Mel_Spotify_Annotated

November 5, 2024

1 Project 4: Music Popularity Prediction

#PROBLEM DEFINITION:

1.0.1 This is a supervised learning problem that will use labeled target data in the form of features collected for popular songs, defining "popular" as those songs that have been on the Top 200 Weekly (Global) charts of Spotify in 2020 & 2021.

1.0.2 The goal is to predict the popularity of a song using a tree-based regression model trained on the most important features.

The outcomes for the project will be to:

- Minimize the cross-validated ***root mean squared error (RMSE)*** when predicting the popularity of a new song.
- Determine the importance of the features in driving the regression result using the parameters of the trees after carefully selecting to avoid over-fitting.

There are three main challenges for this project:

1. Determining the outcome (i.e. target). There is a "popularity" column. But other columns may or may not be more appropriate indicators of popularity.
2. Choosing appropriate predictors (i.e. features). When building a machine learning model, we want to make sure that we consider how the model will be ultimately used. For this project, we are predicting the popularity of a new song. Therefore, we should only include the predictors we would have for a new song.
3. Data cleaning and feature engineering. Some creative cleaning and/or feature engineering may be needed to extract useful information for prediction.

Once again, be sure to go through the whole data science process and document as such in your Jupyter notebook.

The data is available AWS at https://ddc-datascience.s3.amazonaws.com/Projects/Project.4-Spotify/Data/Spotify.csv .

[ ]:
```
# import Tools

import pandas as pd
from functools import reduce
```

```python
import numpy as np

from scipy import stats
import statsmodels.api as sm

import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab

from sklearn.model_selection import train_test_split #training it
from sklearn import datasets
from sklearn.tree import DecisionTreeRegressor #the model
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn import metrics

import re

import pickle
```

#Data Source

[ ]:
```python
# Define a variable "url" with the location of a file on the internet. Then, use the curl command to get information about the file at that location, such as its type, size, and other metadata provided in the header of the webpage. Since only the -I flag is used, it does not download the actual file, but only shows its header information

url = "https://ddc-datascience.s3.amazonaws.com/Projects/Project.4-Spotify/Data/Spotify.csv"
!curl -I {url}

# !: In Colab/Jupyter notebooks, the exclamation mark (!) allows you to execute shell commands (commands you'd typically run in a terminal)

# curl: This is the command used for transferring data to or from a server.

# -I: This option tells curl to only fetch the HTTP headers of the response (not the actual file content). Headers provide metadata about the file, like its size, type, and last modified date.

 # {url}: This part substitutes the value of the defined "url" variable into the command, so curl knows which URL to access.
```

HTTP/1.1 200 OK
x-amz-id-2:
zHlAvoWaCEnFc3cpHmONHSIudE9YtSci42A7bzJVMCaO1kJcqxPVZkC+ZUy71LvBoYjqBCRSVhw=

```
x-amz-request-id: ZP7K7A5F0QCXJ4E9
Date: Tue, 05 Nov 2024 15:33:20 GMT
Last-Modified: Wed, 04 Oct 2023 17:23:56 GMT
ETag: "65b9875b11e0d7ea03ee2af024f45e99"
x-amz-server-side-encryption: AES256
Accept-Ranges: bytes
Content-Type: text/csv
Server: AmazonS3
Content-Length: 738124
```

HTTP/1.1 200 OK: This is the status line. HTTP/1.1 indicates the version of the HTTP protocol used. 200 OK is the status code, and it means the request was successful. Your request to access the file at the specified URL was successful, and the server is sending back the data.

x-amz-id-2, x-amz-request-id: These are Amazon S3-specific headers providing unique identifiers for the request and the data transfer.

Date: This shows when the server generated the response.

Last-Modified: This shows when the file on the server was last modified.

ETag: This is a unique identifier for the current version of the file.

x-amz-server-side-encryption: This tells you the file is encrypted on Amazon S3's servers for security.

Accept-Ranges: This means the server supports partial downloads (e.g., downloading specific ranges of bytes).

Content-Type: This tells you the type of data being sent. In this case, it's "text/csv", indicating a comma-separated values file.

Server: This tells you the software running on the server (Amazon S3).

Content-Length: This shows the size of the file in bytes (738124 bytes).

[ ]:
```
# Above: size = 738124 bytes

# Read the csv file and print the head

spot_df = pd.read_csv(url)
spot_df.head().transpose()
```

[ ]:
```
                                                                0  \
Index                                                           1
Highest Charting Position                                       1
Number of Times Charted                                         8
Week of Highest Charting                   2021-07-23--2021-07-30
Song Name                                                 Beggin'
Streams                                                48,633,449
Artist                                                   Måneskin
```

Artist Followers	3377762
Song ID	3Wrjm47oTz2sjIgck11l5e
Genre	['indie rock italiano', 'italian pop']
Release Date	2017-12-08
Weeks Charted	2021-07-23--2021-07-30\n2021-07-16--2021-07-23…
Popularity	100
Danceability	0.714
Energy	0.8
Loudness	-4.808
Speechiness	0.0504
Acousticness	0.127
Liveness	0.359
Tempo	134.002
Duration (ms)	211560
Valence	0.589
Chord	B

	1 \
Index	2
Highest Charting Position	2
Number of Times Charted	3
Week of Highest Charting	2021-07-23--2021-07-30
Song Name	STAY (with Justin Bieber)
Streams	47,248,719
Artist	The Kid LAROI
Artist Followers	2230022
Song ID	5HCyWlXZPP0y6Gqq8TgA20
Genre	['australian hip hop']
Release Date	2021-07-09
Weeks Charted	2021-07-23--2021-07-30\n2021-07-16--2021-07-23…
Popularity	99
Danceability	0.591
Energy	0.764
Loudness	-5.484
Speechiness	0.0483
Acousticness	0.0383
Liveness	0.103
Tempo	169.928
Duration (ms)	141806
Valence	0.478
Chord	C#/Db

	2 \
Index	3
Highest Charting Position	1
Number of Times Charted	11
Week of Highest Charting	2021-06-25--2021-07-02

Song Name	good 4 u
Streams	40,162,559
Artist	Olivia Rodrigo
Artist Followers	6266514
Song ID	4ZtFanR9U6ndgddUvNcjcG
Genre	['pop']
Release Date	2021-05-21
Weeks Charted	2021-07-23--2021-07-30\n2021-07-16--2021-07-23…
Popularity	99
Danceability	0.563
Energy	0.664
Loudness	-5.044
Speechiness	0.154
Acousticness	0.335
Liveness	0.0849
Tempo	166.928
Duration (ms)	178147
Valence	0.688
Chord	A

	3 \
Index	4
Highest Charting Position	3
Number of Times Charted	5
Week of Highest Charting	2021-07-02--2021-07-09
Song Name	Bad Habits
Streams	37,799,456
Artist	Ed Sheeran
Artist Followers	83293380
Song ID	6PQ88X9TkUIAUIZJHW2upE
Genre	['pop', 'uk pop']
Release Date	2021-06-25
Weeks Charted	2021-07-23--2021-07-30\n2021-07-16--2021-07-23…
Popularity	98
Danceability	0.808
Energy	0.897
Loudness	-3.712
Speechiness	0.0348
Acousticness	0.0469
Liveness	0.364
Tempo	126.026
Duration (ms)	231041
Valence	0.591
Chord	B

	4
Index	5

```
Highest Charting Position                                    5
Number of Times Charted                                      1
Week of Highest Charting                  2021-07-23--2021-07-30
Song Name                            INDUSTRY BABY (feat. Jack Harlow)
Streams                                             33,948,454
Artist                                               Lil Nas X
Artist Followers                                       5473565
Song ID                                 27NovPIUIRrOZoCHxABJwK
Genre                                 ['lgbtq+ hip hop', 'pop rap']
Release Date                                        2021-07-23
Weeks Charted                             2021-07-23--2021-07-30
Popularity                                                  96
Danceability                                             0.736
Energy                                                   0.704
Loudness                                                -7.409
Speechiness                                             0.0615
Acousticness                                            0.0203
Liveness                                                0.0501
Tempo                                                  149.995
Duration (ms)                                           212000
Valence                                                  0.894
Chord                                                    D#/Eb
```

[ ]:
```
spot_df.columns
```

[ ]:
```
Index(['Index', 'Highest Charting Position', 'Number of Times Charted',
       'Week of Highest Charting', 'Song Name', 'Streams', 'Artist',
       'Artist Followers', 'Song ID', 'Genre', 'Release Date', 'Weeks Charted',
       'Popularity', 'Danceability', 'Energy', 'Loudness', 'Speechiness',
       'Acousticness', 'Liveness', 'Tempo', 'Duration (ms)', 'Valence',
       'Chord'],
      dtype='object')
```

[ ]:
```
spot_df.shape
```

[ ]:
```
(1556, 23)
```

#Data Cleaning

[ ]:
```
spot_df = spot_df.nunique()
spot_df
```

[ ]:
```
Index                        1556
Highest Charting Position     200
Number of Times Charted        75
Week of Highest Charting       83
Song Name                    1556
Streams                      1556
```

```
Artist                         716
Artist Followers               600
Song ID                       1517
Genre                          395
Release Date                   478
Weeks Charted                  775
Popularity                      70
Danceability                   530
Energy                         575
Loudness                      1394
Speechiness                    772
Acousticness                   965
Liveness                       606
Tempo                         1461
Duration (ms)                 1486
Valence                        732
Chord                           13
dtype: int64
```

[ ]:
```
#Drop 3 identifier columns, drop 2 messy date columns (i.e.,. we don't care␣
↪about dates when a song was popular), drop 3 nominal columns

spot_df = pd.read_csv(url, header=0)

spot_df.drop(['Index','Week of Highest Charting','Song ID', 'Release Date',␣
↪'Weeks Charted', 'Song Name', 'Artist', 'Genre', 'Chord'], axis=1,␣
↪inplace=True)

spot_df.head().transpose()
```

[ ]:
```
                                    0           1           2           3  \
Highest Charting Position           1           2           1           3
Number of Times Charted             8           3          11           5
Streams                    48,633,449  47,248,719  40,162,559  37,799,456
Artist Followers              3377762     2230022     6266514    83293380
Popularity                        100          99          99          98
Danceability                    0.714       0.591       0.563       0.808
Energy                            0.8       0.764       0.664       0.897
Loudness                       -4.808      -5.484      -5.044      -3.712
Speechiness                    0.0504      0.0483       0.154      0.0348
Acousticness                    0.127      0.0383       0.335      0.0469
Liveness                        0.359       0.103      0.0849       0.364
Tempo                         134.002     169.928     166.928     126.026
Duration (ms)                  211560      141806      178147      231041
Valence                         0.589       0.478       0.688       0.591

                                    4
```

```
Highest Charting Position          5
Number of Times Charted            1
Streams                   33,948,454
Artist Followers             5473565
Popularity                        96
Danceability                   0.736
Energy                         0.704
Loudness                      -7.409
Speechiness                   0.0615
Acousticness                  0.0203
Liveness                      0.0501
Tempo                        149.995
Duration (ms)                 212000
Valence                        0.894
```

[ ]:
```
spot_df_noID = spot_df.copy()
#rename df to indicate identifiers et al have been removed
spot_df_noID.head().transpose()
#print the head to inspect
```

[ ]:
```
                                   0           1           2           3  \
Highest Charting Position          1           2           1           3
Number of Times Charted            8           3          11           5
Streams                   48,633,449  47,248,719  40,162,559  37,799,456
Artist Followers             3377762     2230022     6266514    83293380
Popularity                       100          99          99          98
Danceability                   0.714       0.591       0.563       0.808
Energy                           0.8       0.764       0.664       0.897
Loudness                      -4.808      -5.484      -5.044      -3.712
Speechiness                   0.0504      0.0483       0.154      0.0348
Acousticness                   0.127      0.0383       0.335      0.0469
Liveness                       0.359       0.103      0.0849       0.364
Tempo                        134.002     169.928     166.928     126.026
Duration (ms)                 211560      141806      178147      231041
Valence                        0.589       0.478       0.688       0.591

                                   4
Highest Charting Position          5
Number of Times Charted            1
Streams                   33,948,454
Artist Followers             5473565
Popularity                        96
Danceability                   0.736
Energy                         0.704
Loudness                      -7.409
Speechiness                   0.0615
Acousticness                  0.0203
```

```
Liveness                          0.0501
Tempo                            149.995
Duration (ms)                     212000
Valence                            0.894
```

[ ]:
```
# 1556 rows, 14 cols instead of 23 without Identifiers, Dates, or Nominal data
spot_df_noID.shape
```

[ ]:
```
(1556, 14)
```

[ ]:
```
spot_df_noID.dtypes
# Comparing the head to the dtype output reveals that many "object" type are␣
↪actually numeric type
```

[ ]:
```
Highest Charting Position     int64
Number of Times Charted       int64
Streams                      object
Artist Followers             object
Popularity                   object
Danceability                 object
Energy                       object
Loudness                     object
Speechiness                  object
Acousticness                 object
Liveness                     object
Tempo                        object
Duration (ms)                object
Valence                      object
dtype: object
```

[ ]:
```
# Remove commas from 'Streams' column
if 'Streams' in spot_df_noID.columns:
    spot_df_noID['Streams'] = spot_df_noID['Streams'].astype(str).str.
↪replace(',', '')
```

[ ]:
```
# Prepare to convert object types that are actually numbers into numeric type

# List the precise column names before attempting conversion and confirm they␣
↪exist before conversion

# Convert all columns to numeric
for col in ['Streams', 'Artist Followers', 'Popularity',␣
↪'Liveness','Danceability', 'Energy', 'Loudness', 'Speechiness',␣
↪'Acousticness', 'Tempo', 'Duration (ms)', 'Valence']:
    if col in spot_df_noID.columns:  # Confirm the column exists
        spot_df_noID[col] = pd.to_numeric(spot_df_noID[col], errors='coerce') ␣
↪# Convert to numeric, setting non-convertible to NaN
```

```
    else:
        print(f"Warning: Column '{col}' not found in the DataFrame.")

spot_df_noID.head().transpose()


for col in ['Artist Followers', 'Popularity', 'Danceability',␣
 ↪'Liveness','Energy', 'Loudness', 'Speechiness', 'Acousticness', 'Tempo',␣
 ↪'Duration (ms)', 'Valence']:
    if col in spot_df_noID.columns:  # Check if the column exists
        spot_df_noID[col] = pd.to_numeric(spot_df_noID[col], errors='coerce')
    else:
        print(f"Warning: Column '{col}' not found in the DataFrame.")

spot_df_noID.head().transpose()
```

[ ]:

```
                                       0              1              2  \
Highest Charting Position   1.000000e+00   2.000000e+00   1.000000e+00
Number of Times Charted     8.000000e+00   3.000000e+00   1.100000e+01
Streams                     4.863345e+07   4.724872e+07   4.016256e+07
Artist Followers            3.377762e+06   2.230022e+06   6.266514e+06
Popularity                  1.000000e+02   9.900000e+01   9.900000e+01
Danceability                7.140000e-01   5.910000e-01   5.630000e-01
Energy                      8.000000e-01   7.640000e-01   6.640000e-01
Loudness                   -4.808000e+00  -5.484000e+00  -5.044000e+00
Speechiness                 5.040000e-02   4.830000e-02   1.540000e-01
Acousticness                1.270000e-01   3.830000e-02   3.350000e-01
Liveness                    3.590000e-01   1.030000e-01   8.490000e-02
Tempo                       1.340020e+02   1.699280e+02   1.669280e+02
Duration (ms)               2.115600e+05   1.418060e+05   1.781470e+05
Valence                     5.890000e-01   4.780000e-01   6.880000e-01

                                       3              4
Highest Charting Position   3.000000e+00   5.000000e+00
Number of Times Charted     5.000000e+00   1.000000e+00
Streams                     3.779946e+07   3.394845e+07
Artist Followers            8.329338e+07   5.473565e+06
Popularity                  9.800000e+01   9.600000e+01
Danceability                8.080000e-01   7.360000e-01
Energy                      8.970000e-01   7.040000e-01
Loudness                   -3.712000e+00  -7.409000e+00
Speechiness                 3.480000e-02   6.150000e-02
Acousticness                4.690000e-02   2.030000e-02
Liveness                    3.640000e-01   5.010000e-02
Tempo                       1.260260e+02   1.499950e+02
Duration (ms)               2.310410e+05   2.120000e+05
Valence                     5.910000e-01   8.940000e-01
```

[ ]:
```
spot_df_noID.shape
# Confirm rows and columns are still present - yes
```

[ ]: (1556, 14)

[ ]:
```
spot_df_noID.describe().transpose()
# Basic stats on the variables; notice row numbers for some are different by 11
```

[ ]:
```
                           count          mean           std           min  \
Highest Charting Position  1556.0  8.774422e+01  5.814723e+01  1.000000e+00
Number of Times Charted    1556.0  1.066838e+01  1.636055e+01  1.000000e+00
Streams                    1556.0  6.340219e+06  3.369479e+06  4.176083e+06
Artist Followers           1545.0  1.471690e+07  1.667579e+07  4.883000e+03
Popularity                 1545.0  7.008932e+01  1.582403e+01  0.000000e+00
Danceability               1545.0  6.899968e-01  1.424440e-01  1.500000e-01
Energy                     1545.0  6.334951e-01  1.615770e-01  5.400000e-02
Loudness                   1545.0 -6.348474e+00  2.509281e+00 -2.516600e+01
Speechiness                1545.0  1.236557e-01  1.103827e-01  2.320000e-02
Acousticness               1545.0  2.486945e-01  2.503259e-01  2.550000e-05
Liveness                   1545.0  1.812024e-01  1.440710e-01  1.970000e-02
Tempo                      1545.0  1.228110e+02  2.959109e+01  4.671800e+01
Duration (ms)              1545.0  1.979408e+05  4.714893e+04  3.013300e+04
Valence                    1545.0  5.147038e-01  2.273256e-01  3.200000e-02

                                    25%           50%           75%  \
Highest Charting Position  3.700000e+01  8.000000e+01  1.370000e+02
Number of Times Charted    1.000000e+00  4.000000e+00  1.200000e+01
Streams                    4.915322e+06  5.275748e+06  6.455044e+06
Artist Followers           2.123734e+06  6.852509e+06  2.269875e+07
Popularity                 6.500000e+01  7.300000e+01  8.000000e+01
Danceability               5.990000e-01  7.070000e-01  7.960000e-01
Energy                     5.320000e-01  6.420000e-01  7.520000e-01
Loudness                  -7.491000e+00 -5.990000e+00 -4.711000e+00
Speechiness                4.560000e-02  7.650000e-02  1.650000e-01
Acousticness               4.850000e-02  1.610000e-01  3.880000e-01
Liveness                   9.660000e-02  1.240000e-01  2.170000e-01
Tempo                      9.796000e+01  1.220120e+02  1.438600e+02
Duration (ms)              1.692660e+05  1.935910e+05  2.189020e+05
Valence                    3.430000e-01  5.120000e-01  6.910000e-01

                                    max
Highest Charting Position  2.000000e+02
Number of Times Charted    1.420000e+02
Streams                    4.863345e+07
Artist Followers           8.333778e+07
Popularity                 1.000000e+02
Danceability               9.800000e-01
```

```
Energy                      9.700000e-01
Loudness                    1.509000e+00
Speechiness                 8.840000e-01
Acousticness                9.940000e-01
Liveness                    9.620000e-01
Tempo                       2.052720e+02
Duration (ms)               5.881390e+05
Valence                     9.790000e-01
```

[ ]:
```
spot_df_noID.dtypes
# Confirm datatypes for columns with numbers are now numeric
```

[ ]:
```
Highest Charting Position      int64
Number of Times Charted        int64
Streams                        int64
Artist Followers             float64
Popularity                   float64
Danceability                 float64
Energy                       float64
Loudness                     float64
Speechiness                  float64
Acousticness                 float64
Liveness                     float64
Tempo                        float64
Duration (ms)                float64
Valence                      float64
dtype: object
```

[ ]:
```
# Are there nulls?  yes
null_counts = spot_df_noID.isna().sum()
print(null_counts)
```

```
Highest Charting Position     0
Number of Times Charted       0
Streams                       0
Artist Followers             11
Popularity                   11
Danceability                 11
Energy                       11
Loudness                     11
Speechiness                  11
Acousticness                 11
Liveness                     11
Tempo                        11
Duration (ms)                11
Valence                      11
dtype: int64
```

[ ]:
```
# Remove the nulls
spot_df_noID.dropna(inplace=True)
```

[ ]:
```
# Confirm the nulls are removed
null_counts = spot_df_noID.isna().sum()
print(null_counts)
```

```
Highest Charting Position    0
Number of Times Charted      0
Streams                      0
Artist Followers             0
Popularity                   0
Danceability                 0
Energy                       0
Loudness                     0
Speechiness                  0
Acousticness                 0
Liveness                     0
Tempo                        0
Duration (ms)                0
Valence                      0
dtype: int64
```

[ ]:
```
# Copy the df for Data Analysis and check the head
df_num = spot_df_noID.copy()
df_num.head().transpose()
```

[ ]:
```
                                      0             1             2  \
Highest Charting Position  1.000000e+00  2.000000e+00  1.000000e+00
Number of Times Charted    8.000000e+00  3.000000e+00  1.100000e+01
Streams                    4.863345e+07  4.724872e+07  4.016256e+07
Artist Followers           3.377762e+06  2.230022e+06  6.266514e+06
Popularity                 1.000000e+02  9.900000e+01  9.900000e+01
Danceability               7.140000e-01  5.910000e-01  5.630000e-01
Energy                     8.000000e-01  7.640000e-01  6.640000e-01
Loudness                  -4.808000e+00 -5.484000e+00 -5.044000e+00
Speechiness                5.040000e-02  4.830000e-02  1.540000e-01
Acousticness               1.270000e-01  3.830000e-02  3.350000e-01
Liveness                   3.590000e-01  1.030000e-01  8.490000e-02
Tempo                      1.340020e+02  1.699280e+02  1.669280e+02
Duration (ms)              2.115600e+05  1.418060e+05  1.781470e+05
Valence                    5.890000e-01  4.780000e-01  6.880000e-01

                                      3             4
Highest Charting Position  3.000000e+00  5.000000e+00
Number of Times Charted    5.000000e+00  1.000000e+00
Streams                    3.779946e+07  3.394845e+07
Artist Followers           8.329338e+07  5.473565e+06
```

```
Popularity                 9.800000e+01  9.600000e+01
Danceability               8.080000e-01  7.360000e-01
Energy                     8.970000e-01  7.040000e-01
Loudness                  -3.712000e+00 -7.409000e+00
Speechiness                3.480000e-02  6.150000e-02
Acousticness               4.690000e-02  2.030000e-02
Liveness                   3.640000e-01  5.010000e-02
Tempo                      1.260260e+02  1.499950e+02
Duration (ms)              2.310410e+05  2.120000e+05
Valence                    5.910000e-01  8.940000e-01
```

#Exploratory Data Analysis

[ ]:
```
#Set up for Decision Tree Regression to identify the most important features
↪(that explain the highest proportion of variance relevant to song popularity)



X_train = df_num.drop('Popularity', axis = 1)
y = df_num['Popularity']

numLoops = 500

mean_error = np.zeros(numLoops)

for idx in range(0,numLoops):
  X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
  model = DecisionTreeRegressor(max_depth=5, random_state=0)
  model.fit(X_train,y_train)
  y_pred = model.predict(X_test)
  mean_error[idx] = mean_squared_error(y_test, y_pred)

print(f'RMSE: {np.sqrt(mean_error).mean()*1000}')
print(f'RMSE_std: {np.sqrt(mean_error).std()*1000}')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-20-48c69384102e> in <cell line: 11>()
     10
     11 for idx in range(0,numLoops):
---> 12   X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2
     13   model = DecisionTreeRegressor(max_depth=5, random_state=0)
     14   model.fit(X_train,y_train)

NameError: name 'X' is not defined
```

```
[ ]: max_depths = [1,2,3,4,5,6,7,8,9,10]
rms_depth = np.zeros(len(max_depths))
std_depth = np.zeros(len(max_depths))

numLoops = 500

for n, depth in enumerate(max_depths):
  rms_error = np.zeros(numLoops)

  for idx in range(0,numLoops):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
    model = DecisionTreeRegressor(max_depth=depth)
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    rms_error[idx] = np.sqrt(mean_squared_error(y_test, y_pred))

  rms_depth[n] = rms_error.mean()
  std_depth[n] = rms_error.std( ddof = 1 )
```

```
[ ]: pd.DataFrame( zip( max_depths, rms_depth, std_depth ) )
```

```
[ ]: # Plot result
plt.figure(figsize = (8,5))
plt.plot(max_depths, rms_depth)
plt.xlabel('Max Depth')
plt.ylabel('RMSE')
plt.xlim(0, 10.5)
plt.grid()
```

```
[ ]: # Re run with max depth = 4, and we get lower error than when we ran this␣
 ↪housig data with linear regression
numLoops = 500

rms_error = np.zeros( numLoops )

for idx in range( 0, numLoops ):
  X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2 )
  model = DecisionTreeRegressor( max_depth=4 ) #training the model with␣
 ↪regressors
  model.fit( X_train, y_train )
  y_pred = model.predict( X_test )
  rms_error[idx] = np.sqrt( mean_squared_error( y_test, y_pred ) )

print(f"CV RMSE: {rms_error.mean().round(2)*1000}") #recall RMSE is the mean of␣
 ↪all the RMSE
```

```
#multiplied by 1000 in the last line because the median home value is in the␣
↪thousands so we want to represent our RMSE in the same units: Dollars (as␣
↪the mean of the means)
```

[ ]:
```
import graphviz
from IPython.display import display
from sklearn import tree
```

[ ]:
```
# Option 1 - to train the model we use ALL the features (columns) but the model␣
↪picks the features by looking at all combinations and as a result it chooses␣
↪how many to include at each level.
display(
  graphviz.Source(
    tree.export_graphviz(
      model,
      feature_names = X.columns,
      filled = True,
    )
  )
)
```

[ ]:
```
# Option 2
plt.figure(figsize=(30,15))
tree_plot = tree.plot_tree(
  model,
  feature_names = X.columns,
  filled=True,
)
```

#Forest Regression

[ ]:
```
X = df_num.drop('Popularity', axis = 1)
y = df_num['Popularity']
```

[ ]:
```
# #Set up for Forest Regression to identify the most important features (that␣
↪explain the highest proportion of variance relevant to song popularity)
numLoops = 500

mean_error = np.zeros(numLoops)

for idx in range(0,numLoops):
  X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.2 )
  model = RandomForestRegressor( n_estimators = 10 )
  model.fit( X_train, y_train )
  y_pred = model.predict( X_test )
  mean_error[idx] = mean_squared_error( y_test, y_pred )

print(f'RMSE: {np.sqrt(mean_error).mean()*1000}')
```

```
print(f'RMSE_std: {np.sqrt(mean_error).std()*1000}')
np.sqrt(mean_error)[:50]
```

[ ]:
```
num_trees = range(10,60,10)
cv_loops = 100
rmse_results = np.zeros(len(num_trees))
std_results = np.zeros(len(num_trees))

for n, trees in enumerate(num_trees):
  rmse_cv = np.zeros(cv_loops)
  np.random.seed(42)
  for i in range(cv_loops):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20)
    rfModel = RandomForestRegressor( n_estimators=trees )
    rfModel.fit(X_train, y_train)
    y_pred_rf = rfModel.predict(X_test)
    rmse_cv[i] = np.sqrt(mean_squared_error(y_test, y_pred_rf))

  print(trees,' trees finished.')
  rmse_results[n] = rmse_cv.mean()
  std_results[n] = rmse_cv.std()
#try some more trees...
```

[ ]:
```
pickle.dump(rfModel, open('rfModel.p','wb'))
```

[ ]:
```
plt.plot(num_trees, rmse_results)
plt.xlabel('Tree No.')
plt.ylabel('RMSE')
plt.grid()
#it looks lower without the origin, zoomed in
#put in an origin and it doesn't look that different, but might as well use it..
 ↪.
```

[ ]:
```
pd.DataFrame( zip( rmse_results, std_results, ) )
```

[ ]:
```
print(f'RMSE with 30 trees: {rmse_results[2]*1000}')
```

[ ]:
```
import graphviz
from IPython.display import display
from sklearn import tree
```

[ ]:
```
len(rfModel.estimators_)
```

[ ]:
```
# Display ONE tree from the random forest of 50 trees
display(
  graphviz.Source(
    tree.export_graphviz(
```

```
      rfModel.estimators_[0],
      feature_names = X.columns,
    )
  )
)
#you get 2 to the 15 possibilities - over 300,000
```

[ ]:
```
# Option 2
plt.figure(figsize=(30,15))
tree_plot = tree.plot_tree(
  model,
  feature_names = X.columns,
  filled=True,
)
```

[ ]:
```
importances = rfModel.feature_importances_
forest_importances = pd.Series( importances, index = X.columns )

plt.figure()
# forest_importances.plot.bar()
forest_importances.sort_values( ascending = False ).plot.bar()
plt.title("Feature importances")
plt.ylabel('Feature Importance Score') ;

# the feature importances add up to 100% so this is the % of the data accounted␣
↪for my each feature
```

[ ]:
```
( forest_importances.sort_values( ascending = False ) * 100 ).cumsum()
# this is the cumulative sum of the feature importances and could cut it off at␣
↪prop_tax (there is less than 1% improvement above 96.79 for prop tax, so you␣
↪could drop all those and run it again)
#the percentages are a running total of how much variance accounted for by the␣
↪feature (it's in descending order)
```

[ ]:
```
#One Hot Encode for artist, song name, genre???
```

#Data Visualization

[ ]:
```

```

#Communication of Results

[ ]:
```

```