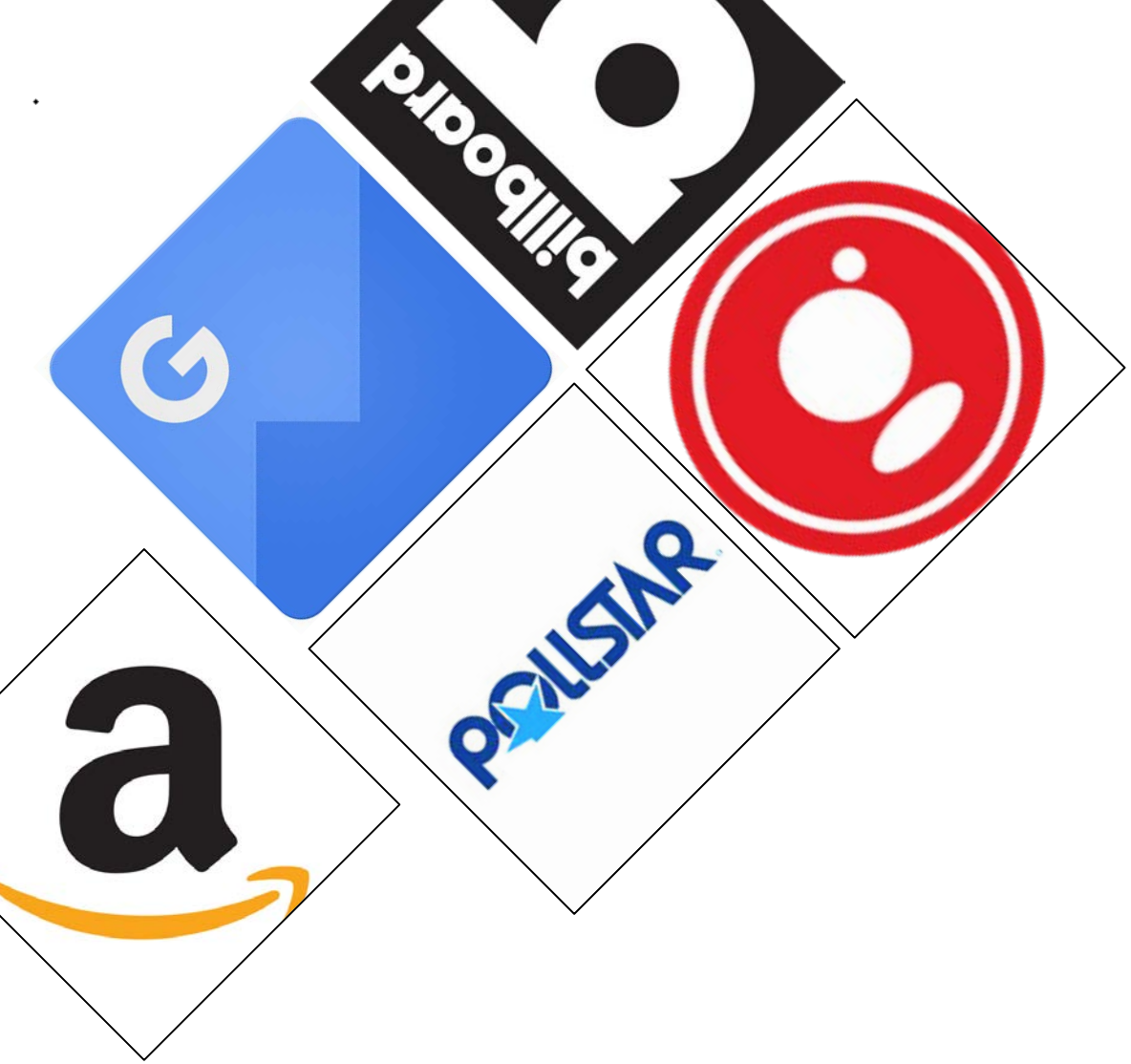
Used the ID I got from MusicBrainz to use pylast's function "get_artist_by_mbid". From there, I appended the artist play count on LastFM and their overall listener count.

Gracenote

Music Database

Library/API: pygn

Used pygn's "search" function to search for the artist by name and then append the decade that the artist started in.



Failed Attempts to Get Artist Data

These sites/APIs had data I wanted but it was too challenging to get.

Amazon: Lack of consistent data among the artists.

Google Trends: No official API, limited to 10 queries a minute. Too slow for mass searching.

Pollstar: API only for artists. Used BeautifulSoup to parse the site for concert ticket cost data, but very limited amount, resulted in too many missing values.

Billboard: Attempted to find the frequency in which an artist was on a Billboard chart but API lacked the relevant charts.

Gracenote: Attempted to get artist origin data but rate limits too low for mass searching on multiple queries, so I prioritized “era”.

Ultimate Music Database: Incredibly robust site, however; no API and very dated, so even parsing with BeautifulSoup proved challenging.

Cleaning The Data

I didn't really have time...

Instead of manually cleaning the data, I opted to add in exceptions to errors. Instead of having missing values to clean, I structure my functions with "try"/"except", so that in the case of an error, the function continues, and would pass through a '0'.

Resulting Dataset

After gathering data from various websites, I was able to acquire and successfully attribute a small portion of the initial data I sought out for this project.

Data Explained

band: *str*, name of band/artist

spotify_id: *str*, alpha numeric string assign to artist (provided by Spotify)

musicbrainz_id: *str*, alpha numeric string assign to artist (provided by MusicBrainz)

member_count: *int*, count of number of people associated with or in the given group/artists (provided by MusicBrainz)

spotify_popularity: *int*, popularity of artist based on Spotify's algorithm range 0-100 (provided by Spotify)

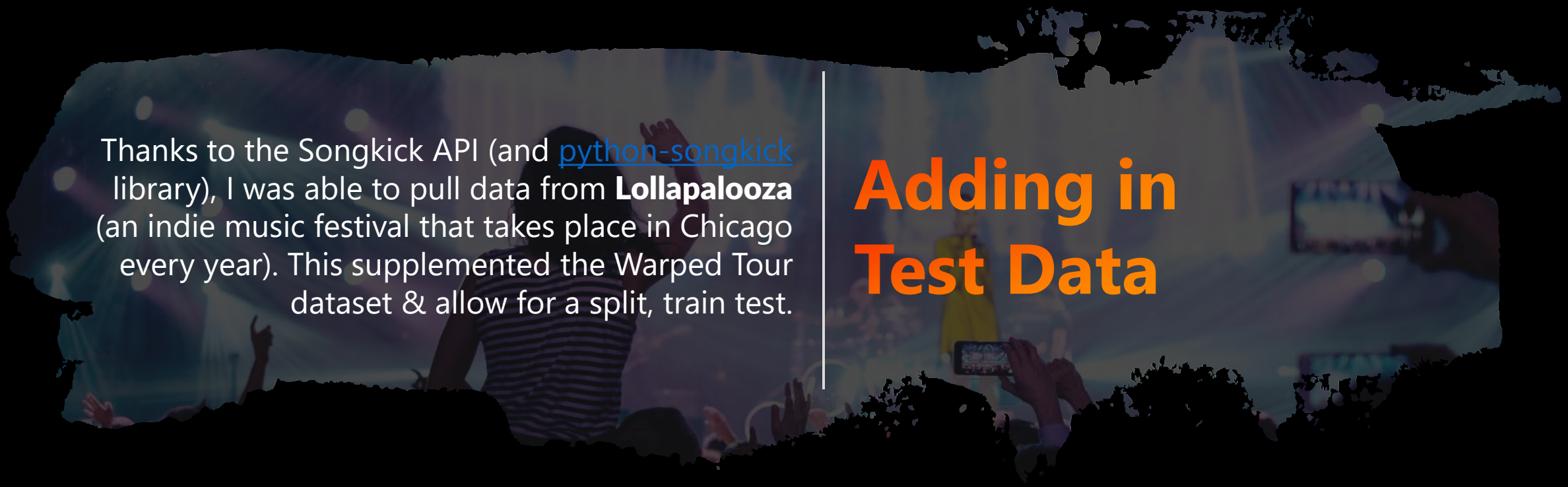
spotify_genre: *str*, the genre assigned the artist (provided by Spotify)

duration_ms: *int*, length of the top 10 songs (as defined by Spotify) of the given artist averaged (provided by Spotify/EchoNest)

acousticness, danceability, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time_signature, valence: *int*, attributes of the top 10 songs (as defined by Spotify) of the given artist averaged (provided by Spotify/EchoNest)

lastfm_play_count: *int*, the number of times an artists song has been counted on LastFM (provided by LastFM)

lastfm_listener_count: *int*, the number of people that have listened to an artist on LastFM (provided by LastFM)



Thanks to the Songkick API (and python-songkick library), I was able to pull data from **Lollapalooza** (an indie music festival that takes place in Chicago every year). This supplemented the Warped Tour dataset & allow for a split, train test.

Adding in Test Data

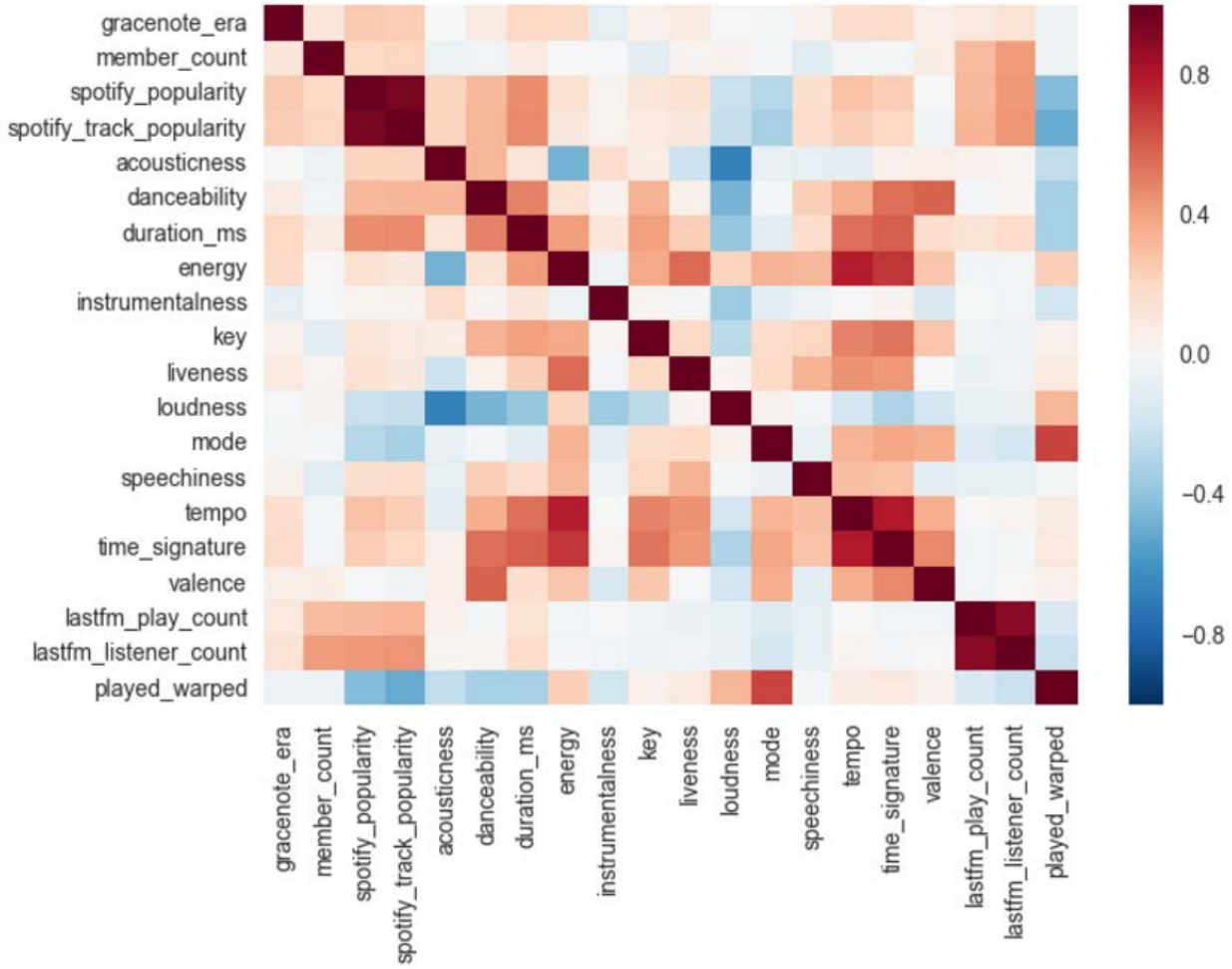
Describe the Full Dataset

	gracenote_era	member_count	spotify_popularity	spotify_track_popularity	acousticness	danceability	duration_ms	energy
count	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000
mean	1827.261905	1.997619	47.021429	36.036614	0.088497	0.485118	209273.640291	0.787645
std	568.654886	2.893907	20.857787	17.651959	0.125192	0.139233	49337.047089	0.190146
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2000.000000	0.000000	35.750000	24.275000	0.006445	0.415800	194693.150000	0.731075
50%	2000.000000	1.000000	51.000000	38.800000	0.048686	0.479350	212608.650000	0.846400
75%	2010.000000	4.000000	61.250000	49.000000	0.109498	0.571700	231531.425000	0.914300
max	2010.000000	25.000000	91.000000	80.000000	0.982200	0.870000	364586.000000	0.990700

```
count                  420
unique                  93
top       alternative emo
freq                    58
Name: spotify_genre, dtype: object
```

instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence	lastfm_play_count	lastfm_listener_count	played_warped
420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	4.200000e+02	4.200000e+02	420.000000
0.055169	5.013029	0.209107	-5.214856	0.525265	0.090169	126.795459	3.766243	0.443311	1.168587e+07	2.891402e+05	0.776190
0.118723	1.499489	0.077679	2.124877	0.349941	0.052427	25.715974	0.705721	0.172109	3.238140e+07	4.965200e+05	0.417293
0.000000	0.000000	0.000000	-20.585900	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000
0.000056	4.200000	0.162757	-6.048950	0.200000	0.055810	119.995725	3.900000	0.336075	4.178188e+05	2.228700e+04	1.000000
0.002250	5.050000	0.202990	-4.885200	0.600000	0.078370	130.044400	4.000000	0.444500	2.707704e+06	1.119410e+05	1.000000
0.061674	6.000000	0.248805	-4.125600	0.800000	0.114827	139.563475	4.000000	0.567675	1.067752e+07	3.221592e+05	1.000000
0.907400	9.000000	0.714000	0.000000	1.000000	0.325720	166.244400	4.300000	0.946400	4.520452e+08	4.520061e+06	1.000000

Correlation



Correlations with Target

	played_warped
gracenote_era	-0.048451
member_count	-0.051827
spotify_popularity	-0.430225
spotify_track_popularity	-0.503037
acousticness	-0.247386
danceability	-0.328497
duration_ms	-0.329588
energy	0.240485
instrumentalness	-0.192919
key	0.054256
liveness	0.087012
loudness	0.332655
mode	0.676218
speechiness	-0.018516
tempo	0.079653
time_signature	0.097469
valence	0.054526
lastfm_play_count	-0.153122
lastfm_listener_count	-0.225267
played_warped	1.000000

The Models

Since acquiring the right data took so long, I did some minimal modeling using Linear Regression, Logistic Regression, Decision Trees & K-Nearest Neighbors

Modeling

Did the basics here to make sure everything worked!

pandas

Used pandas throughout to import and export CSV files in order to store data & manipulate.

sklearn + statsmodel

From sklearn, imported linear_model, feature_selection, import cross_validation, metrics, LinearRegression, cross_val_score, export_graphviz, KNeighborsClassifier, LogisticRegression, DecisionTreeClassifier in order to run Logistic Regression, Linear Regression, Decision Trees & KNN. From statsmodel I used to run a Linear Regression model.

numpy

Imported numpy in order to do basic math: mean, square root, etc.

seaborn

Imported seaborn for data visualizing the correlation heatmap.

Music APIs

Imported pygn, spotipy, billboard, pylast, songkick, discogs_client, pytrends, musicbrainzngs, pytrends

Others

Imported requests, sys, os, itertools, re, bs4, json, time, string, collections

Results

Did the basics here to make sure everything worked!

accuracy

```
Linear Regression Scoring Mean Squared:
0.425836015364

Logistic Regression Scoring Mean Squared Error:
0.519928931156

Logistic Regression Mean Accuracy:
0.709425817045

Decision Tree Root Mean Squared Error
0.123977925841

Decision Tree Mean Accuracy
0.965845648604

KNN Mean Accuracy
0.480986798669

KNN Root Mean Squared Error
0.766276368744
```

confusion matrix

```
CONFUSION MATRIXES

LOGISTIC REGRESSION CONFUSION MATRIX
[[  9  85]
 [ 10 316]]

DECISION TREE CONFUSION MATRIX
[[ 89   5]
 [  0 326]]

KNN CONFUSION MATRIX
[[ 37  57]
 [ 10 316]]
```

train/test scores

```
LINEAR REGRESSION TRAIN MODEL SCORES (model/r^2)
0.689662646986
0.689662646986

LINEAR REGRESSION TEST MODEL SCORES (model/r^2)
0.611597288673
0.611597288673

LOGISTIC REGRESSION TRAIN MODEL SCORES (model/r^2)
0.753571428571
-0.397569444444

LOGISTIC REGRESSION TEST MODEL SCORES (model/r^2)
0.778571428571
-0.315151515152

DECISION TREE TRAIN MODEL SCORES (model/r^2)
0.996428571429
0.97974537037

DECISION TREE TEST MODEL SCORES (model/r^2)
0.971428571429
0.830303030303

KNN TRAIN MODEL SCORES (model/r^2)
0.835714285714
0.068287037037

KNN TEST MODEL SCORES (model/r^2)
0.771428571429
-0.357575757576
```

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	0.8958	0.074	12.051	0.000	0.750 1.042
gracenote_era	1.127e-05	2.24e-05	0.503	0.615	-3.28e-05 5.53e-05
member_count	0.0060	0.005	1.249	0.212	-0.003 0.016
spotify_popularity	0.0077	0.002	3.930	0.000	0.004 0.012
spotify_track_popularity	-0.0138	0.002	-5.907	0.000	-0.018 -0.009
acousticness	0.1557	0.162	0.961	0.337	-0.163 0.474
danceability	-0.4772	0.166	-2.879	0.004	-0.803 -0.151
duration_ms	-1.014e-06	3.99e-07	-2.545	0.011	-1.8e-06 -2.31e-07
energy	0.3084	0.194	1.587	0.113	-0.074 0.690
instrumentalness	-0.3026	0.118	-2.574	0.010	-0.534 -0.072
key	0.0177	0.010	1.797	0.073	-0.002 0.037
liveness	-0.5972	0.203	-2.948	0.003	-0.996 -0.199
loudness	0.0263	0.011	2.488	0.013	0.006 0.047
mode	0.6619	0.048	13.807	0.000	0.568 0.756
speechiness	0.7232	0.288	2.513	0.012	0.157 1.289
tempo	-0.0023	0.001	-2.294	0.022	-0.004 -0.000
time_signature	0.1010	0.043	2.348	0.019	0.016 0.186
valence	-0.3007	0.120	-2.499	0.013	-0.537 -0.064
lastfm_play_count	1.231e-09	9.03e-10	1.363	0.174	-5.45e-10 3.01e-09
lastfm_listener_count	-7.819e-08	6.49e-08	-1.205	0.229	-2.06e-07 4.94e-08

Omnibus:	38.559	**Durbin-Watson:**	1.407
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	83.801
Skew:	-0.501	**Prob(JB):**	6.35e-19
Kurtosis:	4.946	**Cond. No.**	8.90e+08

Dep. Variable:	played_warped	**R-squared:**	0.673
Model:	OLS	**Adj. R-squared:**	0.658
Method:	Least Squares	**F-statistic:**	43.37
Date:	Wed, 14 Dec 2016	**Prob (F-statistic):**	9.36e-85
Time:	14:09:09	**Log-Likelihood:**	6.4830
No. Observations:	420	**AIC:**	27.03
Df Residuals:	400	**BIC:**	107.8
Df Model:	19		
Covariance Type:	nonrobust		

Linear Regression Results

For the sake of simplicity, I opted to use this Linear Regression model in my function though the overall R2 is not very strong.

Getting It All To Work Together!

Getting New Festival Data

I needed to automate getting the data for each artist, for each festival, so I created a function to run that process. Given a list of bands, the function can iterate through that list, append all the data necessary and output a pandas data frame.

Predicting Band Likelihood to Play

I created the ability to input an artist and then have a function run that:

1. Use the above function that collects all the data for the artist needed to run the model and places it in a pandas data frame.
2. Runs the model
3. Outputs a plain text answer to the artist's likelihood of playing that festival

Results

Getting new festival data:

Lollapalooza 2016:

	bands	spotify_id	musicbrainz_id	gracenote_era	member_count	spotify_popularity	spotify_genre
0	Red Hot Chili Peppers	0L8ExT028jH3ddEcZwqJJ5	8bfac288-ccc5-448d-9573-c33ea2aa5c30	1980	10	85	alternative rock
1	Radiohead	4Z8W4fKeB5YxbusRsdQVPb	a74b1b7f-71a5-4011-9441-d0b5e4122711	1990	6	79	alternative rock
2	Ellie Goulding	0X2BH1fck6amBloJhDVmmJ	33ca19f4-18c8-4411-98df-ac23890ce9f5	2010	1	82	dance pop
3	Lana Del Rey	00FQb4jTyendYWaN8pK0wa	b7539c32-53e7-4908-bda3-81449c367da6	2010	1	84	dance pop
4	M83	63MQldklfxkjYDoUE4Tppz	6d7b7cd4-254b-4c25-83f6-dd20f98ceacd	2000	5	72	alternative dance
5	J. Cole	6l3HvQ5sa6mXTsMTB19rO5	875203e1-8e58-4b86-8dcb-7190faf411c5	2010	1	90	dwn trap

94 rows × 23 columns

Getting a Prediction:

```
In [156]: get_data2('Blink-182')
Blink-182 is likely to play Warped Tour!

In [157]: get_data2('Korn')
Korn is not likely to play Warped Tour.
```

Conclusions

There is so much left to do!

Limitations

APIs & Websites

Gathering data from the internet is time & memory intensive. In addition, restrictions & rate limits, as well as continuously learning new libraries were strong barriers. Using “sleep” is important!

Lack of Historical Data

In order to have a large enough data set, I looked at bands that have played the festival across several years, however; I don’t have the data to say how that band was doing within the year they played. This automatically skews my data, thus finding more artist attributed data would help the modeling process.

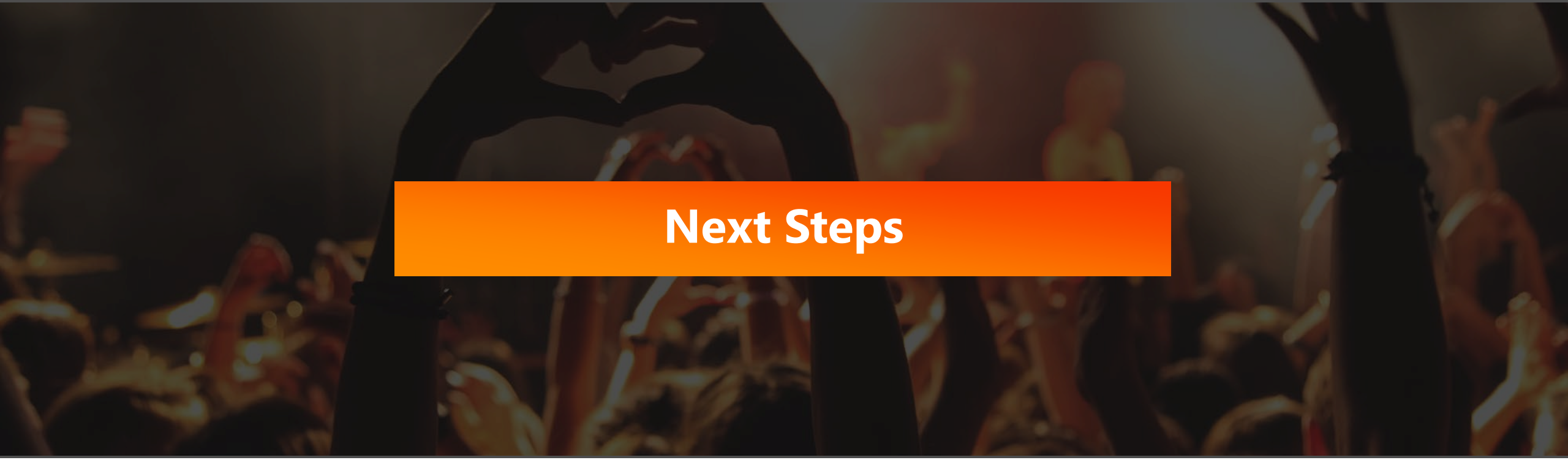
What I Learned

Understanding yield, return, except & try

These commands were instrumental to making my functions work properly without having to have completely clean data.

Importance of Labels & Conventions

When you have just a few lines of code it’s easy to remember what you’re doing but when you approach hundreds, it’s important to make note of what you’re doing and use variables that make sense. Also, important to know when to use a function & when to just use a variable.



Next Steps

Getting More Data

During this process I requested access to several APIs such as Next Big Sound that have yet to be granted, which would provide me with additional artist data. In addition, I came across the Discography.com data late in my process & hope to incorporate some of that data as well.

Adding More Festivals

Now that I have the functionality created for one festival, I hope to replicate amongst several festivals to eventually allow a user to enter an artist name and it will query data from multiple festivals and provide the likelihood of them playing each one.

Improving The Model

Within the scope of this project I focused primarily on data acquisition and creating a proof of concept, so given more time, I intend to refine and improve upon my modeling.

Creating a User Interface

Ultimately, I'd like for this to be web based so users can simply input an artists & search, then it will display the results. In order for this to happen and deliver the results in a timely fashion, I may need to start exploring cache & memory.

Thank You!