Analyzing Genre Influence on Spotify Track Characteristics

Necessary R libraries:

```r
library(base)
library(reticulate) #for Python interface
library(psych) #for skew/kurt functions
library(moments)
library(stats)
library(lsr)
```

Necessary Python libraries:

```python
import kaggle
import requests
import kaggle.cli
import sys
import numpy
import pandas as pd
from pathlib import Path
from zipfile import ZipFile
from kaggle.cli import main
```

```python
#download data set from Kaggle
dataset="thedevastator/spotify-tracks-genre-dataset"
sys.argv=[sys.argv[0]] + f"datasets download {dataset}".split(" ")

kaggle_zip=ZipFile(f"{dataset.split('/')[1]}.zip")
songdat={i.filename:pd.read_csv(kaggle_zip.open(i)) for i in kaggle_zip.infolist() }["tr
ain.csv"]

#Close the ZipFile object
kaggle_zip.close()
```

Clean data (python): group subgenres into broader catagories for new genre row

```
import numpy

#Pop
songdat.loc[songdat['track_subgenre'].str.contains('pop', case=False, na=False)| songdat
['track_subgenre'].str.contains('idol', case=False, na=False), 'track_genre']='Pop'

#Metal
songdat.loc[songdat['track_subgenre'].str.contains('metal', case=False, na=False) | (son
gdat['track_subgenre']=='hardcore'), 'track_genre']='Metal'

#Rock
rock_conditions=songdat['track_subgenre'].str.contains('rock', case=False, na=False) | s
ongdat['track_subgenre'].isin(['goth', 'emo', 'garage', 'grunge', 'punk'])
songdat.loc[rock_conditions, 'track_genre']='Rock'

#Reggae
songdat.loc[songdat['track_subgenre'].str.contains('reggae', case=False, na=False) | (so
ngdat['track_subgenre']=='ska'), 'track_genre']='Reggae'

#Electronic
electronic_conditions=songdat['track_subgenre'].str.contains('dance', case=False, na=Fal
se) | songdat['track_subgenre'].str.startswith('electro') | songdat['track_subgenre'].st
r.startswith('dub') | songdat['track_subgenre'].str.contains('house', case=False, na=Fal
se) | songdat['track_subgenre'].str.contains('techno', case=False, na=False) | songdat
['track_subgenre'].isin(['edm', 'party', 'dubstep', 'hardstyle', 'breakbeat', 'club', 'd
rum-and-bass', 'idm', 'trance', 'trip-hop', 'industrial', 'happy'])
songdat.loc[electronic_conditions, 'track_genre']='Electronic'

#Disco
songdat.loc[songdat['track_subgenre'].str.contains('disco', case=False, na=False), 'trac
k_genre']='Disco'

#Jazz
songdat.loc[songdat['track_subgenre'].str.contains('jazz', case=False, na=False), 'track
_genre']='Jazz'

#Blues
songdat.loc[songdat['track_subgenre']=='blues', 'track_genre']='Blues'

#Folk
songdat.loc[songdat['track_subgenre'].isin(['folk', 'bluegrass']), 'track_genre']='Folk'

#Country
songdat.loc[songdat['track_subgenre']=='country', 'track_genre']='Country'

#R&B/Soul
songdat.loc[songdat['track_subgenre'].isin(['r-n-b', 'funk', 'soul', 'groove', 'afrobea
t']), 'track_genre']='R&B/Soul'

#Latin
latin_conditions=songdat['track_subgenre'].str.contains('latin', case=False, na=False) |
songdat['track_subgenre'].isin(['brazil', 'forro', 'mpb', 'pagode', 'salsa', 'samba', 's
```

```
ertanejo', 'spanish'])
songdat.loc[latin_conditions, 'track_genre']='Latin'

#World
world_conditions=songdat['track_subgenre'].str.contains('world', case=False, na=False) |
songdat['track_subgenre'].isin(['french', 'german', 'indian', 'malay', 'swedish', 'turki
sh', 'british', 'anime'])
songdat.loc[world_conditions, 'track_genre']='World'

#Hip-hop
songdat.loc[songdat['track_subgenre'].isin(['hip-hop', 'sad']), 'track_genre']='Hip-hop'

#Kids
songdat.loc[songdat['track_subgenre'].isin(['children', 'kids', 'disney', 'comedy', 'sho
w-tunes']), 'track_genre']='Kids/Family'

#Alternative
songdat.loc[songdat['track_subgenre'].isin(['alternative', 'chill', 'indie', 'new-age',
'sleep', 'ambient']), 'track_genre']='Alternative'

#Instrumental
songdat.loc[songdat['track_subgenre'].isin(['acoustic', 'guitar', 'piano']), 'track_genr
e']='Instrumental'

#Singer-songwriter
songdat.loc[songdat['track_subgenre'].str.contains('songwriter', case=False, na=False),
'track_genre']='Singer-songwriter'

#Classical
songdat.loc[songdat['track_subgenre'].isin(['classical', 'gospel', 'opera']), 'track_gen
re']='Classical'


#Capitalize first letter in Track Subgenre
songdat['track_subgenre']=songdat['track_subgenre'].apply(lambda x: x.capitalize())
```

Check if we’ve left behind any track_subgenre rows that haven’t been matched with track_genre

```
if(songdat['track_genre'].isnull().sum())!=0:
    null_rows=songdat[songdat['track_genre'].isnull()]
    grouped_null=null_rows.groupby('track_subgenre').size().reset_index(name='count')
    print(grouped_null)
```

```
##   track_subgenre  count
## 0      Grindcore   1000
## 1     Honky-tonk   1000
## 2        Iranian   1000
## 3        Romance   1000
## 4          Study   1000
## 5          Tango   1000
```

```
#Missed Grindcore, Honky-tonk, Iranian, Romance, Study, Tango

#Add Grindcore to Metal
songdat.loc[songdat['track_subgenre']=='Grindcore', 'track_genre']='Metal'

#Add Honky-tonk to Country
songdat.loc[songdat['track_subgenre']=='Honky-tonk', 'track_genre']='Country'

#Add Iranian to World
songdat.loc[songdat['track_subgenre']=='Iranian', 'track_genre']='World'

#Add Tango to Latin
songdat.loc[songdat['track_subgenre']=='Tango', 'track_genre']='Latin'

#Add Study to Alternative
songdat.loc[songdat['track_subgenre']=='Study', 'track_genre']='Alternative'

#Not sure where Romance should go, let's look at the artist names
romance=songdat[songdat['track_subgenre']=='Romance']
print(romance[['artists','track_name']].sort_values(by='artists'))
```

```
##                              artists                     track_name
## 93700          Alexander Vertinsky                    Lilovyy negr
## 93279          Alexander Vertinsky  Ja segodnja smejus' nad soboy
## 93312          Alexander Vertinsky  Ja segodnja smejus' nad soboy
## 93416          Alexander Vertinsky                Pesenka o zhene
## 93457          Alexander Vertinsky                Pesenka o zhene
## ...                            ...                            ...
## 93802              Тамара Церетели        Взгляд твоих чёрных очей
## 93405  Татьяна Комова;Георгий Квик                        Солнышко
## 93859  Татьяна Комова;Георгий Квик                    Пятеро детей
## 93454  Татьяна Комова;Георгий Квик                        Солнышко
## 93553  Татьяна Комова;Георгий Квик                         Сумерки
##
## [1000 rows x 2 columns]
```

```
#Looks like Romance is a form of classical Russian music, add to World
songdat.loc[songdat['track_subgenre']=='Romance', 'track_genre']='World'
```

Choose numeric variables we want to compare against track_genre, our predictor variable. Let’s try “popularity”, “energy”, “tempo”, “duration_ms”, “danceability”,“loudness”. We’ll scale all our raw values first to make comparison easier.

```
# Convert py$songdat from Python to R dataframe


numeric_columns <- c("popularity", "energy", "tempo", "duration_ms", "danceability","lou
dness")

#Quick function to scale our rows quickly
scale_column <- function(df, column_name) {
  if (!column_name %in% colnames(df)) {
    stop("Column not found in the songdat")
  }

  df_scaled <- df
  df_scaled[[paste0(column_name, "_scaled")]] <- scale(df[[column_name]], scale=FALSE)

  return(df_scaled)
}

#cycle through all our rows in numeric_colummns, scaling each, and append to songdat
for (column_name in as.character(numeric_columns)) {
  scaled_column<-scale(py$songdat[column_name],scale=FALSE)
    py$songdat[[paste0(column_name, "_scaled")]] <-scaled_column[, 1]
}
```

Now check the Skew/Kurtosis values for all our dv's of interest.

```
#Look at skew/kurt for scaled variables
skewness_scaled <- skew(py$songdat[paste0(numeric_columns,"_scaled")])
kurtosis_scaled <- kurtosis(py$songdat[paste0(numeric_columns,"_scaled")])


#Combine results into a data frame for easier viewing
skew_kurtosis_df <- data.frame(skewness=skewness_scaled, kurtosis=kurtosis_scaled)
print(skew_kurtosis_df)
```

```
##                        skewness   kurtosis
## popularity_scaled    0.04640129   2.072233
## energy_scaled       -0.59698571   2.474260
## tempo_scaled         0.23228875   2.891372
## duration_ms_scaled  11.19488687 357.936795
## danceability_scaled -0.39948612   2.815453
## loudness_scaled     -2.00648913   8.895967
```

```
#Duration_ms has way too much skew/kurt, 11.19503417 and 357.936795! Not worth attemptin
g a transformation, too much variance
numeric_columns<-subset(numeric_columns, numeric_columns!="duration_ms") #Drop Duration_
ms
```

All variables look good except for 2, duration_ms and loudness. We removed duration_ms because the skew/kurt was to extreme to fix through transformations. Loudness has a kurtosis of 8.9, which is problematic but might benefit from transformations. Let's try square root, inverse, and log transformations.

```
song_pos=abs(py$songdat$loudness+1e-10)#There are 0s, to avoid NaN results add a really small number so it's technically positive

#Square Root
songNormSqt <- data.frame((song_pos+1)^0.5)
skewnessSqt <- skew(song_pos)
kurtosisSqt <- kurtosis(song_pos)

skewnessSqt
```

```
## [1] 2.009743
```

```
kurtosisSqt
```

```
## [1] 8.902822
```

```
#Inverse
songNormIn <- 1/(song_pos+1)
skewnessIn <- skew(songNormIn)
kurtosisIn <- kurtosis(songNormIn)

skewnessIn
```

```
## [1] 2.606847
```

```
kurtosisIn
```

```
## [1] 19.33271
```

```
#Log
songNormLg <- log10(song_pos+1)

skewnessLg <- skew(songNormLg)
kurtosisLg <- kurtosis(songNormLg)

kurtosisLg #Log looks good, kurtosis improved from 8.9 to 3.6 and skew is still minimal at .20.
```

```
## [1] 3.586076
```

```
skewnessLg
```

```
## [1] 0.1690073
```

The Log transformation for loudness looks great, skew dropped to 3.6 from 8.9. Let's apply this normally distributed version and use it as our loudness variable.

```
#Let's replace our raw loudness data with our more normalized log transformation
#use unlist to ensure songNormLg is a 1-dim array
loudness_log_scaled_standardized <- scale(unlist(songNormLg)) #rescale

#assign scaled loudness values to songdat
py$songdat$loudness_log_scaled <- loudness_log_scaled_standardized[,1]

#update the numeric column list since we aren't using the raw
#Find the index of "loudness"
index_to_replace <- which(numeric_columns=="loudness")

#replace "loudness" with "loudness_log_scaled"
numeric_columns[index_to_replace] <- "loudness_log_scaled"
```

Let's perform a series of one-way ANOVAs, using our numeric_column ("popularity", "energy", "tempo","danceability", "loudness_log_squared") as our dv's, and track_genre as our iv

```
for(column_name in as.character(numeric_columns)){
print(column_name)
  aovTemp<-aov(py$songdat[[column_name]]~py$songdat$track_genre)
print(summary(aovTemp)) #Print ANOVA results
print(etaSquared(aovTemp,type=3,anova=TRUE)) #...and eta-squard (effect size)
}
```

```
## [1] "popularity"
##                          Df   Sum Sq Mean Sq F value Pr(>F)    
## py$songdat$track_genre     18  3043694  169094   359.1 <2e-16 ***
## Residuals              113981 53672693     471                   
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##                            eta.sq eta.sq.part       SS     df          MS
## py$songdat$track_genre 0.05366515  0.05366515  3043694     18 169094.0867
## Residuals              0.94633485          NA 53672693 113981    470.8916
##                               F  p
## py$songdat$track_genre 359.0935  0
## Residuals                    NA NA
## [1] "energy"
##                          Df Sum Sq Mean Sq F value Pr(>F)    
## py$songdat$track_genre     18   2146  119.22    2682 <2e-16 ***
## Residuals              113981   5066    0.04                   
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##                           eta.sq eta.sq.part       SS     df          MS
## py$songdat$track_genre 0.2975362   0.2975362 2145.938     18 119.2187990
## Residuals              0.7024638          NA 5066.422 113981   0.0444497
##                               F  p
## py$songdat$track_genre 2682.106  0
## Residuals                    NA NA
## [1] "tempo"
##                          Df   Sum Sq Mean Sq F value Pr(>F)    
## py$songdat$track_genre     18  3158697  175483   201.4 <2e-16 ***
## Residuals              113981 99291326     871                   
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##                            eta.sq eta.sq.part       SS     df          MS
## py$songdat$track_genre 0.03083159  0.03083159  3158697     18 175483.1472
## Residuals              0.96916841          NA 99291326 113981    871.1217
##                              F  p
## py$songdat$track_genre 201.445  0
## Residuals                   NA NA
## [1] "danceability"
##                          Df Sum Sq Mean Sq F value Pr(>F)    
## py$songdat$track_genre     18    680   37.78    1564 <2e-16 ***
## Residuals              113981   2753    0.02                   
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##                           eta.sq eta.sq.part        SS     df          MS
## py$songdat$track_genre 0.1980524   0.1980524  679.9724     18 37.77624624
## Residuals              0.8019476          NA 2753.3225 113981  0.02415598
##                               F  p
## py$songdat$track_genre 1563.847  0
## Residuals                    NA NA
## [1] "loudness_log_scaled"
##                          Df Sum Sq Mean Sq F value Pr(>F)    
## py$songdat$track_genre     18  22528  1251.5    1560 <2e-16 ***
## Residuals              113981  91471     0.8                   
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##                            eta.sq eta.sq.part       SS     df           MS
## py$songdat$track_genre 0.1976133   0.1976133 22527.72     18 1251.5398287
## Residuals              0.8023867          NA 91471.28 113981    0.8025134
##                               F  p
## py$songdat$track_genre 1559.525  0
## Residuals                    NA NA
```

Surprisingly all of our dv's produced significant results, meaning track popularity, energy, tempo, danceability, and loudness are all influenced by a song's genre, p<.001. The largest effect sizes were seen for danceability (η2=.24) and loudness (η2=.22).

Since we know the means differ between track genre's, lets do some summary stats for each of our iv's by track genre. Pairwise analysis would be too much here since we have 18 distinct genre values, but we know we can trust our mean/median/modes due to significant ANOVA results.

```
options(max.print =1000000)

#print the mean, median, sd for all our values of interest
summary_stats_by_genre <- aggregate(py$songdat[numeric_columns], by=list(track_genre=py$songdat$track_genre), FUN=function(x) c(mean=mean(x), median=median(x), sd=sd(x)))

print(summary_stats_by_genre)
```

```
##          track_genre popularity.mean popularity.median popularity.sd
## 1        Alternative        35.90257          39.00000      23.80630
## 2              Blues        31.18800          34.00000      27.48425
## 3          Classical        26.43833          25.00000      17.95328
## 4            Country        16.69150          12.00000      20.29929
## 5              Disco        33.52200          32.00000      24.75695
## 6         Electronic        30.84392          27.00000      21.80143
## 7               Folk        31.84100          27.00000      18.94955
## 8            Hip-hop        45.06900          55.00000      25.47522
## 9       Instrumental        39.09400          40.50000      17.88250
## 10              Jazz        13.62800           0.00000      23.18290
## 11       Kids/Family        26.66900          23.00000      14.65071
## 12             Latin        34.37182          40.00000      18.85740
## 13             Metal        31.25086          25.00000      19.44254
## 14               Pop        41.35000          45.00000      22.64844
## 15          R&B/Soul        30.20560          32.00000      23.36409
## 16            Reggae        26.53800          33.00000      26.25452
## 17              Rock        37.63000          37.00000      22.84109
## 18 Singer-songwriter        37.81300          43.00000      27.68428
## 19             World        33.22636          38.00000      21.92392
##    energy.mean energy.median energy.sd tempo.mean tempo.median  tempo.sd
## 1    0.4153339     0.3960000 0.2822690  110.32292    107.99200  35.60832
## 2    0.5818775     0.5835000 0.2205792  116.56835    114.54000  30.70551
## 3    0.3610459     0.3270000 0.2490275  113.05316    113.26450  30.72567
## 4    0.4818810     0.4530000 0.2230807  120.26009    117.99700  30.84552
## 5    0.7375650     0.7770000 0.1877227  121.97438    123.98100  19.22186
## 6    0.7555168     0.7880000 0.1769618  127.68878    125.11850  25.61376
## 7    0.5380433     0.5445000 0.2130263  122.52417    120.96550  27.67009
## 8    0.5725001     0.5825000 0.1981201  117.91727    112.46150  29.98537
## 9    0.3601566     0.3030000 0.2630827  116.36518    114.78550  31.72331
## 10   0.3529544     0.3320000 0.1858928  112.63647    109.15200  31.64017
## 11   0.5017937     0.5010000 0.2549701  114.50411    112.95550  30.66556
## 12   0.6681083     0.7010000 0.1892528  122.81746    119.03650  30.74957
## 13   0.8859689     0.9370000 0.1318622  126.01259    123.00800  29.89331
## 14   0.6471027     0.6610000 0.2167343  124.35875    123.88100  29.40967
## 15   0.6542870     0.6750000 0.1985774  119.87530    119.99300  26.63007
## 16   0.7525510     0.7680000 0.1347853  122.35945    109.98400  32.93197
## 17   0.7165144     0.7620000 0.2094055  126.45852    125.00300  30.45898
## 18   0.4341884     0.4360000 0.2064300  119.73732    117.98200  30.94120
## 19   0.5507639     0.5640000 0.2573138  118.22343    118.32300  29.92687
##    danceability.mean danceability.median danceability.sd
## 1          0.4830041           0.5150000       0.2253829
## 2          0.5685670           0.5790000       0.1472112
## 3          0.3895946           0.3890000       0.1494489
## 4          0.5632585           0.5640000       0.1203286
## 5          0.6766920           0.6930000       0.1232074
## 6          0.6349987           0.6470000       0.1416097
## 7          0.5466030           0.5520000       0.1195352
## 8          0.7142660           0.7270000       0.1234201
## 9          0.5149096           0.5260000       0.1494902
## 10         0.5099750           0.4990000       0.1413249
## 11         0.5998361           0.6070000       0.1948149
```

```
## 12        0.6200453         0.6300000      0.1386085
## 13        0.3999535         0.4000000      0.1538802
## 14        0.5768700         0.5810000      0.1365507
## 15        0.6292348         0.6470000      0.1559566
## 16        0.6948427         0.7210000      0.1370737
## 17        0.5161622         0.5200000      0.1410385
## 18        0.5620220         0.5660000      0.1298911
## 19        0.5263003         0.5400000      0.1893377
##    loudness_log_scaled.mean loudness_log_scaled.median loudness_log_scaled.sd
## 1              0.9242381997               0.8860563633           1.1442901784
## 2              0.1644425561               0.1563732667           0.7827120462
## 3              0.8985326803               0.8000491275           1.1389713908
## 4              0.3778392010               0.4944670859           0.7724719383
## 5             -0.1260990675              -0.0845407694           0.8825831523
## 6             -0.3260947153              -0.3264271720           0.9287134098
## 7              0.3923454423               0.4004746721           0.6957829603
## 8              0.0451322918              -0.0002549701           0.8444869827
## 9              0.8976135603               0.8737704039           1.1029524403
## 10             0.7700157736               0.7353991642           0.6959407367
## 11             0.5356254782               0.5076108818           0.9239830595
## 12            -0.1986341967              -0.1961881537           0.7740651700
## 13            -0.5797516891              -0.6225886092           0.7550342060
## 14            -0.1764631239              -0.1049569029           0.8789564985
## 15            -0.0994416018              -0.0750636006           0.7329113067
## 16            -0.5869576556              -0.6137241482           0.6817710769
## 17            -0.2283807470              -0.2486230457           0.8347214506
## 18             0.4444801848               0.4755011094           0.7132964409
## 19             0.3202800714               0.2667884074           1.0009273097
```

Looks good, let’s export our summary data into a CSV to import into Tableau!

```
write.csv(summary_stats_by_genre, file="summary_stats_by_genre.csv", row.names=FALSE)
```