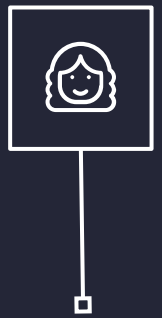
Spotify API

An analysis of genre ‘Audio Features’

MEET THE TEAM



KATHRYN PANGER

WILLIAM FORSYTH

HEESUNG SHIM

CAITLIN CALSBEEK

Project Description

Utilize Spotify's API, to explore and identify trends in the audio qualities in music genres.

TABLE OF CONTENTS

“In God we trust, all others must bring data”

— W. Edwards Deming

01

RESEARCH QUESTIONS

Thought-provoking questions to ask and answer through data analysis

KATHRYN'S RESEARCH QUESTIONS

RESEARCH QUESTION 1
(Inductive Exploratory)

Which variables seem to have a relationship with one another in this dataset

RESEARCH QUESTION 2
(After Exploration)

What is the relationship between "energy" and "danceability?"

RESEARCH QUESTION 3
(After Exploration)

Does this relationship differ by genre?

RESEARCH QUESTION 4
(After model-building)

How does Spotify define its "energy" variable?

HEESUNG'S RESEARCH QUESTION

"WHICH GENRE USUALLY HAS THE LONGEST AND THE SHORTEST TRACK?"

CAITLIN'S RESEARCH QUESTION
"HOW DOES 'KEY' VARY ACROSS GENRES?"

Predominent Keys ?

What is/are the predominant key/s across genres?

Key Distribution

How much variety is there in distribution of keys between genres ?

02

DATA SOURCING

Identifying and understanding the data
used to answer our research questions

QUERYING THE SPOTIFY API (SPOTIPY – WEB API WRAPPER)

We decided to utilize a lightweight Python library called **Spotipy** because -

1# Python Library for the Spotify API

Lightweight and provides all the features of Spotify's API

We think it's the best resourced Web API Wrapper for the Spotify API.

PIECING TOGETHER SPOTIPY

Understanding the User Credentials Flow

Client ID & Secret Setup

API Reference | Object | Module analysis

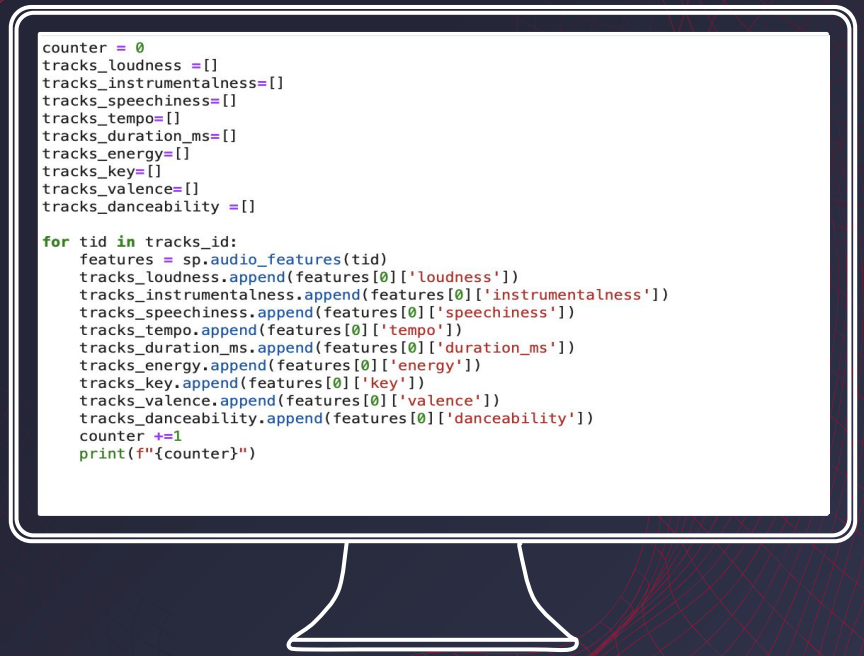
Overcoming Spotify API rate limits

CALLING THE SPOTIFY API

Introduction

We formulated 3x API Calls, with different clear objectives -

- Return a list of **Genres**.
- Return a list of **Tracks & Characteristics**.
- Return **Audio Features** for each Track.



```
counter = 0
tracks_loudness =[]
tracks_instrumentalness=[]
tracks_speechiness=[]
tracks_tempo=[]
tracks_duration_ms=[]
tracks_energy=[]
tracks_key=[]
tracks_valence=[]
tracks_danceability =[]

for tid in tracks_id:
    features = sp.audio_features(tid)
    tracks_loudness.append(features[0]['loudness'])
    tracks_instrumentalness.append(features[0]['instrumentalness'])
    tracks_speechiness.append(features[0]['speechiness'])
    tracks_tempo.append(features[0]['tempo'])
    tracks_duration_ms.append(features[0]['duration_ms'])
    tracks_energy.append(features[0]['energy'])
    tracks_key.append(features[0]['key'])
    tracks_valence.append(features[0]['valence'])
    tracks_danceability.append(features[0]['danceability'])
    counter +=1
    print(f"{counter}")
```

RETURNING A GENRE LIST

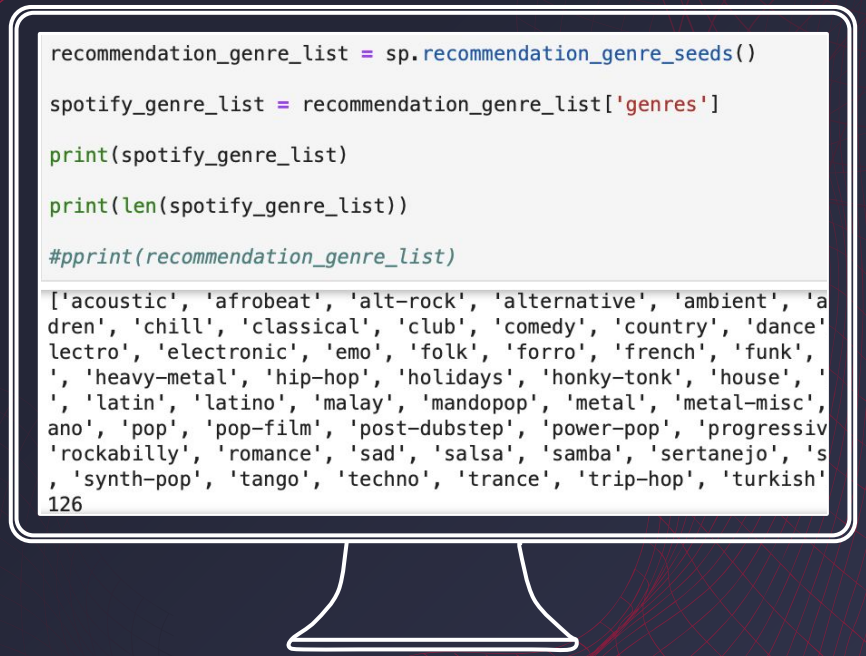
(API Call #1-3)

The Team wanted to ensure we captured as many Genres as possible. We used -

```
recommendation_genre_list = sp.recommendation_genre_seeds()
```

which called the API for all the 'recommended' genres available on Spotify. We then saved these genres to a list, for later reference

```
spotify_genre_list = recommendation_genre_list['genres'].
```



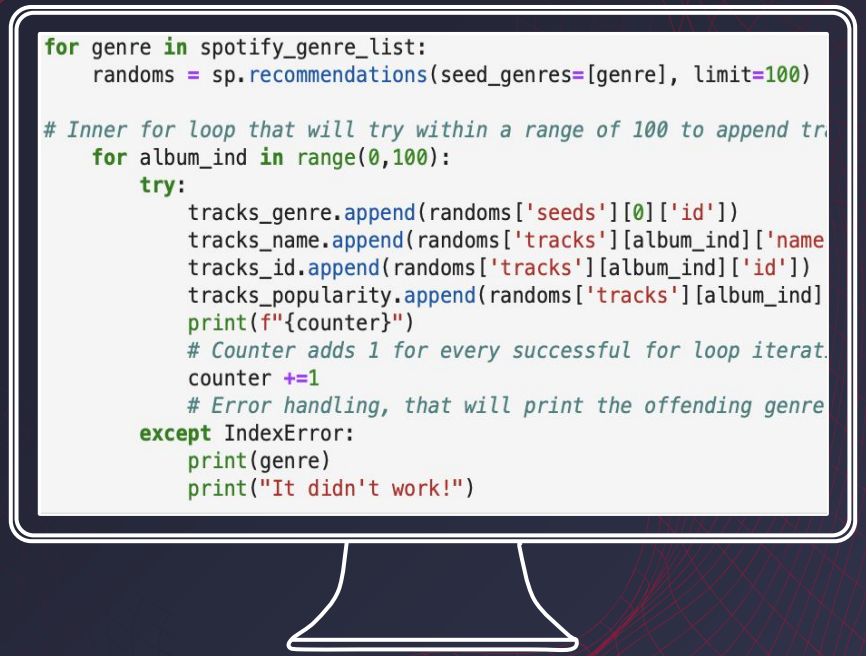
RETURNING TRACKS & CHARACTERISTICS

(API Call #2-3)

The next stage, was to take the 126 returned genres and perform an API Call to return 100 random tracks for each **genre** thereby giving us c.12,600 tracks.

Our code was composed of -

- An **outer for loop** - to iterate for our list containing the 126 genres.
- An **inner for loop** to iterate through `album_ind`
- **`Try: | Except:`** statement to append the returned information for each track to our `randoms` list

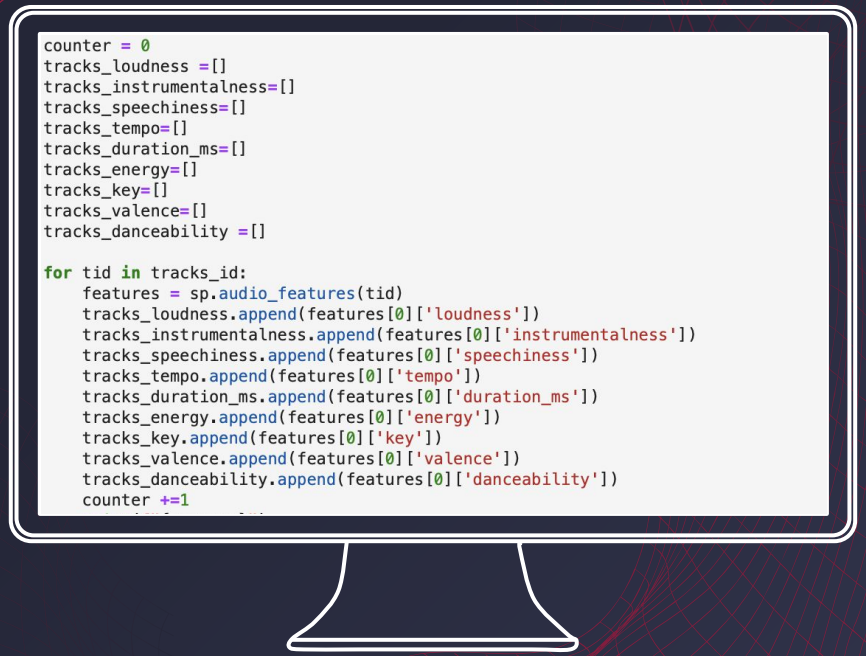


```
for genre in spotify_genre_list:
    randoms = sp.recommendations(seed_genres=[genre], limit=100)

# Inner for loop that will try within a range of 100 to append tr
    for album_ind in range(0,100):
        try:
            tracks_genre.append(randoms['seeds'][0]['id'])
            tracks_name.append(randoms['tracks'][album_ind]['name
            tracks_id.append(randoms['tracks'][album_ind]['id'])
            tracks_popularity.append(randoms['tracks'][album_ind]
            print(f"{counter}")
            # Counter adds 1 for every successful for loop iterat
            counter +=1
            # Error handling, that will print the offending genre
        except IndexError:
            print(genre)
            print("It didn't work!")
```

RETURNING TRACK AUDIO FEATURES

(API Call #3-3)

- The final API Call was concerned with appending all the **Audio Characteristics** to a DataFrame (`track_dataframe`) along with the previously API Call returned data.
- The biggest challenge was API Timeout 404 Errors.
- By implementing a time.sleep() argument, we could bypass Spotify's API limits, by pausing our search momentarily between tracks.



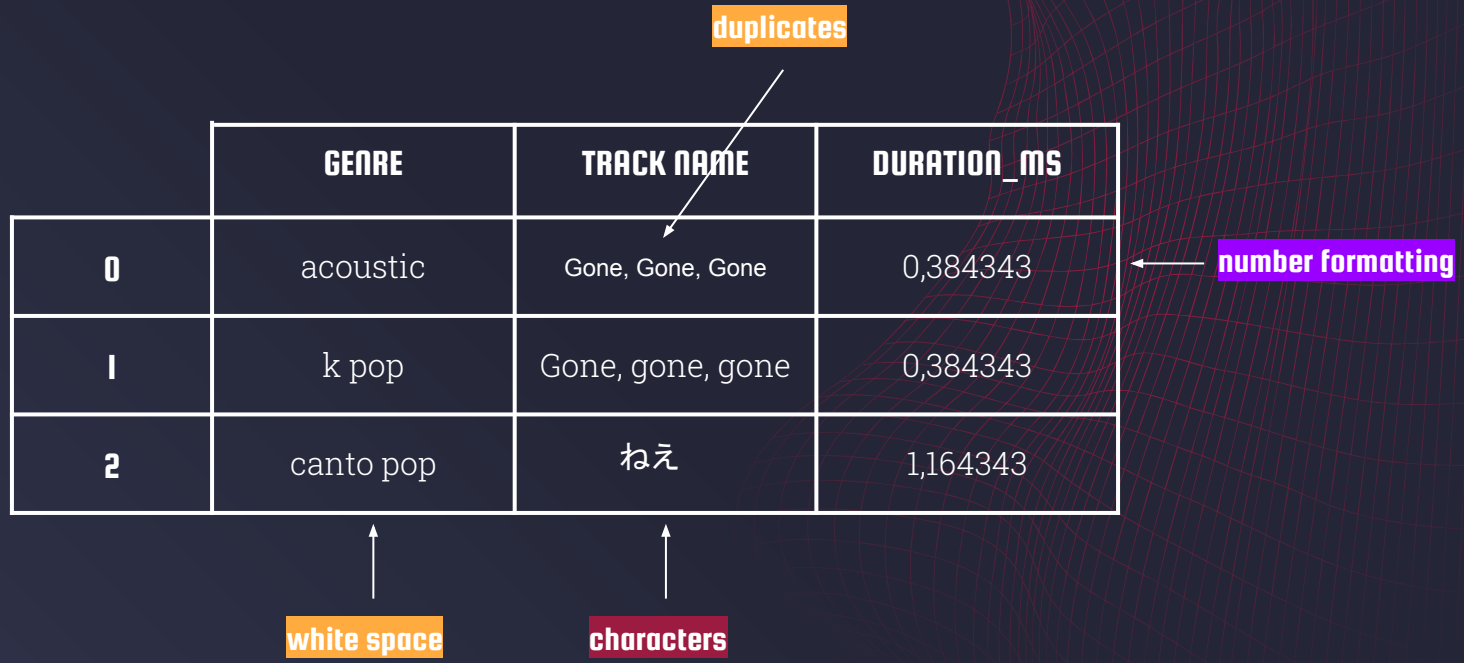
```
counter = 0
tracks_loudness =[]
tracks_instrumentalness=[]
tracks_speechiness=[]
tracks_tempo=[]
tracks_duration_ms=[]
tracks_energy=[]
tracks_key=[]
tracks_valence=[]
tracks_danceability =[]

for tid in tracks_id:
    features = sp.audio_features(tid)
    tracks_loudness.append(features[0]['loudness'])
    tracks_instrumentalness.append(features[0]['instrumentalness'])
    tracks_speechiness.append(features[0]['speechiness'])
    tracks_tempo.append(features[0]['tempo'])
    tracks_duration_ms.append(features[0]['duration_ms'])
    tracks_energy.append(features[0]['energy'])
    tracks_key.append(features[0]['key'])
    tracks_valence.append(features[0]['valence'])
    tracks_danceability.append(features[0]['danceability'])
    counter +=1
```

03

DATA EXPLORATION & CLEANUP

Utilizing Pandas & 'Spotipy' to explore and cleanse our data

WHILST OUR DATAFRAME LOOKED GREAT ON THE SURFACE, SIGNIFICANT CLEANSING WAS REQUIRED...



HOW DID WE CLEANSE OUR DATA?

Duplicates

```
DataFrame.drop_duplicates
```

Bad Genres

```
genre_list.remove("funk")
```

Nan Values

```
DataFrame.dropna()
```

PRESENTING OUR DATAFRAME...

	Genre	Popularity	Loudness	Tempo	Duration_ms	Energy	Key	Instrumentalness	Speechiness	Valence	Danceability
0	acoustic	31.770000	-10.109200	119.453410	236517.440000	0.417889	4.920000	0.049715	0.043706	0.384144	0.549900
1	afrobeat	11.490000	-7.940680	119.634750	291627.600000	0.670631	5.700000	0.081000	0.078462	0.734760	0.659280
2	alt-rock	28.150000	-7.046410	128.581140	242760.980000	0.744190	5.060000	0.085210	0.053519	0.470889	0.469367
3	alternative	45.842105	-5.941326	120.900768	231153.905263	0.766053	5.621053	0.048122	0.050540	0.494327	0.530789
4	ambient	18.400000	-19.527770	110.091350	328447.300000	0.260270	4.450000	0.792598	0.045212	0.121041	0.299040

04

DATA ANALYSIS PROCESS

Harnessing the power of Matplotlib
and other libraries to analyze our data

6 DEPENDENCIES – SAY WHAT AGAIN!?

MATPLOTLIB

Most our charts utilised this dependency for charting static figures

SEABORN

seaborn.pairplot() was utilised for plotting pairwise relationships.

NUMPY

Functions such as np.concatenate() were utilised for coding our charts

MATH

We utilized this library for its Pi properties, whilst working with radians for our Radar plots

SKLEARN

This library was critical for building a simple line regression function

JSON

Critical to our API Calls. We required a JSON encoder / decoder.

05

CONCLUSIONS & IMPLICATIONS

What the data told us and what it means

KATHRYN'S CONCLUSIONS

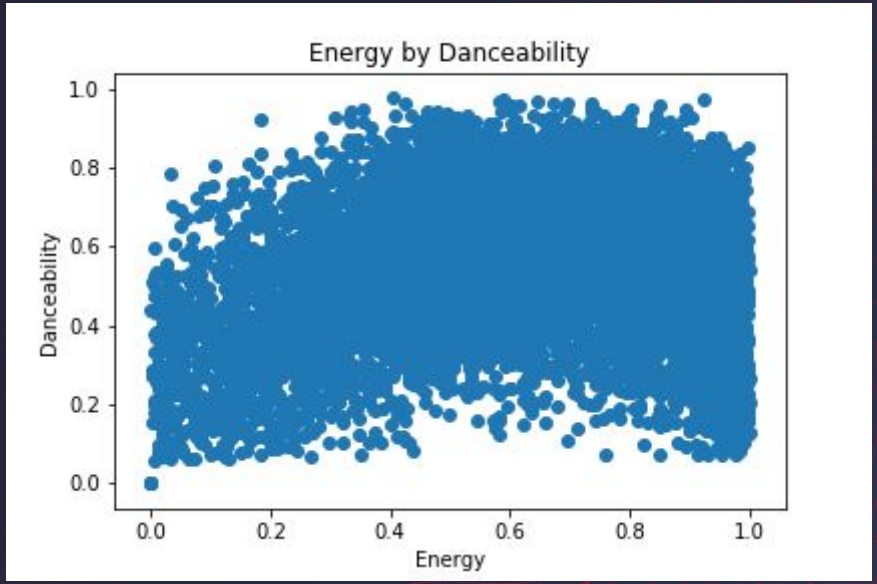
RESEARCH QUESTION I

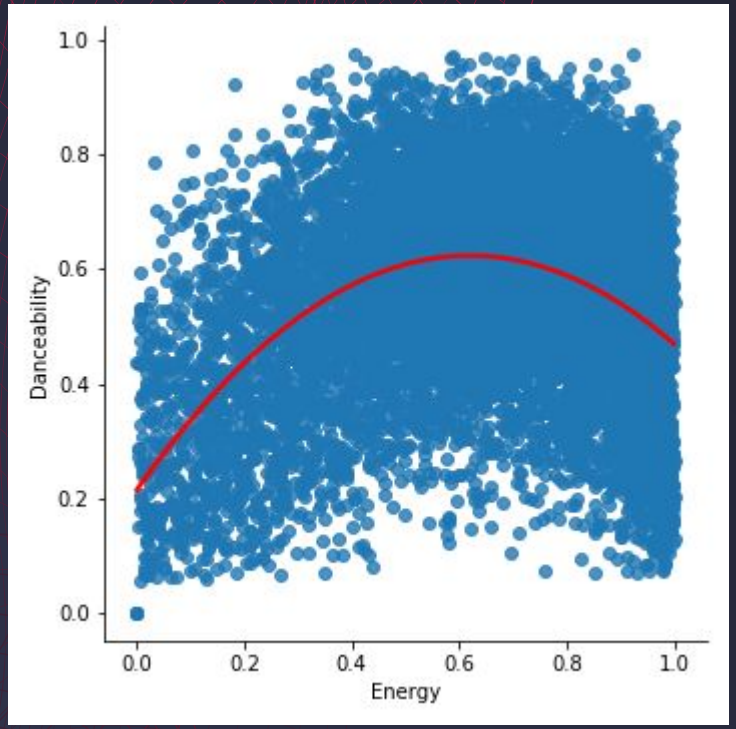
(Inductive Exploratory)

Which variables seem to have a relationship with one another in this dataset?

Observations -

"I began by plotting all variables and analyzing for correlations. Energy & Danceability were the Audio Features, that presented a clear (although condensed) relationship."





RESEARCH QUESTION 2

(After Exploration)

What is the relationship between "energy" and "danceability?"

Observations -

I identified a very quadratic scatterplot, that predictably showed a relationship between Energy & Danceability, especially as both these audio qualities are Spotify manufactured.

RESEARCH QUESTION 3

(After Exploration)

Does this relationship differ by genre?

Observations -

Re-running our API Call code showed consistent relationships.

- **Classical genre** - least ‘energetic’ and ‘danceable’.
- **Pop genre** - has the clearest curve at similar values.
- **Hip-Hop genre** - consistently the most ‘danceable’.
- Relationships between lines (highest to lowest) for each genre’s Danceability stays the same.

```
. reg energy tempo_sq tempo  loudness_sq loudness

      Source |       SS       df       MS              Number of obs =   12300
-------------+------------------------------           F(  4, 12295) = 5421.05
       Model |  486.862836     4  121.715709            Prob > F      =  0.0000
    Residual |  276.05255  12295  .022452424            R-squared     =  0.6382
-------------+------------------------------           Adj R-squared =  0.6380
       Total |  762.915386 12299  .062030684            Root MSE      =  .14984

------------------------------------------------------------------------------
      energy |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
   tempo_sq |  -.0000114   1.26e-06    -9.07   0.000    -.0000139   -8.97e-06
      tempo |   .0036047   .0003259    11.06   0.000     .0029658    .0042435
loudness_sq |   .0009083   .0000319    28.48   0.000     .0008458    .0009709
   loudness |   .0626207   .0008404    74.51   0.000     .0609733    .0642681
      _cons |    .828748   .0211051    39.27   0.000     .7873786    .8701173
------------------------------------------------------------------------------
```

- Best fit model has linear and quadratic terms for tempo and loudness
- **High correlation Energy vs. Danceability** (Danceability dropped for multicollinearity
- **R** looks awesome!
- Everything is extremely statistically significant.
- Underpins the accuracy of the prior questions charts.

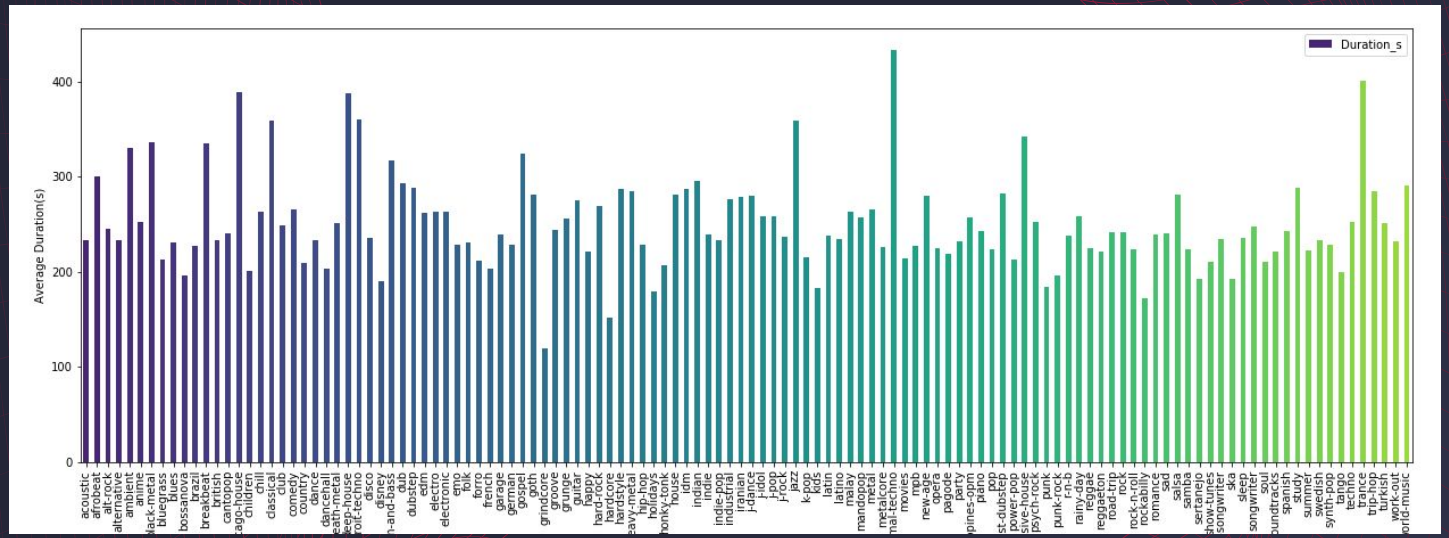
RESEARCH QUESTION 4

(After model-building)

How does Spotify define its “energy” variable?

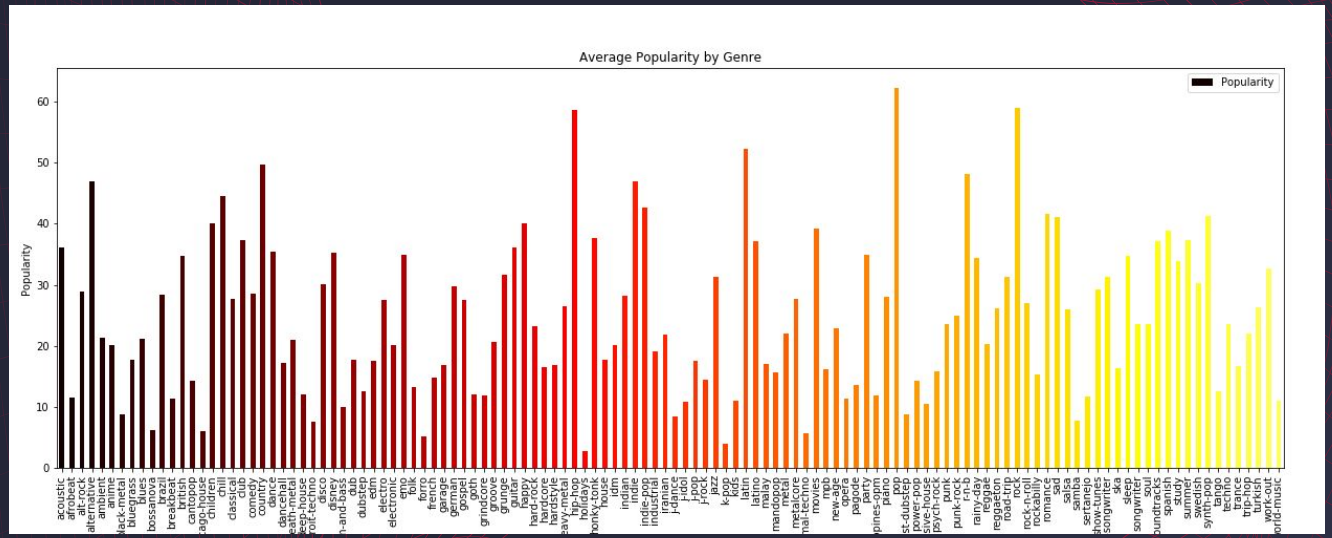
HEESUNG'S CONCLUSIONS

“WHICH GENRE USUALLY HAS THE LONGEST AND THE SHORTEST TRACK?”

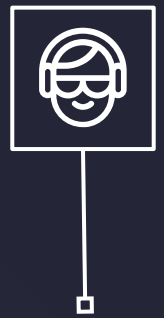


Among the 122 different genres queried, ‘minimal-techno’ had the longest time and ‘grindcore’ has the shortest time.

“WHAT IS THE MOST AND LEAST POPULAR GENRE ON SPOTIFY?”



Among 122 genres queried, Pop is the most popular and holidays is the least popular



WILLIAM'S CONCLUSIONS

Isolating required data

(DataFrame creation)

I wanted to ensure the data was complete (no *Nan* values) and convert the Audio Qualities to % for more elegant plotting.

.round ()

	Genre	Popularity	Speechiness	Danceability	Valence	Energy
10	brazil	28.42	56.12	59.97	61.69	60.34
12	british	39.10	57.96	56.29	56.50	71.25
13	cantopop	14.30	42.33	58.54	41.24	49.76
37	french	13.55	75.96	55.73	54.17	48.08
39	german	27.04	85.57	65.68	58.62	72.43
56	indian	23.50	69.32	62.63	58.17	65.45
60	iranian	21.43	77.42	56.95	59.33	58.03
70	malay	15.89	35.35	50.09	40.90	53.30
71	mandopop	16.41	38.25	54.30	37.45	51.37
81	philippines_opm	10.42	40.43	49.04	33.31	47.53
110	spanish	36.40	57.26	66.40	71.80	72.20
113	swedish	31.88	76.28	58.92	60.34	68.19
119	turkish	26.93	61.21	63.62	58.42	69.97

.isin ()

Storing & Organising

(DataFrame creation)

(1). Saved only data relevant to my questions in a new DataFrame and created lists to store Genres and Audio Features

```
# List of languages
language_genre_list = ['brazil', 'british', 'cantopop', 'mandopop', 'french', 'german', 'indian', 'ir
# List containing columns in scope.
audio_features_list = ['Genre', 'Popularity', 'Speechiness', 'Danceability', 'Valence', 'Energy']

# Dropping any Genres that do not appear in my 'language_genre_list'
language_genre_df = audiofeature_dataframe[audiofeature_dataframe['Genre'].isin(language_genre_list)]

# Only leaving those columns, for which I require data from
language_genre_df = language_genre_df[audio_features_list]
```

(2). Standardized my Audio Features units-of-measurement, by turning them into %.

```
# Turning Valence and Energy (measurement scale = 0.0 - 1.0) into percentages, by multiplying by 100
language_genre_df["Energy"] = language_genre_df["Energy"] * 100
language_genre_df["Valence"] = language_genre_df["Valence"] * 100
language_genre_df["Danceability"] = language_genre_df["Danceability"] * 100
language_genre_df["Speechiness"] = language_genre_df["Speechiness"] * 1000

# Rounding to 2-decimal places.
language_genre_df = language_genre_df.round({"Popularity":2, "Speechiness":2, "Danceability":2, "Valence":2, "Energy":2})

# changing Genre name, to avoid ambiguity
language_genre_df.loc[81, 'Genre'] = 'philippines_opm'
```

(3). Finally, created a dictionary to hold Audio Features, to be called for the Radar Charts.

```
# Iterate over each row and creating a list of lists, containing genre data
for index, rows in language_genre_df.iterrows():
    # Create list for the current row
    my_list =[rows.Popularity, rows.Speechiness, rows.Danceability, rows.Valence, rows.Energy]
    Genre_data_list.append(my_list)
Genre_data_list

# Dictionary - every key is a country and the values are audio features for tha key / country.
d = dict((country, Genre_data_list[idx]) for idx, country in enumerate(language_genre_list))
```

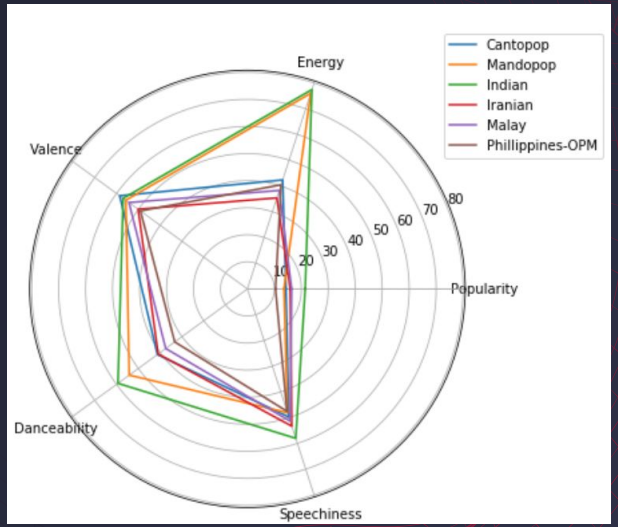
RESEARCH QUESTION I

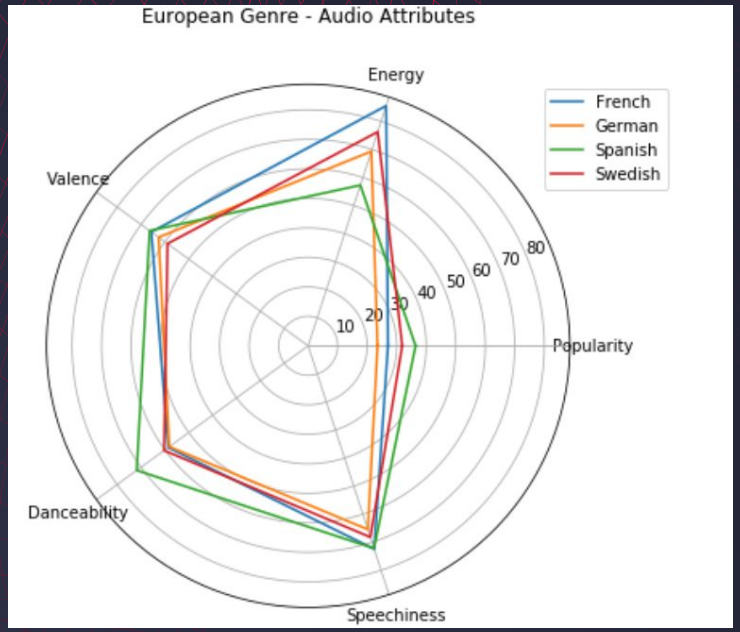
(Inductive Exploratory)

Are there clear correlations in the Audio Qualities of genres, that derive from different continents?

Observations -

Interestingly, Indian and Mandopop have double the energy of any other Asian Genre. Otherwise all the Audio Qualities show that Asian Genres have similar profiles.





- **All Genres** had **Energy** levels that spiked well above there next of kin **Audio Features.**

  Popularity should not have been included. There are too many variables that could influence how many people have access to cast a Popularity vote, such as certain Tracks only being available to certain audiences.

- **Speechiness, Danceability and Valence** seemed to consistently influence each other.

RESEARCH QUESTION 2

(After model-building)

Do Audio Qualities influence each other? E.g can a song have a low *Danceability* %, but a high *Energy Level* ?

Observations -

It is quite clear that Audio Qualities do influence each other, however as Audio Qualities are subjective, it would be intersting to understand the alcoriyth that ‘measures’ these.

CATILIN'S CONCLUSIONS

Data Exploration / Clean Up

(Part I – 2)

(1). Converted "Key" column from numeric to standard pitch class notation (music)

```
    #Convert Key Column to string and replace numeric values with Lettered pitch class
cleansed_df['Key'] = cleansed_df['Key'].astype(str)
cleansed_df['Key'].replace({"0": "C",
                            "1": "C#, Db",
                            "2": "D",
                            "3":"D#, Eb",
                            "4": "E",
                            "5": "F",
                            "6": "F#, Gb",
                            "7": "G",
                            "8": "G#, Ab",
                            "9": "A",
                            "10": "A#, Bb",
                            "11":"B"}, inplace=True)
```

(2). Organized Key counts for entire song sample in to a dataframe to plot

```
#Print a list of all available genres to choose
genre_counts = cleansed_df["Genre"].value_counts()
pd.set_option('display.max_rows', 1000)
genre_counts
```

(3). Created input function to search for specific genres

```
#input function to get genre
genre = input("Pick a genre... ")
```

(4). Organized Key counts for selected genre into dataframe to plot

```
#boolean check if genre matches input, then populates all songs within 'genre' input into new dataframe
genre_check = cleansed_df['Genre']== genre
g_check = cleansed_df[genre_check]
genre_df = pd.DataFrame(g_check)
```

Data Exploration / Clean Up

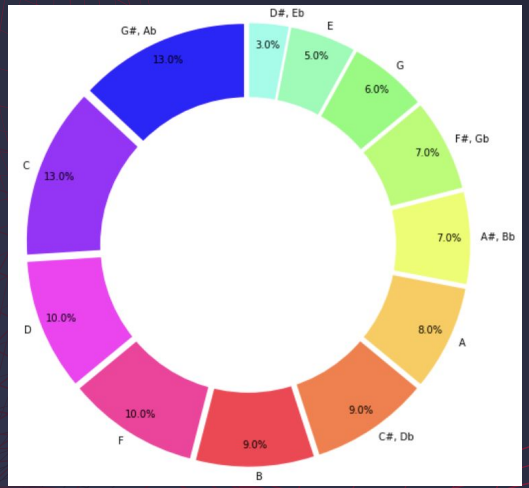
(Part 2 – 2)

(5). Formatted a donut chart out of a pie plot by adding a white center circle to the figure

```
#Create center circle for donut chart
center_circle = plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(center_circle)
```

(6). Genre-specific plots have auto-generated titles and png file names according to the selected genre

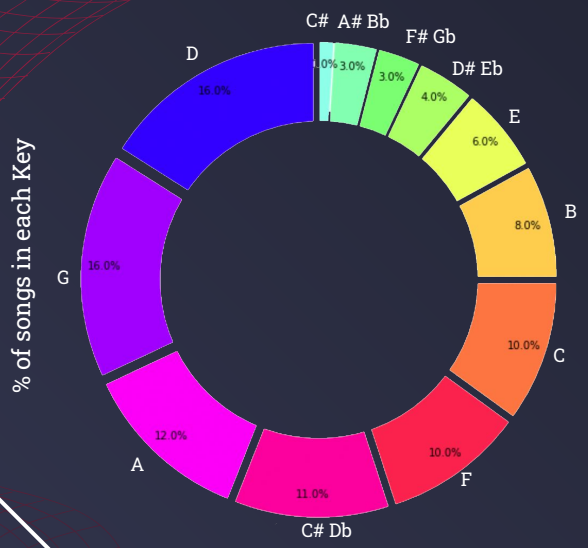
```
#Format Title and Labels and save
plt.title('Song Key Percentages of ' + genre + ' Genre', fontsize=20)
plt.ylabel('% of Songs in Each Key', fontsize=16)
plt.tight_layout()
plt.savefig('outputs/' + genre + '_keys.png')
```



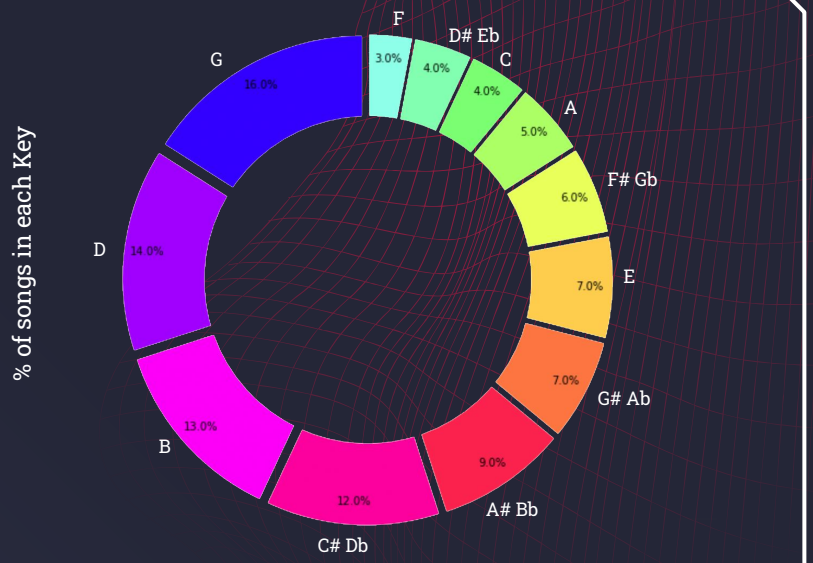
“HOW DOES ‘KEY’ VARY ACROSS GENRES?”

Key charts for a few popular and differing genres:

Song Key Percentages of classical Genre

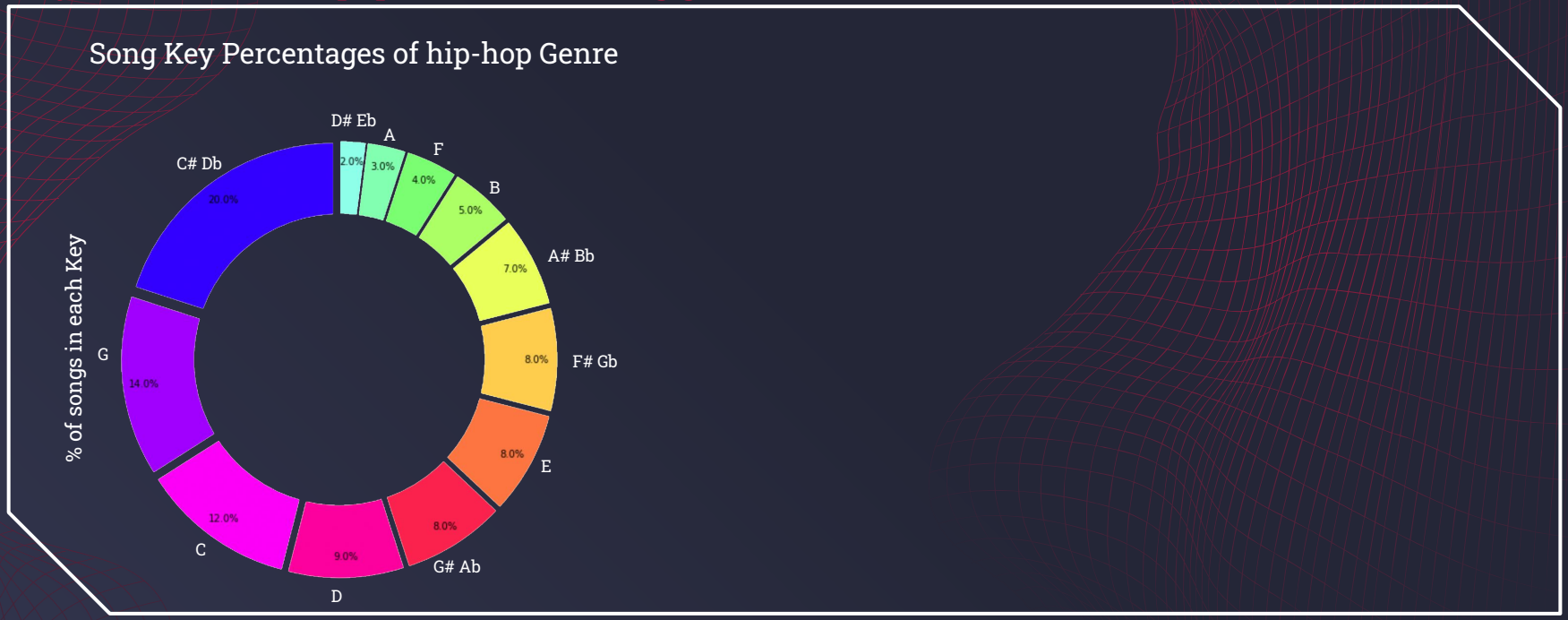


Song Key Percentages of death-metal Genre



“HOW DOES ‘KEY’ VARY ACROSS GENRES?”

Key charts for a few popular and differing genres:

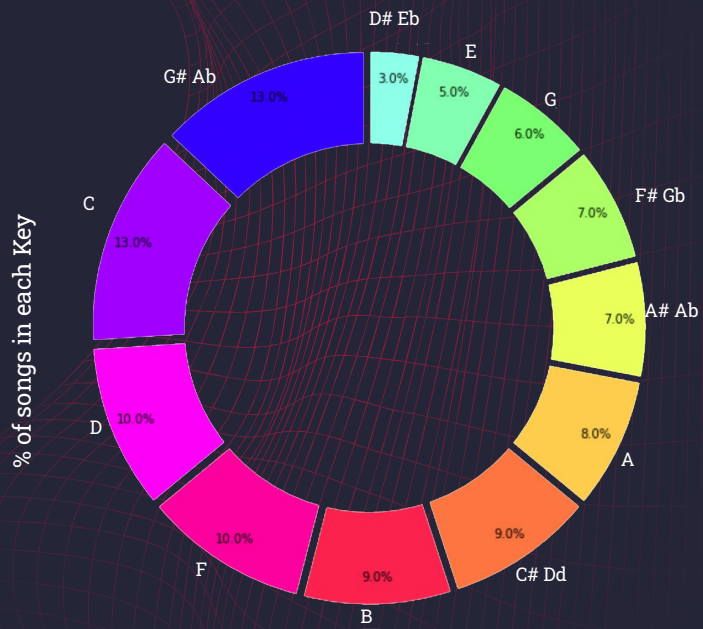


OBSERVATIONS

- Genres seem to favor 1-3 dominant keys
- The distribution of songs across keys is genre dependent
- When looking at key distribution while ignoring genre, there is a much more even dispersal of songs.

QUESTIONS FOR FURTHER EXPLORATION:

- Does the key of a song have an impact on it's mood?
- Is there a relationship between the predominant keys of a genre and it's average valence score?



Thank you

Does anyone have any questions?