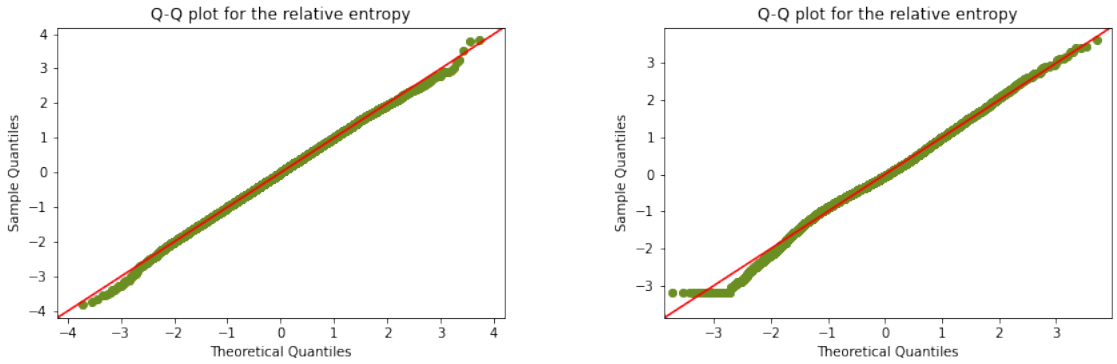
Appendices

A Lost Artists names due to the duplicate removal

Chris Farlow, D.J.Generous, Jermaine Dupri featuring Jay-Z, Lilly Allen, Lito y Manolo - The Gypsies, Meck ft. Dino, Oasis And Friends Inc Johnny Depp, Paola Dipietromaria, RUN-DMC Featuring Method Man, Kenny Cash, Mike Ransom, and Jamel Simmons, Richard Hawley And Death Ramps, Arctic Monkeys, Shakira ft. Wyclef Jean, Tineke Schouten/Linda De Mol/Franklin Brown, Tony Marshall.

B Assessing normality of scores

Standardising the song-level relative entropy and artist-level Shannon entropy scores and plotting their quantiles against the theoretical quantiles of the standard normal distribution yielded the following Q-Q plots.



(a) Song level relative entropy

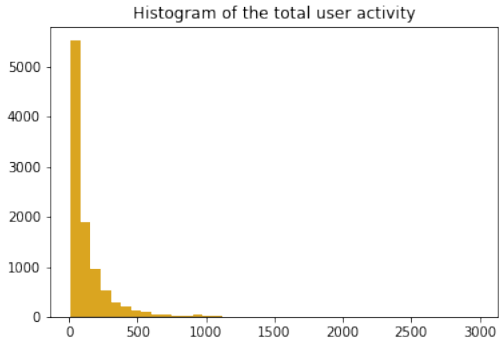
(b) Artist level Shannon entropy

Figure 1: Q-Q plots

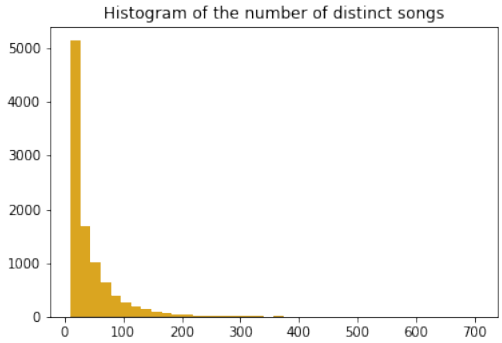
The quantiles of the relative entropy scores seem to follow theoretical quantiles very well with some minor deviation on the tails and close to an ideal fit around the majority of the mass of the standard normal distribution. Thus, it can be assumed that the values of the relative entropy calculated on the song level are approximately distributed as the normal distribution.

Similarly, the quantiles of the artist-level Shannon entropy scores also seem to follow the standard normal distribution well with some larger deviance at the left tail. Thus, with a certain degree of tolerance, we could consider artist-level Shannon entropy scores to also approximately follow the normal distribution.

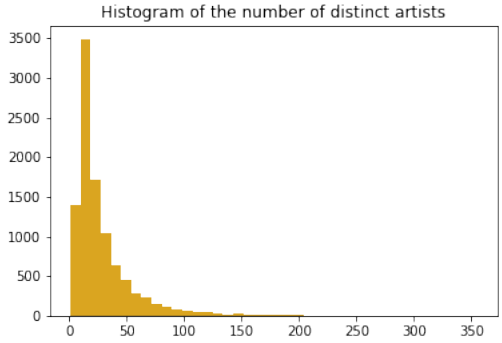
C Histograms of the attributes



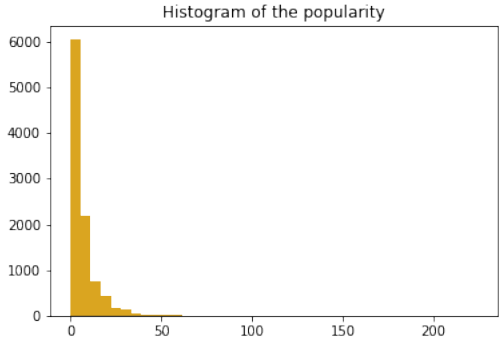
(a) Total user activity



(b) Distinct songs



(c) Distinct artists

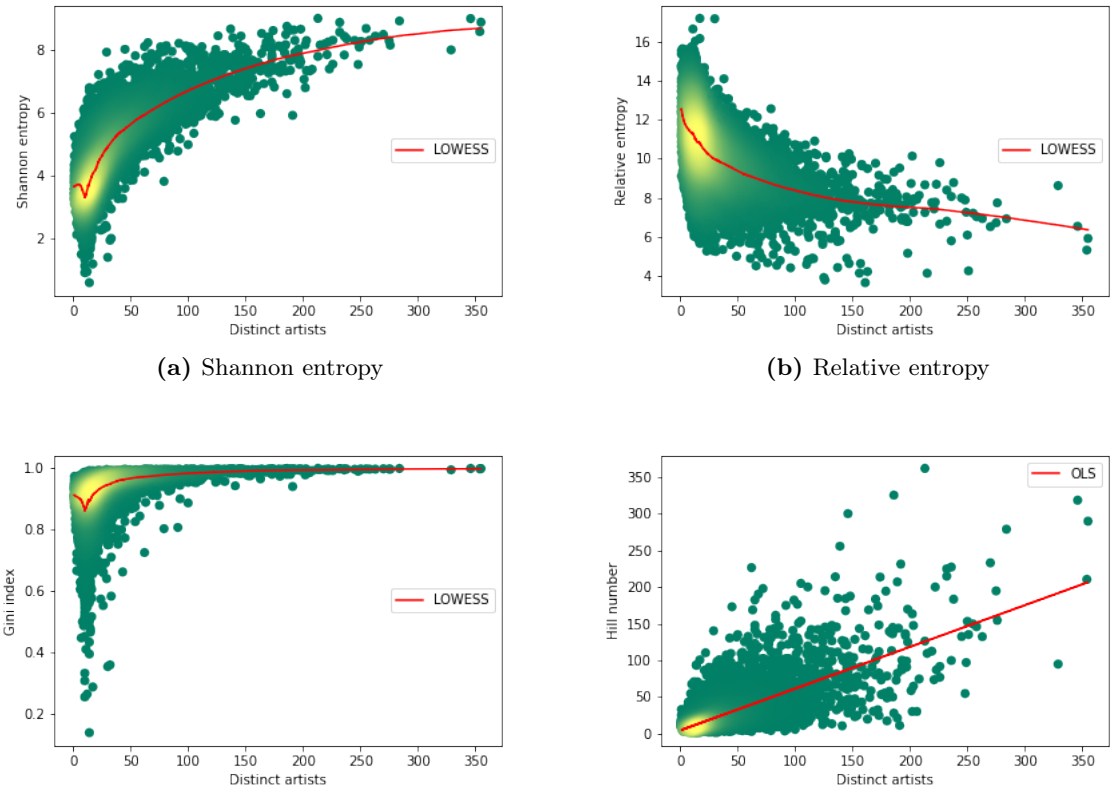


(d) Popularity

Figure 2: Histograms of the attributes

D Statistical diversity measures vs Distinct Artists

The dependence of the statistical diversity measures on the number of distinct artist seems to be very similar but slightly noisier and weaker as compared to the dependence on the *Distinct Songs*. Nonetheless, all of the relationships indicate that users with a larger number of distinct artists in the playlist tend to be more generalist or mainstream generalists.



(a) Shannon entropy

(b) Relative entropy

(c) Gini index

(d) Hill number

Figure 3: Diversity measures vs Distinct Artists

E Discussed User Playlists

All the discussed playlists are provided here. However, note that due to the matching issues of the Million Song Datasets, some `Songs` and `Artists` refers to a different songs and artists than stated. For example, an artist *Harmonia* in reality corresponds to *Katy Perry*.

Count	Artist	Song
5	Harmonia	Sehr kosmisch
25	Björk	Undo
3	Florence + The Machine	Dog Days Are Over (Radio Edit)
20	Dwight Yoakam	You're The One
1	Train	Hey_ Soul Sister
1	Charttraxx Karaoke	Fireflies
3	OneRepublic	Secrets
15	Kings Of Leon	Revelry
1	Train	Marry Me
1	Usher featuring will.i.am	OMG
1	Lil Wayne / Eminem	Drop The World

Table 1: Playlist of the user 5703 (B)

Count	Artist	Song
1	Björk	Undo
1	Dwight Yoakam	You're The One
1	OneRepublic	Secrets
1	Kings Of Leon	Revelry
1	Sam Cooke	Ain't Misbehavin
1	The Killers	The Ballad of Michael Valentine
1	The Killers	When You Were Young
1	Edward Sharpe & The Magnetic Zeros	Home
1	The Killers	Smile Like You Mean It
1	The Killers	Romeo And Juliet
1	The Killers	This Is Your Life
1	The Killers	Under The Gun
1	The Killers	A Dustland Fairytale

Table 2: Playlist of the user 3,075 (A)

Artist	Count
Amy Winehouse	7
Beyoncé	2
Boys Noize	4
Caribou (formerly Dan Snaith's Manitoba)	1
Coldplay	3
Curved Air	4
Cut Copy	7
Daft Punk	1
Deadmau5	1
Faith No More	1
Flying Lotus	5
Holy Fuck	4
Jonas Brothers	7
Justice	5
Justin Timberlake	1
Kanye West	1
Katy Perry	1
Kelly Clarkson	2
LCD Soundsystem	5
Lady GaGa	3
Lily Allen	2
MSTRKRFT	1
Mariah Carey / Twista	1
Miley Cyrus	4
OceanLab	41
Octopus Project	3
Plies	2
Rihanna	2
Skream	3
Steely Dan	1
The All-American Rejects	3
The Killers	7
The Pussycat Dolls	3
...	

Table 3: Subsample (33/50 artists) of the playlist of the user 6,918 (C)

Artist	Count
Björk	1
Brant Bjork	1
Busta Rhymes / Swizz Beatz	1
Carlos Santana & Mahavishnu John McLaughlin	2
Cartola	1
Cirrus	1
Counting Crows	1
Céline Dion	1
Dwight Yoakam	1
Elbow	1
Elton John	1
Emilio Navaira	1
Ennio Morricone	1
Enrique Iglesias	1
Florence + The Machine	1
George Lopez	2
Jerry Rivera	2
Jorge Alfano	1
Juanes	76
Juanes / Nelly Furtado	7
Just Jack	2
Limp Bizkit	1
MGMT	1
Maelo Ruiz	4
Marc Anthony	11
Marc Anthony;Jennifer Lopez	5
Metro Area	1
Natalie Cole	7
Nathan Fake	1
Neri Per Caso	1
Nick Cave	2
Nick Drake	1
Portishead	7
Rex Mundi	1
Ruben Blades	2
Rubin Steiner	1
Rubén Blades	4
Saga	1
The Police	1
Thievery Corporation	2
Vilma Palma e Vampiros	2
...	

Table 4: Subsample (40/79 artists) of the playlist of the user 8,536 (E)

Count	Artist	Song
1	Kim Carnes	Bette Davis Eyes
1	Westlife	You Raise Me Up
1	The Grateful Dead	Peggy-O [Studio Outtake]
1	Bishop Allen	Shanghaied
1	Kylie Minogue	This Girl
1	Harold Budd	Gypsy Violin
1	The Hold Steady	Chips Ahoy!
1	Lords Of The New Church	Dreams & Desires
1	Animals on Wheels	Build A Church With Your Fear
1	2 In A Room	Wiggle It (Radio Version)
1	Westlife	My Girl
1	The Pains Of Being Pure At Heart	Falling Over (DJ Downfall Sprechenbann Mix)
1	Don McLean	In A Museum
1	The Cure	M
1	Army Navy	Jail Is Fine
1	Code Of Ethics	Without Reason (Trance Dub)
1	Stella Starlight Trio	Don't You (Forget About Me)
1	Soulfly	Moses (With Sample)
1	David Bowie	John_ I'm Only Dancing (1990 Digital)
1	Jimmy Page	Prison Blues

Table 5: Playlist of the user 8,243 (D)

Count	Artist	Song
1	Harmonia	Sehr kosmisch
1	Justin Bieber	Somebody To Love
1	Björk	Undo
1	Florence + The Machine	Dog Days Are Over (Radio Edit)
2	Train	Hey_ Soul Sister
3	Charttraxx Karaoke	Fireflies
3	OneRepublic	Secrets
1	Kings Of Leon	Revelry
1	Usher featuring will.i.am	OMG
1	Lil Wayne / Eminem	Drop The World
1	Kings Of Leon	Use Somebody
1	Linkin Park	Bleed It Out [Live At Milton Keynes]
1	Adam Lambert	Whataya Want From Me

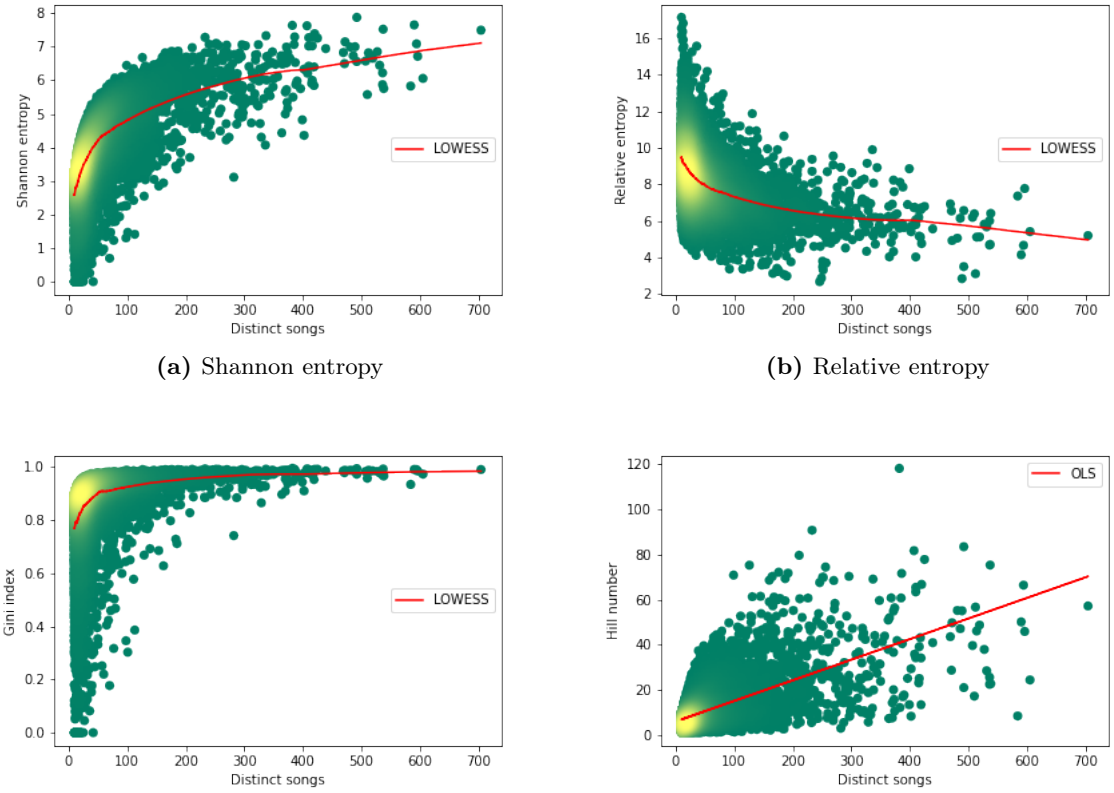
Table 6: Playlist of the user 9,416 (G)

Artist	Count
10 Years	2
4hero	1
A Perfect Circle	9
American Hi-Fi	15
Ben Folds	8
Benny Benassi	11
Bijelo Dugme	1
Bill Wells & Maher Shalal Hash Baz	5
Box Car Racer	9
Crystal Castles	11
D-12	1
DAVE MATTHEWS BAND	2
DJ Khaled	9
DJ Sammy & Yanou Featuring Do	1
Daft Punk	1
Dashboard Confessional	10
Hoobastank	22
Jack Johnson	31
Jimmy Eat World	26
Justice	11
Kanye West	12
Lights	18
Morgan Page	15
New Found Glory	10
Placebo	10
SAMULI PUTRO	10
Silversun Pickups	17
Skye Sweetnam	6
Snow Patrol	35
Tarot	27
Yellowcard	37
...	

Table 7: Subsample (30/226 artists) of the playlist of the user 8,600 (F)

F Hierarchical statistical diversity measures vs Distinct Songs

The dependence of the hierarchical statistical diversity measures on the number of distinct songs is weaker than with the *Distinct Artists*. This is expected since all the measures consider the number of distinct songs only via how many of those songs are played by a certain artist.



(a) Shannon entropy

(b) Relative entropy

(c) Gini index

(d) Hill number

Figure 4: Diversity measures vs Distinct Songs

G Non-normalised TS-SS score vs the fraction of popular songs

There seems to be a mild positive relationship between the $D^{TS-R}(S_i)$ and the fraction of popular songs in the playlist (Figure 21). However, high values of the fraction variable cover the full range of the $D^{TS-R}(S_i)$ values leading to a very pronounced *triangle-pattern*. Thus, people who listen to relatively few popular songs do not yield high values of the $D^{TS-R}(S_i)$ and thus are closer to the specialists according to the score, thus users with a high $D^{TS-R}(S_i)$ score are likely to have a high ratio of popular songs in their playlist. The reverse conclusion cannot be made.

The Figure 21 shows that the $D^{TS-R}(S_i)$ score has more to itself than a mere fraction of popular songs. At the same time, this is the attribute the score correlates the most with.

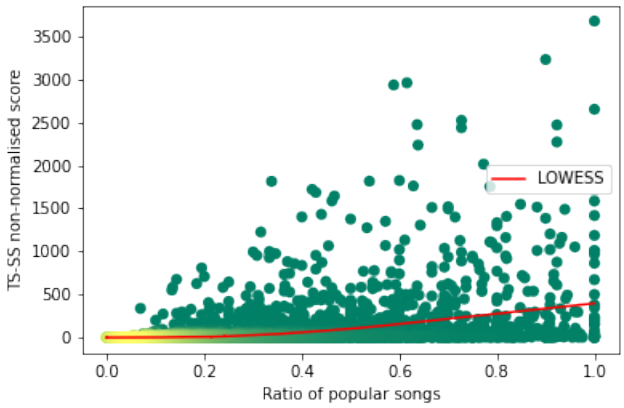
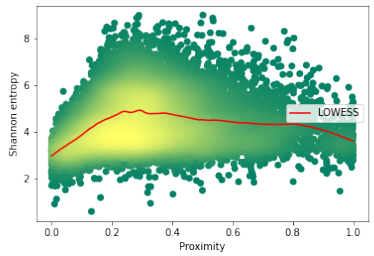
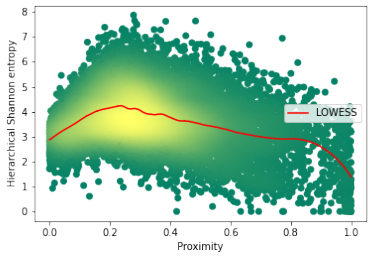


Figure 5: Non-normalised TS-SS score vs the fraction of popular songs

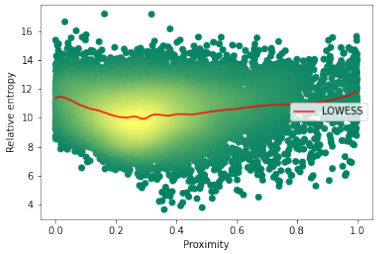
H Scatter plots between the diversity scores and the Proximity



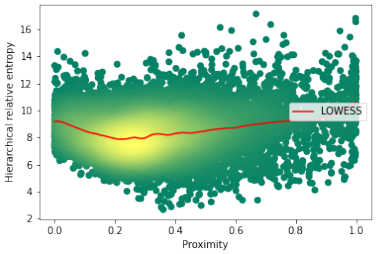
(a) Shannon entropy



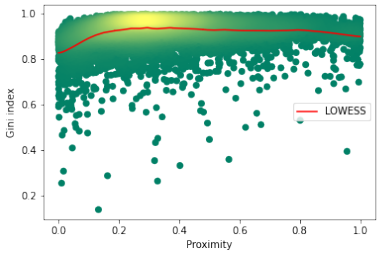
(b) Hierarchical Shannon entropy



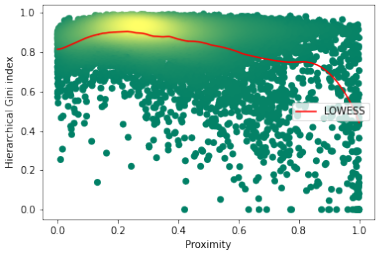
(c) Relative entropy



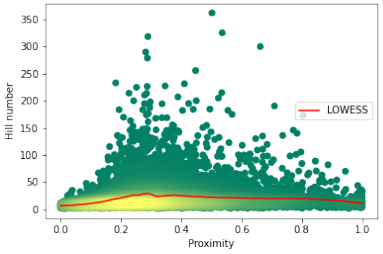
(d) Hierarchical relative entropy



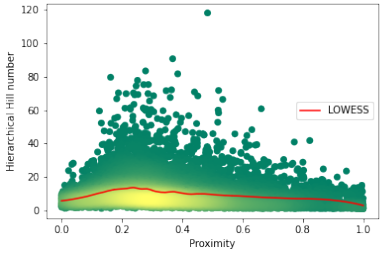
(e) Gini index



(f) Hierarchical Gini index



(g) Hill number



(h) Hierarchical Hill number

Figure 6: Diversity measures vs Proximity

I Code

Listing 1: Creating subsample

```
# Load triplet and bridge data
# dropping duplicate values
bridge.drop_duplicates(subset = "SongId", inplace=True)
# merging the triplets with bridge data
main_merge = pd.merge(left = triplets, right = bridge, left_on="SongId"
    , right_on="SongId")

# Subsampling 10,000 user data
toy_triplets = pd.DataFrame(main_merge['UserId'].unique(), columns=["
    UserId"]).sample(n=10000, random_state=123)
toy_triplets = pd.merge(left = toy_triplets, right = main_merge,
    left_on="UserId", right_on="UserId")
```

Listing 2: Implementation of the statistical diversity scores

```
#### Shannon entropy function
def entropy_me(p):
    c = []
    for i in p:
        if i == 0:
            b = 0
        else:
            b = -i*math.log2(i)
        c.append(b)
    d = sum(c)
    return d

# getting a list of unique UserIds
users = toy_triplets["UserId"].unique().tolist()

# calculating the Shannon entropy for each user separately and stacking
# everything into the list "shannon"
shannon = []
for i in tqdm(users):
  b = toy_triplets["Count"][toy_triplets["UserId"]==i]/(np.sum(
      toy_triplets["Count"][toy_triplets["UserId"]==i]))
  #a = entropy_me(b)
```

```
    a = entropy(b, base = 2)
    shannon.append(a)

#### Kullback-Leibler divergence function
# Must use it since the inbuilt entropy command normalises q by default
def KL(p,q):
    c = []
    for i, j in zip(p,q):
        if i == 0:
            b = 0
        else:
            b = i*math.log2(i/j)
        c.append(b)
    d = sum(c)
    return d

# getting Q - a user who listens only to popular songs

# define q^1 - each instance of song being played at least once
    corresponds to
# a one count in q^1
sum_song = toy_triplets.groupby(["SongId"]).size()
sum_song = sum_song.to_frame()
sum_song["SongId"] = sum_song.index
sum_song.columns = ["Count_q", "SongId"]
sum_song.index.names = ["Index"]

Q_max = sum_song.sort_values(by = "Count_q", ascending=False)[:500]

# getting q_j as probability
Q_joint = sum_song
Q_joint["Count_q"] = Q_joint["Count_q"]/np.sum(Q_joint["Count_q"])

# calculating the KL divergence with Q
relative = []
for u in tqdm(users):
  # subset the Count and SongID data for each user
  p = toy_triplets[toy_triplets["UserId"] == u]
  # merge with Q so that the absent songs would disappear (p_ij = 0
```

```
    contributes
  # 0 to KL, hence it is equivalent to discard it a priori)
  mid = pd.merge(left = p, right = Q_joint, left_on = "SongId",
                 right_on = "SongId")
  # defining a p_ij variable as a percentage
  a = mid["Count"]/np.sum(mid["Count"])
  # KL function aggregate
  relative.append(KL(a, mid["Count_q"]))

#### Gini function
def gini(p):
    c = []
    for i in p:
        b = i**2
        c.append(b)
    d = 1 - sum(c)
    return d

# getting gini index for each user
gini_index = []
for i in tqdm(users):
  b = toy_triplets["Count"][toy_triplets["UserId"]==i]/(np.sum(
      toy_triplets["Count"][toy_triplets["UserId"]==i]))
  a = gini(b)
  gini_index.append(a)

#### Hill number function
def hill(p,q):
    c =[]
    for i in p:
        b = i**q
        c.append(b)
    d = sum(c)**(1/(1-q))
    return d

# implementation for each user and then aggregating in list "hill_no"
hill_no = []
for i in users:
  b = toy_triplets["Count"][toy_triplets["UserId"]==i]/(np.sum(
```

```
    toy_triplets["Count"][toy_triplets["UserId"]==i]))
  a = hill(b,3)
  hill_no.append(a)
```

Listing 3: Creating attributes and function to plot relationships

```
####Total user activity
total_playcount = []
for i in users:
  total_playcount.append(np.sum(toy_triplets["Count"][toy_triplets["
      UserId"]==i]))
# np.save("total_playcount", total_playcount)

#### song coverage
distinct_songs = toy_triplets.groupby(["UserId"]).size()
distinct_songs = pd.merge(left = pd.DataFrame(users, columns = ["UserId
    "]), right = distinct_songs.to_frame(),
                              left_on = "UserId", right_on =
                                  distinct_songs.to_frame().index)
np.save("distinct_songs", distinct_songs)

#### artist coverage
distinct_artists = []
for i in users:
  a = len(toy_triplets["Artist"][toy_triplets["UserId"]==i].unique().
      tolist())
  distinct_artists.append(a)
#np.save("distinct_artists", distinct_artists)

#### popularity score
pop_score = []
for i in tqdm(users):
  b = []
  a = toy_triplets["SongId"][toy_triplets["UserId"]==i].values
  for j in Q_max["SongId"]:
    if j in a:
      b.append(1)
    else:
      b.append(0)
  pop_score.append(np.sum(b))
```

```
# function to plot relationships with LOWESS
def plot(p,q, xlab, ylab, filename, frac=0.2):
  # Plots scatterplot with mass gradient and LOwESS
  # p - x variable
  # q - y variable

  # stacking data for the density gradient
  xy = np.vstack([p,q])
  # calling
  z = gaussian_kde(xy)(xy)

  idx = z.argsort()
  x, y, z = p[idx], q[idx], z[idx]

  df = pd.DataFrame({'x': x, 'Raw': y})
  df["pred"] = lo.lowess(df["Raw"], df["x"], frac, 2)

  sorted_index = np.argsort(df["Raw"])
  X_sort = df["Raw"][sorted_index]
  y_sort = df["pred"][sorted_index]

  fig, ax = plt.subplots()
  # main scatter
  ax.scatter(df["Raw"], df["x"], c=z, s=50, edgecolor='', cmap=plt.cm.
      summer)
  # LOWESS
  ax.plot(X_sort, y_sort, "r", label="LOWESS")
  leg = ax.legend(loc=7)
  plt.xlabel(xlab)
  plt.ylabel(ylab)
  plt.savefig(filename, format = "png", transparent = True)
  plt.show()

# function to plot relationships with OLS

def plot_hill(p,q, xlab, ylab, filename):
  # p - x variable
  # q - y variable
```

```
  xy = np.vstack([p,q])
  z = gaussian_kde(xy)(xy)

  idx = z.argsort()
  x, y, z = p[idx], q[idx], z[idx]

  m, b = np.polyfit(p, q, 1)

  fig, ax = plt.subplots()
  ax.scatter(x, y, c=z, s=50, edgecolor='', cmap=plt.cm.summer)
  ax.plot(p, m*p + b, label="OLS", color="r")
  leg = ax.legend(loc=1)
  plt.xlabel(xlab)
  plt.ylabel(ylab)
  plt.savefig(filename, format = "png", transparent = True)
  plt.show()
```

Listing 4: Implementation of hierarchical statistical diversity measures

```
# getting a list of unique UserIds
users = toy_triplets["UserId"].unique().tolist()

#### Shannon entropy
# calculating the Shannon entropy for each user separately and stacking
# everything into the list "shannon"
shannon = []
for i in tqdm(users):
  un_art = toy_triplets[toy_triplets["UserId"]==i].groupby(["Artist"])[
      "Count"].sum()
  p = un_art/np.sum(toy_triplets["Count"][toy_triplets["UserId"]==i])
  #a = entropy_me(b)
  a = entropy(p, base = 2)
  shannon.append(a)

##### KL divergence
# defining Q.
big_list = []
for u in tqdm(users):
  # How many distinct artist a user is listening to
```

```
    art_unique_user = toy_triplets["Artist"][toy_triplets["UserId"]==u].
        unique().tolist()
    # stacking all of these users together
    big_list = np.hstack([big_list, np.array(art_unique_user)])

# the total number of time artist was mentioned is q_j
sum_art = pd.DataFrame(big_list, columns=["Artists"]).groupby(["Artists
    "]).size().to_frame()

# cleaning for better representation
sum_art["Artist"] = sum_art.index
sum_art.columns = ["Count_q", "Artist"]
sum_art.index.names = ["Index"]

# getting q_j as probability
Q_joint_art = sum_art
Q_joint_art["Count_q"] = Q_joint_art["Count_q"]/np.sum(Q_joint_art["
    Count_q"])

# calculating actual KL divergence
relative = []
for u in tqdm(users):
    # subset the Count and SongID data for each user
    p = toy_triplets[toy_triplets["UserId"] == u]
    # summing the counts of distinct songs by the same artist and getting
        p_ij
    a = p.groupby(["Artist"])["Count"].sum()/np.sum(p["Count"])
    # merge with Q so that the absent artists would disappear (p_ij = 0
        contributes
    # 0 to KL, hence it is equivalent to discard it a priori)
    mid = pd.merge(left = a, right = Q_joint_art, left_on = "Artist",
                   right_on = "Artist")
    # KL function aggregate
    relative.append(KL(mid["Count"], mid["Count_q"]))

    #### Gini index

gini_index = []
for i in tqdm(users):
```

```
    un_art = toy_triplets[toy_triplets["UserId"]==i].groupby(["Artist"])[
        "Count"].sum()
    p = un_art/np.sum(toy_triplets["Count"][toy_triplets["UserId"]==i])
    a = gini(p)
    gini_index.append(a)

    #### Hill number

hill_no= []
for i in tqdm(users):
    un_art = toy_triplets[toy_triplets["UserId"]==i].groupby(["Artist"])[
        "Count"].sum()
    p = un_art/np.sum(toy_triplets["Count"][toy_triplets["UserId"]==i])
    a = hill(p, 3)
    hill_no.append(a)
```

Listing 5: Implementation of geometrical diversity scores

```
# load the vectors and call them embeds.
# embeds["beta"] - the vectors
# embeds["mapping"] - SongId

# normalizing the vectors to have a unit norm
emb_norm = preprocessing.normalize(embeds["beta"], norm='l2')
# moving to pandas to exploit indexing
pan_emb = pd.DataFrame(emb_norm)

# moving to pandas raw vectors too
pan_emb_nonorm = pd.DataFrame(embeds["beta"])

# check which songs are remaining in the subsample and leaving only
    those
songs = toy_triplets["SongId"].unique().tolist()
songs_pd = pd.DataFrame(songs, columns=["SongId"])
embeds_songs = pd.DataFrame(embeds["mapping"], columns=["SongId"])
embeds_songs["Location"] = embeds_songs.index

embeds_songs = pd.merge(left = embeds_songs, right = songs_pd, left_on=
    "SongId", right_on="SongId")
```

```
#### Centroid
def centroid(p,w):
  # assume we have a joint np array.
  # Joint means all song vectors are stacked vertically
  # w gotta be a n_i x 1 np array
  weighted = p * w
  joint = weighted.sum(axis=0)
  return joint/np.sum(w)

# normalised centrois
centre = []
for i in tqdm(users):
  songs_loop = toy_triplets[["SongId", "Count"]][toy_triplets["UserId"
      ]==i]
  # merging with the location as found in the embeds["mapping"]
  loc_merge = pd.merge(left = embeds_songs, right = songs_loop, left_on
      ="SongId", right_on="SongId")
  loc_temp = loc_merge["Location"].tolist()
  # those location correspond the the right songs. Extracting them
  extract_vec = pan_emb.iloc[loc_temp]
  # function centroid requires weight vector to be a column vector
  weights = songs_loop["Count"].to_numpy().reshape(-1,1)
  # appending function values
  centre.append(centroid(extract_vec.to_numpy(), weights))

 # non-normalised centroid
 centre_nonorm = []
for i in tqdm(users):
  songs_loop = toy_triplets[["SongId", "Count"]][toy_triplets["UserId"
      ]==i]
  # merging with the location as found in the embeds["mapping"]
  loc_merge = pd.merge(left = embeds_songs, right = songs_loop, left_on
      ="SongId", right_on="SongId")
  loc_temp = loc_merge["Location"].tolist()
  # those location correspond the the right songs. Extracting them
  extract_vec = pan_emb_nonorm.iloc[loc_temp]
  # function centroid requires weight vector to be a column vector
  weights = songs_loop["Count"].to_numpy().reshape(-1,1)
```

```
  # appending function values
  centre_nonorm.append(centroid(extract_vec.to_numpy(), weights))

#### Generalist-Specialist score
# Squared norm of the centroid is the GS score
GS = []
for x in centre:
  GS.append(np.linalg.norm(x))

#### TS-SS score

import math
import numpy as np

class TS_SS:

    def Cosine(self, vec1: np.ndarray, vec2: np.ndarray):
        return np.dot(vec1, vec2.T)/(np.linalg.norm(vec1) * np.linalg.
            norm(vec2))

    def VectorSize(self, vec: np.ndarray):
        return np.linalg.norm(vec)

    def Euclidean(self, vec1: np.ndarray, vec2: np.ndarray):
        return np.linalg.norm(vec1-vec2)

    def Theta(self, vec1: np.ndarray, vec2: np.ndarray):
        return np.arccos(self.Cosine(vec1, vec2)) + np.radians(10)

    def Triangle(self, vec1: np.ndarray, vec2: np.ndarray):
        theta = np.radians(self.Theta(vec1, vec2))
        return (self.VectorSize(vec1) * self.VectorSize(vec2) * np.sin(
            theta))/2

    def Magnitude_Difference(self, vec1: np.ndarray, vec2: np.ndarray):
        return abs(self.VectorSize(vec1) - self.VectorSize(vec2))

    def Sector(self, vec1: np.ndarray, vec2: np.ndarray):
        ED = self.Euclidean(vec1, vec2)
```

```
        MD = self.Magnitude_Difference(vec1, vec2)
        theta = self.Theta(vec1, vec2)
        return math.pi * (ED + MD)**2 * theta/360


    def __call__(self, vec1: np.ndarray, vec2: np.ndarray):
        return self.Triangle(vec1, vec2) * self.Sector(vec1, vec2)


 # TS-SS score implementation - normalized data
TSSS2 = []
for i, x in tqdm(zip(users, centre)):
  # data management
  songs_loop = toy_triplets[["SongId", "Count"]][toy_triplets["UserId"
      ]==i]
  # merging with the location as found in the embeds["mapping"]
  loc_merge = pd.merge(left = embeds_songs, right = songs_loop, left_on
      ="SongId", right_on="SongId")
  loc_temp = loc_merge["Location"].tolist()
  # those location correspond the the right songs. Extracting them
  extract_vec = pan_emb.iloc[loc_temp]
  # function centroid requires weight vector to be a column vector
  weights = songs_loop["Count"].to_numpy()

# TS-SS starts here
  a = []
  for j, w in zip(extract_vec.to_numpy(), weights):
    #call a class
    tsss = TS_SS()
    a.append(w *tsss(j,x))
  joint = np.sum(a)/np.sum(weights)
  TSSS2.append(joint)



  # TS-SS raw score implementation
TSSS2 = []
for i, x in tqdm(zip(users, centre)):
  # data management
  songs_loop = toy_triplets[["SongId", "Count"]][toy_triplets["UserId"
```

```
        ]==i]
    # merging with the location as found in the embeds["mapping"]
    loc_merge = pd.merge(left = embeds_songs, right = songs_loop, left_on
        ="SongId", right_on="SongId")
    loc_temp = loc_merge["Location"].tolist()
    # those location correspond the the right songs. Extracting them
    extract_vec = pan_emb_nonorm.iloc[loc_temp]
    # function centroid requires weight vector to be a column vector
    weights = songs_loop["Count"].to_numpy()

# TS-SS starts here
    a = []
    for j, w in zip(extract_vec.to_numpy(), weights):
      #call a class
      tsss = TS_SS()
      a.append(w *tsss(j,x))
    joint = np.sum(a)/np.sum(weights)
    TSSS2.append(joint)
```

Listing 6: Recommender system prototype

```
#### Creating train/test data

train = pd.DataFrame(columns=["UserId", "SongId", "Count", "Artist", "
    Song"])
test = pd.DataFrame(columns=["UserId", "SongId", "Count", "Artist", "
    Song"])
for i in tqdm(users2):
    a = toy_triplets[["UserId", "SongId", "Count", "Artist", "Song"]][
        toy_triplets["UserId"]==i]

    shuffle_df = a.sample(frac=1)
    train_size = int(0.8 * len(a))
    train_one = shuffle_df[:train_size]
    test_one = shuffle_df[train_size:]

    train = pd.concat([train, train_one])
    test = pd.concat([test, test_one])
```

```
### Getting user preference vector

centre_train = []
for i in tqdm(users2):
  songs_loop = train[["SongId", "Count"]][train["UserId"]==i]
  # merging with the location as found in the embeds["mapping"]
  loc_merge = pd.merge(left = embeds_songs, right = songs_loop, left_on
     ="SongId", right_on="SongId")
  loc_temp = loc_merge["Location"].tolist()
  # those location correspond the the right songs. Extracting them
  extract_vec = pan_emb.iloc[loc_temp]
  # function centroid requires weight vector to be a column vector
  weights = songs_loop["Count"].to_numpy().reshape(-1,1)
  # appending function values
  centre_train.append(centroid(extract_vec.to_numpy(), weights))

 ### Computing "Proximity"

 from sklearn.metrics.pairwise import cosine_similarity

proximity = []
for i, mu in tqdm(zip(users2, centre_train)):
  # merging the test SongId with the embedded vectors
  songs_loop = test[test["UserId"]==i]
  loc_merge = pd.merge(left = embeds_songs, right = songs_loop, left_on
     ="SongId", right_on="SongId")
  loc_temp = loc_merge["Location"].tolist()
  # those location correspond the the right songs. Extracting them
  extract_vec = pan_emb.iloc[loc_temp].to_numpy()

  cosines = []
  for j in extract_vec:
    cosines.append(cosine_similarity(j.reshape(1, -1), mu.reshape(1,
       -1)))
  proximity.append(np.sum(cosines)/len(cosines))

 #### LOWESS

 # GS
```

```
    df = pd.DataFrame({'x': proximity, 'Raw': GS2})
    # using lowess package
    df["pred"] = lo.lowess(df["Raw"], df["x"], 0.2, 2)
    GS_hat = df["pred"]

# TS_SS
    df = pd.DataFrame({'x': proximity, 'Raw': TS_SS_norm2*1000})
    df["pred"] = lo.lowess(df["Raw"], df["x"], 0.2, 2)
    TS_hat = df["pred"]
```