PREDICTING TRACK SUCCESS ON SPOTIFY USING COVER ART

- Capstone Project
- Student- Sabina Bains
- Presentation Date- 10/07/2022
- Instructor- Abhineet Kulkarni

BUSINESS UNDERSTANDING ¶

Album art has played an important role in music. It gives visual representation and additional context to the story behind an album. Perhaps most importantly, it helped artists sell music. Before streaming was introducted into the world of music, albums were largely judged by their artwork. Iconic albums such as "Abbey Road" by The Beatles and "Nevermind" by Nirvana have artwork that are still frequently talked about today.

However, in recent years streaming services such as Spotify have dominated, with streaming making up 80% of revenue in the U.S. music industry. Artists and labels have been left wondering if this shift from hard copies to streaming has affected the prominence of albums, and therefore album artwork. The Playist Effect is a phenomenon that suggests with the rise of streaming, subscribers largest listen to curated auto-play playlists with individual tracks rather an albums as a whole.

with Spotify leading the music streaming industry, it's crucial for record labels to understand the effectiveness of albums and their artwork in this space.

This analysis will look at the popularity of tracks from Spotify's discovery playlists and their respective artwork to determine which album styles are associated with track success.

DATA UNDERSTANDING

In [1]:

```
!pip3 install spotipy --upgrade
!pip3 install pillow
!pip3 install ipynb
```

```
Requirement already up-to-date: spotipy in /opt/anaconda3/envs/learn-env/lib/python3.
8/site-packages (2.20.0)
Requirement already satisfied, skipping upgrade: redis>=3.5.3 in /opt/anaconda3/envs/
learn-env/lib/python3.8/site-packages (from spotipy) (4.3.4)
Requirement already satisfied, skipping upgrade: urllib3>=1.26.0 in /opt/anaconda3/en
vs/learn-env/lib/python3.8/site-packages (from spotipy) (1.26.11)
Requirement already satisfied, skipping upgrade: six>=1.15.0 in /opt/anaconda3/envs/l
earn-env/lib/python3.8/site-packages (from spotipy) (1.15.0)
Requirement already satisfied, skipping upgrade: requests>=2.25.0 in /opt/anaconda3/e
nvs/learn-env/lib/python3.8/site-packages (from spotipy) (2.28.1)
Requirement already satisfied, skipping upgrade: deprecated>=1.2.3 in /opt/anaconda3/
envs/learn-env/lib/python3.8/site-packages (from redis>=3.5.3->spotipy) (1.2.13)
Requirement already satisfied, skipping upgrade: packaging>=20.4 in /opt/anaconda3/en
vs/learn-env/lib/python3.8/site-packages (from redis>=3.5.3->spotipy) (20.4)
Collecting async-timeout>=4.0.2
  Using cached async_timeout-4.0.2-py3-none-any.whl (5.8 kB)
Requirement already satisfied, skipping upgrade: idna<4,>=2.5 in /opt/anaconda3/envs/
learn-env/lib/python3.8/site-packages (from requests>=2.25.0->spotipy) (2.10)
Requirement already satisfied, skipping upgrade: charset-normalizer<3,>=2 in /opt/ana
conda3/envs/learn-env/lib/python3.8/site-packages (from requests>=2.25.0->spotipy)
(2.1.0)
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in /opt/anaconda
3/envs/learn-env/lib/python3.8/site-packages (from requests>=2.25.0->spotipy) (2021.1
0.8)
Requirement already satisfied, skipping upgrade: wrapt<2,>=1.10 in /opt/anaconda3/env
s/learn-env/lib/python3.8/site-packages (from deprecated>=1.2.3->redis>=3.5.3->spotip
y) (1.12.1)
Requirement already satisfied, skipping upgrade: pyparsing>=2.0.2 in /opt/anaconda3/e
nvs/learn-env/lib/python3.8/site-packages (from packaging>=20.4->redis>=3.5.3->spotip
y) (2.4.7)
Installing collected packages: async-timeout
  Attempting uninstall: async-timeout
    Found existing installation: async-timeout 3.0.1
    Uninstalling async-timeout-3.0.1:
      Successfully uninstalled async-timeout-3.0.1
ERROR: After October 2020 you may experience errors when installing or updating packa
ges. This is because pip will change the way that it resolves dependency conflicts.

We recommend you use --use-feature=2020-resolver to test your packages with the new r
esolver before it becomes the default.

aiohttp 3.6.2 requires async-timeout<4.0,>=3.0, but you'll have async-timeout 4.0.2 w
hich is incompatible.
Successfully installed async-timeout-4.0.2
Requirement already satisfied: pillow in /opt/anaconda3/envs/learn-env/lib/python3.8/
site-packages (7.2.0)
Requirement already satisfied: ipynb in /opt/anaconda3/envs/learn-env/lib/python3.8/s
ite-packages (0.5.1)
```

Importing Packages

In [2]:

```
from sklearn.model_selection import train_test_split
from matplotlib import image as mpimg
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPool2D
import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import collections
import tensorflow.keras.backend as K

# changing colors of output
import matplotlib as mpl
COLOR = 'white'
mpl.rcParams['text.color'] = 'grey'
mpl.rcParams['axes.labelcolor'] = COLOR
mpl.rcParams['xtick.color'] = COLOR
mpl.rcParams['ytick.color'] = COLOR

# importing color coding functions from another file
%run "helper_functions.ipynb"
```

Importing Target Variables

In [3]:

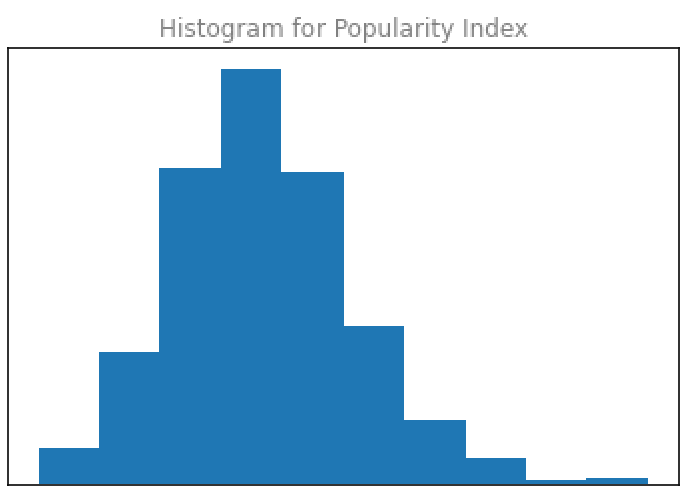
```
# importing dataset created from API pulls in other jupyter notebook.
df_full = pd.read_csv('data/popularity_index.csv')
```

Descriptive Statistics for Target Variable

In [4]:

```
# Looking at descriptive statistics for target variable Popularity In
print(df_full[['key','popularity']].describe())

# Plotting Histogram of data
df_full.hist('popularity')
plt.title('Histogram for Popularity Index')
plt.grid(False)

print('\n\nData is slightly skewed to the right with a mean of 30, wh
```

```
        popularity
count  1860.000000
mean     30.078495
std      11.852113
min       0.000000
25%      22.000000
50%      29.000000
75%      37.000000
max      81.000000


Data is slightly skewed to the right with a mean of 30, which makes sense because thi
s playlist is for up and coming artists / labels, outliers would be more popular musi
c.
```



Printing Genre list

In [5]:
```
for i in list(df_full.playlist_name.unique()):
    print(i)
```

```
Korea
Vietnam
Hip-Hop
Indie
Brasil
2021
Folk
NZ
2019
IE
Country
nan
Dance
Latin
R&B
Rock
GSA
Jazz
Pop
Experimental
Italia
España
EDM
All Genres
```

Preview of Album Artwork

In [6]:

```
# Displaying example of photos downloaded from Spotify API
album_art_example = mpimg.imread(os.getcwd()+"/data/album_art/"+os.li

plt.imshow(album_art_example)
plt.xlabel("X pixel scaling")
plt.ylabel("Y pixels scaling")
plt.show()
```

In [7]:

```
# importing saved album art files as values in dictionary, with key b

# initializing dict
image_dict = {}

# looping through each album art saved
for file in os.listdir("data/album_art_resized/"):

    filename = file.split('.jpg')[0]

    # assigning track name as key, and value as matrix form of album
    try:
        image_dict[filename] = (mpimg.imread('data/album_art_resized/

            # Not including image if it is not in uniform shape
            if image_dict[filename].shape == (60,60):
                    image_dict[filename] = np.stack((image_dict[filename]

                    newshape = image_dict[filename].shape
                    print(file+' was resized to '+str(newshape))

            if image_dict[filename].shape != (60,60,3):
                del image_dict[filename]

    # created error list to observe files that were not read in prope
    except:
        print(file+' did not import')
```

```
Baby Choppa_2021.jpg was resized to (60, 60, 3)
Say Datt_Hip-Hop.jpg was resized to (60, 60, 3)
.DS_Store did not import
Spirito della Domenica_Italia.jpg was resized to (60, 60, 3)
Consolation Prize_Rock.jpg was resized to (60, 60, 3)
Something_NZ.jpg was resized to (60, 60, 3)
PDF_All Genres.jpg was resized to (60, 60, 3)
HEAVY METAL_2021.jpg was resized to (60, 60, 3)
Rave_All Genres.jpg was resized to (60, 60, 3)
Family Man_Rock.jpg was resized to (60, 60, 3)
concussion_2021.jpg was resized to (60, 60, 3)
Lost_NZ.jpg was resized to (60, 60, 3)
Goner feat Kellin Quinn_Rock.jpg was resized to (60, 60, 3)
White Picket Fence_2021.jpg was resized to (60, 60, 3)
At Least_Folk.jpg was resized to (60, 60, 3)
Portals_R&B.jpg was resized to (60, 60, 3)
Inconsciencia_Latin.jpg was resized to (60, 60, 3)
Rave_Dance.jpg was resized to (60, 60, 3)
```

```
BE FREE_R&B.jpg was resized to (60, 60, 3)
Be Your Lover_NZ.jpg was resized to (60, 60, 3)
Grimiest Ever_Dance.jpg was resized to (60, 60, 3)
WORK 4 A SMILE DEMO_R&B.jpg was resized to (60, 60, 3)
The Wolves_Country.jpg was resized to (60, 60, 3)
Black Visa_Hip-Hop.jpg was resized to (60, 60, 3)
```

In [8]:

```
# filtering down dataframe so that only the files that were read in p
df = df_full.loc[df_full.key.isin(image_dict.keys())]
```

In [9]:

```
del_list = []

for key in image_dict.keys():
    if key in df.key.to_list():
        pass
    else:
        print("Could not find associated value for "+key+", delete fr
        del_list.append(key)

for key in del_list:
    del image_dict[key]
```

```
Could not find associated value for 4 Ya Kiss_Experimental, delete from the album art
dictionary
Could not find associated value for .SYSTVMRVSTVRT, delete from the album art diction
ary
Could not find associated value for .SYSTVMRVSTVRT*, delete from the album art dictio
nary
```

In [10]:

```
# Sorting Dictionary to match DataFrame order
image_dict = collections.OrderedDict(sorted(image_dict.items()))
```

In [11]:

```
# Checking if order of names matches, so we can accuractly match feat

if df.key.to_list() == list(image_dict.keys()):
    print('data aligns, ready to process')
else:
    print('data does not align')
```

```
data aligns, ready to process
```

Preliminary analysis of album artwork colors and relation to popularity index

Prior to running CNN models, I wanted to take a look at potential differences in the average popularity values between dominant colors on albums. I also take a look at the average popularity between albums with black and white images vs those with color.

In [12]:
```
# using get_top_colors function from helper functions notebook to loc
top_colors_dict = {}

for k, v in image_dict.items():
    try:
        top_colors_dict[k] = get_top_colors(image_dict[k])
    except:
        print(k, image_dict[k].shape)
```

```
18 feat Paloalto_Korea (60, 60, 3)
```

In [13]:
```
# looping through newly created top_colors_dict to further classify t
top_labels_dict = {}

for k, v in top_colors_dict.items():
    top_labels_dict[k] = []
    for color in top_colors_dict[k]:
         top_labels_dict[k].append(classify(color))
```

In [14]:
```
# deleting albums that only have two colors, as they will produce erı
del_list = []

for k, v in top_labels_dict.items():
    if len(v) != 3:
        del_list.append(k)

for key in del_list:
    del top_labels_dict[key]
    print(key+' was deleted')
```

```
Exit Music_GSA was deleted
I'm Dope_NZ was deleted
So Long_All Genres was deleted
```

In [15]:
```
# putting dict into dataframe
colors = pd.DataFrame(top_labels_dict.items(), columns=['key','color'
```

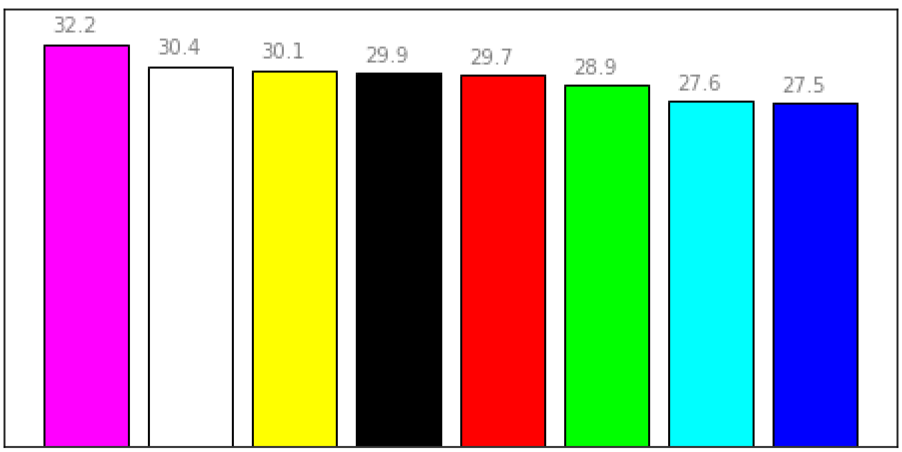
In [16]:
```
# adding column to distinguish black and white images from regular on
colors['b_w'] = colors.color.apply(b_w)
```

In [17]:
```
# merging in the popularity values for analysis
colors = pd.merge(colors, df[['key','popularity']], on='key', how = '
colors[['color 1','color 2','color 3']] = colors.color.apply(pd.Serie
```

In [18]:

```
# grouping by dominant color and sorting for graph output
colors_grouped = pd.DataFrame(colors.groupby('color 1')['popularity']

# creating bar graph
plt.figure(figsize=(8,4))
bars = plt.bar(colors_grouped['color 1'], colors_grouped['popularity'
plt.ylim([0, 35])
plt.ylabel('Average Popularity Index')
plt.xlabel('Dominant Color in Album Artwork')


# adding axes values
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + .08, yval + 1, "{:.1f}".format(yval))

print('''No major difference in mean between dominant color in album
         of shapes and faces in the artwork ''')
```

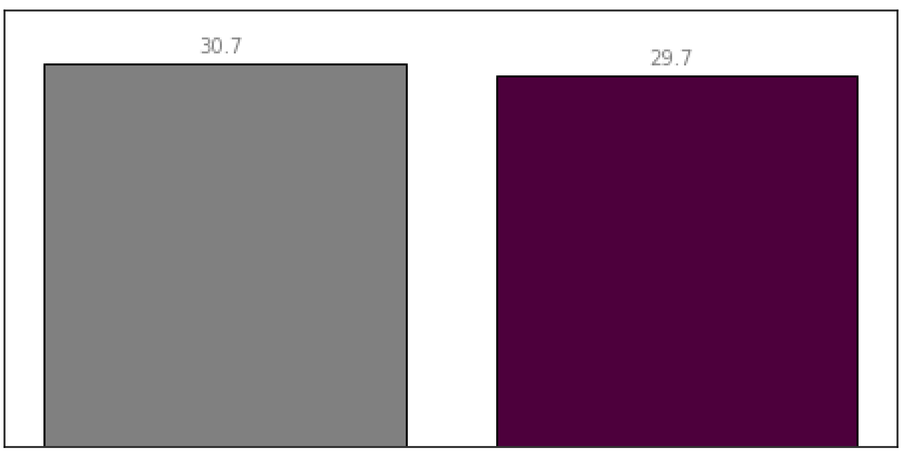
```
No major difference in mean between dominant color in album artwork. This may foressh
adow the model not performing well, but perhaps the CNN model will be able to make se
nse
         of shapes and faces in the artwork
```



In [19]:

```
# grouping by black and white photos and colored photos
b_w_grouped = pd.DataFrame(colors.groupby('b_w')['popularity'].mean()


# creating batplot and adding axes
plt.figure(figsize=(8,4))
bars = plt.bar(b_w_grouped['b_w'], b_w_grouped['popularity'], edgecol
plt.ylim([0, 35])
plt.ylabel('Average Popularity Index')
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + .34, yval + 1, "{:.1f}".format(yval))

print('No major difference in mean between color artwork and black an
```

```
No major difference in mean between color artwork and black and white artwork. Furthe
r support that our model may not generate significant results
```



DATA PREPARATION

In [20]:
```
# Creating feature array from album artwork dictionary
X = np.array(list(image_dict.values()))

# Assuring shape is correct
X.shape
```

```
(1859, 60, 60, 3)
```

In [21]:
```
# Creating target array from df with popularity index
y = np.array(df.popularity).reshape(len(df.popularity), 1)

# Assuring shape is correct
y.shape
```

```
(1859, 1)
```

In [22]:
```
# Split the data into Train and Test, so we can later validate our mc
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
```

In [23]:
```
# dividing Training and testing data by 255 to normalize
X_train = X_train / 255
X_test  = X_test  / 255
```

MODELING

Creating functions to evaluate models more efficiently

In [24]:
```
# creating function to plot history MSE of CNN Model
def plot_history(model, history, X_test, y_test):

    plt.plot(history.history['mse'], label='mse')
    plt.plot(history.history['val_mse'], label = 'val_mse')

    plt.xlabel('Epoch')
    plt.ylabel('Mean Squared Error')
    plt.legend(loc='upper right')

    loss, mse, R2 = model.evaluate(X_test,  y_test)
    return np.sqrt(mse), R2
```

In [25]:

```
# creating function to produce scatter plot of predicted values and t
def plot_scatter(model, X_test):

    y_hat = model.predict(X_test);

    error = y_test - y_hat
    rmse = np.sqrt(np.mean(error**2))

    plt.figure(figsize=(5, 5))
    plt.scatter(y_hat,y_test)
    plt.xlabel("Predicted Value")
    plt.ylabel("True Value")
    yvy = np.linspace(0,80,2)
    plt.plot(yvy, yvy, color='red', linestyle='dashed')
    plt.xlim([0, 80])
    plt.ylim([0, 80])
```

In [26]:

```
# function to calculate R2 score to assess model's ability to predict
def r2_score(y_true, y_pred):
    SS_res =  K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )
```

Model 2: Reduced Pixels

The dimensions for each photo in this run is (60,60,3). This will hopefully run much faster, and therefore will give the ability to add layers and tune parameters. For this model we will use the adam optimizer, which is different to classical stochastic gradient descent due to it's adaptive learning rate as learning occurs. model loss will be MSE, to determine the average distance the predicted values are to the true values.

In [27]:

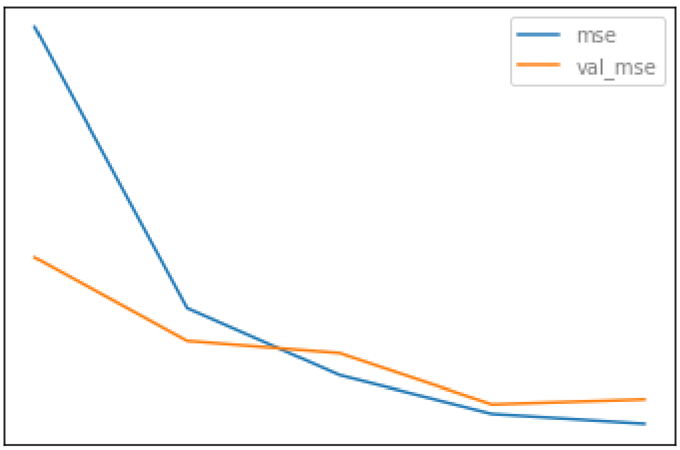
```
model_2 = Sequential()
model_2.add(layers.Conv2D(64, kernel_size=(3,3), input_shape = X_trai
model_2.add(Flatten())
model_2.add(layers.Dense(1, activation='linear'))

model_2.compile(optimizer='adam',
              loss='mse',
              metrics=['mse', r2_score])

history_2 = model_2.fit(X_train, y_train, epochs=5,
                        validation_data=(X_test, y_test))
```

```
Epoch 1/5
44/44 [==============================] - 1s 34ms/step - loss: 306.3106 - mse: 306.310
6 - r2_score: -1.3051 - val_loss: 202.9367 - val_mse: 202.9367 - val_r2_score: -0.547
7
Epoch 2/5
44/44 [==============================] - 1s 32ms/step - loss: 180.2110 - mse: 180.211
0 - r2_score: -0.3388 - val_loss: 165.4612 - val_mse: 165.4612 - val_r2_score: -0.241
1
Epoch 3/5
44/44 [==============================] - 1s 32ms/step - loss: 150.1637 - mse: 150.163
7 - r2_score: -0.1248 - val_loss: 159.9958 - val_mse: 159.9958 - val_r2_score: -0.236
7
Epoch 4/5
44/44 [==============================] - 1s 32ms/step - loss: 132.8277 - mse: 132.827
7 - r2_score: 0.0237 - val_loss: 137.0912 - val_mse: 137.0912 - val_r2_score: -0.0357
Epoch 5/5
44/44 [==============================] - 2s 37ms/step - loss: 128.4172 - mse: 128.417
2 - r2_score: 0.0576 - val_loss: 139.3533 - val_mse: 139.3533 - val_r2_score: -0.0657
```

In [28]:

```
rmse_2, r2_2 = plot_history(model_2, history_2, X_test, y_test)

print('''\n\nOur second model after reducing pixels outperforms our b
with a significantly faster computiation time. and a RMSE of {:.1f}.
```

```
15/15 [==============================] - 0s 12ms/step - loss: 139.3533 - mse: 139.353
3 - r2_score: -0.0657


Our second model after reducing pixels outperforms our baseline model with the larger
dimensions,
with a significantly faster computiation time. and a RMSE of 11.8. This will allow us
to increase epochs and add layers.
```



Model #3: increased layers and epochs

Now that our model runs quicker, I will add another convolution and pooling layer. I will increase epochs as well. the rest of the parameters will remain the same

In [29]:

```
# as it's good to use a repeating structure for cnns, lets add some m
model_3 = Sequential()
model_3.add(layers.Conv2D(64, kernel_size =(3, 3), input_shape = X_tr
model_3.add(layers.MaxPooling2D(pool_size =(2, 2)))
model_3.add(layers.Conv2D(64, (3, 3)))
model_3.add(layers.MaxPooling2D(pool_size =(2, 2)))
model_3.add(Flatten())
model_3.add(layers.Dense(1, activation='linear'))


# training
model_3.compile(optimizer='adam',
                loss='mse',
                metrics=['mse', r2_score])

# fitting
history_3 = model_3.fit(X_train, y_train, epochs=15,
                        validation_data=(X_test, y_test))
```

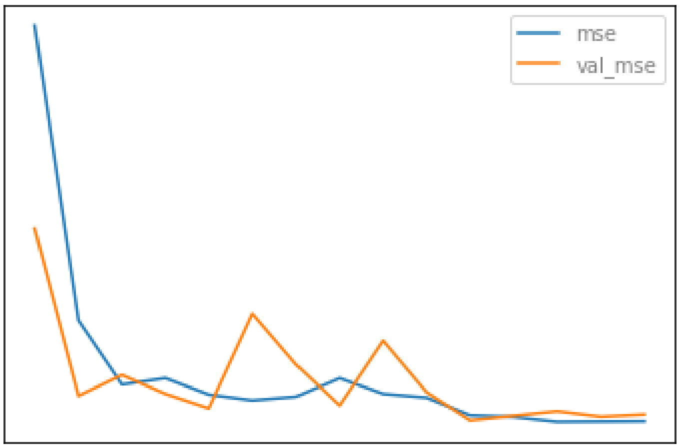
```
Epoch 1/15
44/44 [==============================] - 5s 103ms/step - loss: 322.4494 - mse: 322.44
94 - r2_score: -1.5662 - val_loss: 228.0503 - val_mse: 228.0503 - val_r2_score: -0.75
40
Epoch 2/15
44/44 [==============================] - 4s 100ms/step - loss: 185.3055 - mse: 185.30
55 - r2_score: -0.3358 - val_loss: 150.2081 - val_mse: 150.2081 - val_r2_score: -0.15
94
Epoch 3/15
44/44 [==============================] - 5s 102ms/step - loss: 155.5364 - mse: 155.53
64 - r2_score: -0.1524 - val_loss: 159.9653 - val_mse: 159.9653 - val_r2_score: -0.21
79
Epoch 4/15
44/44 [==============================] - 4s 100ms/step - loss: 158.4972 - mse: 158.49
72 - r2_score: -0.1602 - val_loss: 151.1226 - val_mse: 151.1226 - val_r2_score: -0.16
60
Epoch 5/15
44/44 [==============================] - 4s 102ms/step - loss: 150.7871 - mse: 150.78
71 - r2_score: -0.0973 - val_loss: 144.3956 - val_mse: 144.3956 - val_r2_score: -0.10
04
Epoch 6/15
44/44 [==============================] - 4s 102ms/step - loss: 148.1056 - mse: 148.10
56 - r2_score: -0.0958 - val_loss: 188.3815 - val_mse: 188.3815 - val_r2_score: -0.43
05
Epoch 7/15
44/44 [==============================] - 4s 100ms/step - loss: 149.7861 - mse: 149.78
61 - r2_score: -0.0860 - val_loss: 164.8248 - val_mse: 164.8248 - val_r2_score: -0.25
06
Epoch 8/15
44/44 [==============================] - 5s 103ms/step - loss: 158.4757 - mse: 158.47
```

```
57 - r2_score: -0.1672 - val_loss: 145.7368 - val_mse: 145.7368 - val_r2_score: -0.12
74
Epoch 9/15
44/44 [==============================] - 4s 101ms/step - loss: 151.1282 - mse: 151.12
82 - r2_score: -0.1319 - val_loss: 175.8534 - val_mse: 175.8534 - val_r2_score: -0.33
26
Epoch 10/15
44/44 [==============================] - 4s 101ms/step - loss: 149.3684 - mse: 149.36
84 - r2_score: -0.0960 - val_loss: 151.8857 - val_mse: 151.8857 - val_r2_score: -0.18
28
Epoch 11/15
44/44 [==============================] - 5s 109ms/step - loss: 141.2272 - mse: 141.22
72 - r2_score: -0.0283 - val_loss: 138.8681 - val_mse: 138.8681 - val_r2_score: -0.06
18
Epoch 12/15
44/44 [==============================] - 4s 102ms/step - loss: 140.4764 - mse: 140.47
64 - r2_score: -0.0298 - val_loss: 141.0142 - val_mse: 141.0142 - val_r2_score: -0.07
57
Epoch 13/15
44/44 [==============================] - 4s 102ms/step - loss: 138.2346 - mse: 138.23
46 - r2_score: -0.0136 - val_loss: 143.0794 - val_mse: 143.0794 - val_r2_score: -0.10
38
Epoch 14/15
44/44 [==============================] - 4s 102ms/step - loss: 138.3520 - mse: 138.35
20 - r2_score: 0.0054 - val_loss: 140.6028 - val_mse: 140.6028 - val_r2_score: -0.067
4
Epoch 15/15
44/44 [==============================] - 4s 102ms/step - loss: 138.4865 - mse: 138.48
65 - r2_score: -0.0043 - val_loss: 141.6201 - val_mse: 141.6201 - val_r2_score: -0.07
35
```

In [30]:

```
rmse_3, r2_3 = plot_history(model_3, history_3, X_test, y_test)

print('''\n\nThe third model iteration just barely outperforms our pr
so we will continue to add layers / increase epochs'''.format(rmse_3)
```

```
15/15 [==============================] - 0s 26ms/step - loss: 141.6201 - mse: 141.620
1 - r2_score: -0.0735


The third model iteration just barely outperforms our previous model with an RMSE of
11.9, testing data doesn't show any signs of overfitting,
so we will continue to add layers / increase epochs
```



Model 4. further increase in layers and epochs

Checking to see if adding yet another conv and pooling later will improve results

In [40]:

```
# adding yet another Conv and Pooling layer.
model_4 = Sequential()
model_4.add(layers.Conv2D(64, kernel_size=(3,3), input_shape = X_trai
model_4.add(layers.MaxPooling2D((2, 2), strides=(2,2)))
model_4.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_4.add(layers.MaxPooling2D((2, 2), strides=(2,2)))
model_4.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_4.add(layers.MaxPooling2D((2, 2)))
model_4.add(Flatten())
model_4.add(layers.Dense(1, activation='linear'))

model_4.compile(optimizer='adam',
                loss='mse',
                metrics=['mse', r2_score])

# fitting the model
history_4 = model_4.fit(X_train, y_train, epochs=25,
                        validation_data=(X_test, y_test))
```

```
Epoch 1/25
44/44 [==============================] - 5s 118ms/step - loss: 360.6615 - mse: 360.6
615 - r2_score: -1.6632 - val_loss: 226.2244 - val_mse: 226.2244 - val_r2_score: -0.
7502
Epoch 2/25
44/44 [==============================] - 6s 125ms/step - loss: 196.3481 - mse: 196.3
481 - r2_score: -0.4610 - val_loss: 179.3866 - val_mse: 179.3866 - val_r2_score: -0.
3722
Epoch 3/25
44/44 [==============================] - 5s 125ms/step - loss: 161.5405 - mse: 161.5
405 - r2_score: -0.1853 - val_loss: 150.8046 - val_mse: 150.8046 - val_r2_score: -0.
1647
Epoch 4/25
44/44 [==============================] - 5s 117ms/step - loss: 154.8679 - mse: 154.8
679 - r2_score: -0.1524 - val_loss: 158.6795 - val_mse: 158.6795 - val_r2_score: -0.
2145
Epoch 5/25
44/44 [==============================] - 5s 121ms/step - loss: 149.2170 - mse: 149.2
170 - r2_score: -0.0930 - val_loss: 144.7128 - val_mse: 144.7128 - val_r2_score: -0.
1173
Epoch 6/25
44/44 [==============================] - 5s 123ms/step - loss: 149.8006 - mse: 149.8
006 - r2_score: -0.1034 - val_loss: 155.6062 - val_mse: 155.6062 - val_r2_score: -0.
2152
Epoch 7/25
44/44 [==============================] - 6s 127ms/step - loss: 155.4905 - mse: 155.4
905 - r2_score: -0.1337 - val_loss: 154.7637 - val_mse: 154.7637 - val_r2_score: -0.
2092
Epoch 8/25
44/44 [==============================] - 5s 116ms/step - loss: 147.4949 - mse: 147.4
```

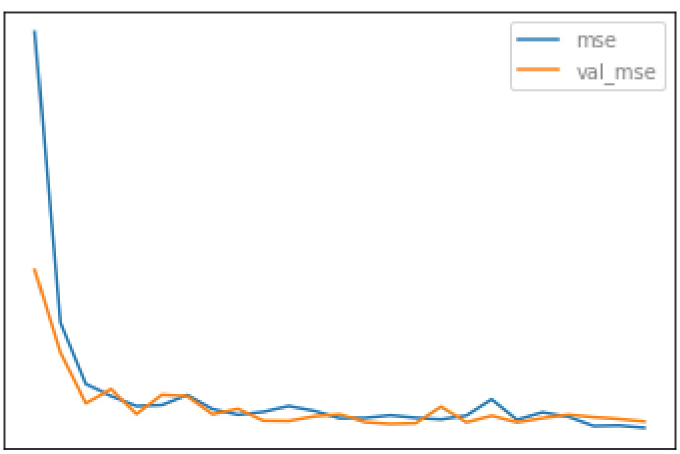
```
949 - r2_score: -0.0719 - val_loss: 144.5546 - val_mse: 144.5546 - val_r2_score: -0.
1050
Epoch 9/25
44/44 [==============================] - 5s 112ms/step - loss: 144.3258 - mse: 144.3
258 - r2_score: -0.0501 - val_loss: 147.6428 - val_mse: 147.6428 - val_r2_score: -0.
1241
Epoch 10/25
44/44 [==============================] - 5s 115ms/step - loss: 145.9433 - mse: 145.9
433 - r2_score: -0.0702 - val_loss: 140.9322 - val_mse: 140.9322 - val_r2_score: -0.
0808
Epoch 11/25
44/44 [==============================] - 5s 114ms/step - loss: 149.2540 - mse: 149.2
540 - r2_score: -0.0820 - val_loss: 140.7653 - val_mse: 140.7653 - val_r2_score: -0.
0802
Epoch 12/25
44/44 [==============================] - 5s 113ms/step - loss: 146.4921 - mse: 146.4
921 - r2_score: -0.0775 - val_loss: 143.4161 - val_mse: 143.4161 - val_r2_score: -0.
1079
Epoch 13/25
44/44 [==============================] - 5s 120ms/step - loss: 142.3577 - mse: 142.3
577 - r2_score: -0.0365 - val_loss: 144.5236 - val_mse: 144.5236 - val_r2_score: -0.
0984
Epoch 14/25
44/44 [==============================] - 5s 116ms/step - loss: 142.4725 - mse: 142.4
725 - r2_score: -0.0337 - val_loss: 140.1429 - val_mse: 140.1429 - val_r2_score: -0.
0679
Epoch 15/25
44/44 [==============================] - 5s 118ms/step - loss: 144.0080 - mse: 144.0
080 - r2_score: -0.0525 - val_loss: 139.0889 - val_mse: 139.0889 - val_r2_score: -0.
0649
Epoch 16/25
44/44 [==============================] - 5s 114ms/step - loss: 142.5732 - mse: 142.5
732 - r2_score: -0.0487 - val_loss: 139.5135 - val_mse: 139.5135 - val_r2_score: -0.
0632
Epoch 17/25
44/44 [==============================] - 5s 116ms/step - loss: 141.6727 - mse: 141.6
727 - r2_score: -0.0252 - val_loss: 148.8573 - val_mse: 148.8573 - val_r2_score: -0.
1525
Epoch 18/25
44/44 [==============================] - 5s 116ms/step - loss: 143.8257 - mse: 143.8
257 - r2_score: -0.0403 - val_loss: 140.0019 - val_mse: 140.0019 - val_r2_score: -0.
0664
Epoch 19/25
44/44 [==============================] - 5s 119ms/step - loss: 153.0458 - mse: 153.0
458 - r2_score: -0.1253 - val_loss: 143.7627 - val_mse: 143.7627 - val_r2_score: -0.
1078
Epoch 20/25
44/44 [==============================] - 5s 115ms/step - loss: 141.4686 - mse: 141.4
686 - r2_score: -0.0253 - val_loss: 139.9798 - val_mse: 139.9798 - val_r2_score: -0.
0666
Epoch 21/25
44/44 [==============================] - 5s 121ms/step - loss: 145.6800 - mse: 145.6
800 - r2_score: -0.0573 - val_loss: 142.3870 - val_mse: 142.3870 - val_r2_score: -0.
0928
Epoch 22/25
44/44 [==============================] - 5s 115ms/step - loss: 143.2239 - mse: 143.2
239 - r2_score: -0.0403 - val_loss: 144.3155 - val_mse: 144.3155 - val_r2_score: -0.
```

```
0966
Epoch 23/25
44/44 [==============================] - 5s 115ms/step - loss: 137.9131 - mse: 137.9
131 - r2_score: -0.0156 - val_loss: 142.9554 - val_mse: 142.9554 - val_r2_score: -0.
0959
Epoch 24/25
44/44 [==============================] - 5s 120ms/step - loss: 138.2183 - mse: 138.2
183 - r2_score: 0.0047 - val_loss: 141.8870 - val_mse: 141.8870 - val_r2_score: -0.0
873
Epoch 25/25
44/44 [==============================] - 5s 118ms/step - loss: 136.9081 - mse: 136.9
081 - r2_score: 0.0058 - val_loss: 140.4769 - val_mse: 140.4769 - val_r2_score: -0.0
685
```

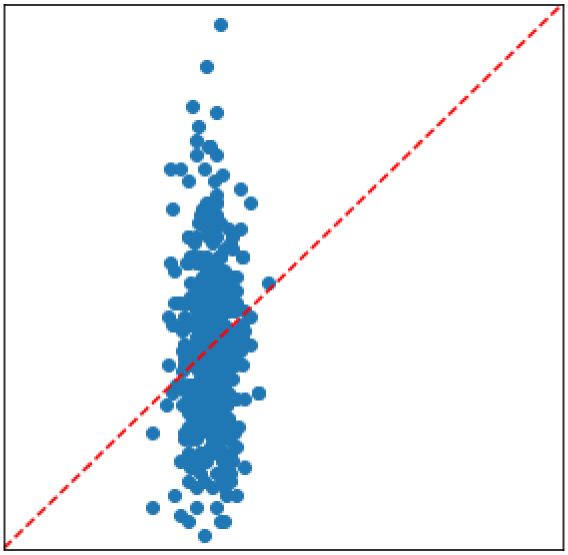
In [59]:

```
rmse_4, r2_4 = plot_history(model_4, history_4, X_test, y_test)

print('''\n\nThe fourth model does slightly worse than model 3, with
doesnt seem to have any predictive power. We will begin to tune hyper
is useful. Although the validation MSE does not show any signs of ove
```

```
15/15 [==============================] - 0s 28ms/step - loss: 140.4769 - mse: 140.476
9 - r2_score: -0.0685


The fourth model does slightly worse than model 3, with an RMSE of 11.9. Looking at t
he below scatterplot this model
doesnt seem to have any predictive power. We will begin to tune hyperparameters, but
will need to look at an R Squared value to determine if this model
is useful. Although the validation MSE does not show any signs of overfitting, MSE se
ems to be plateauing, so adding layers doesn't seem as if it would help.
```



In [42]:
```
# scatterplot shows model is not predicting well
plot_scatter(model_4, X_test)
```



Model 5: tuning hyperparameters with function

removing additional layer, and creating function to test hyperparameters and other model features, such as padding, pooling layer size, kernel size, activation type, and epochs

In [52]:

```
# building a function to test hyperparameters
def build_cnn(X_train, y_train, X_test, y_test, neurons, kernel_size,

    # building the model
    model = Sequential()
    model.add(layers.Conv2D(neurons, kernel_size=kernel_size, activat
    model.add(layers.MaxPooling2D(pool_size = pool_size))
    model.add(layers.Conv2D(neurons, kernel_size=kernel_size, activat
    model.add(layers.MaxPooling2D(pool_size = pool_size))
    model.add(Flatten())
    model.add(layers.Dense(1, activation=activation))

    # training
    model.compile(optimizer='adam', loss='mse', metrics=['mse', r2_sc

    # fitting
    history = model.fit(X_train, y_train, epochs=epochs, validation_d

    return model, history
```

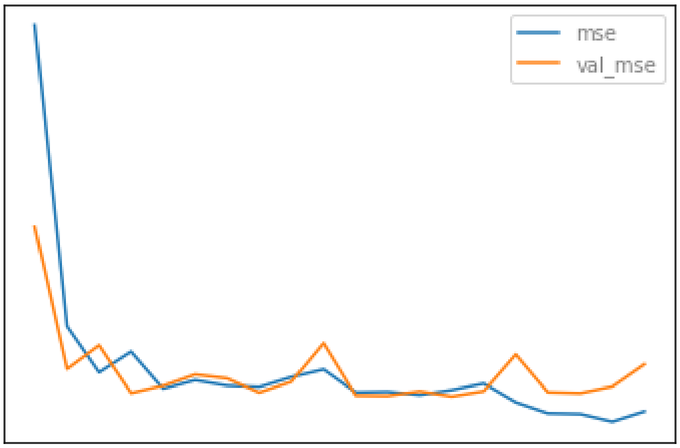
In [53]:
```
model_5, history_5 = build_cnn(X_train, y_train, X_test, y_test, 64,
```

```
Epoch 1/20
44/44 [==============================] - 4s 102ms/step - loss: 333.0540 - mse: 333.0
540 - r2_score: -1.5642 - val_loss: 228.2571 - val_mse: 228.2571 - val_r2_score: -0.
7538
Epoch 2/20
44/44 [==============================] - 4s 100ms/step - loss: 176.5565 - mse: 176.5
565 - r2_score: -0.2950 - val_loss: 154.4023 - val_mse: 154.4023 - val_r2_score: -0.
2000
Epoch 3/20
44/44 [==============================] - 4s 98ms/step - loss: 152.6666 - mse: 152.66
66 - r2_score: -0.1122 - val_loss: 166.7142 - val_mse: 166.7142 - val_r2_score: -0.2
654
Epoch 4/20
44/44 [==============================] - 5s 105ms/step - loss: 163.4061 - mse: 163.4
061 - r2_score: -0.2219 - val_loss: 141.9107 - val_mse: 141.9107 - val_r2_score: -0.
0842
Epoch 5/20
44/44 [==============================] - 5s 104ms/step - loss: 144.2053 - mse: 144.2
053 - r2_score: -0.0519 - val_loss: 145.7878 - val_mse: 145.7878 - val_r2_score: -0.
1245
Epoch 6/20
44/44 [==============================] - 4s 101ms/step - loss: 148.7088 - mse: 148.7
088 - r2_score: -0.0810 - val_loss: 151.6102 - val_mse: 151.6102 - val_r2_score: -0.
1735
Epoch 7/20
44/44 [==============================] - 4s 101ms/step - loss: 145.8558 - mse: 145.8
558 - r2_score: -0.0623 - val_loss: 149.5899 - val_mse: 149.5899 - val_r2_score: -0.
1546
Epoch 8/20
44/44 [==============================] - 4s 102ms/step - loss: 145.1119 - mse: 145.1
119 - r2_score: -0.0699 - val_loss: 142.1795 - val_mse: 142.1795 - val_r2_score: -0.
0861
Epoch 9/20
44/44 [==============================] - 5s 103ms/step - loss: 150.2796 - mse: 150.2
796 - r2_score: -0.1143 - val_loss: 147.8442 - val_mse: 147.8442 - val_r2_score: -0.
1191
Epoch 10/20
44/44 [==============================] - 5s 104ms/step - loss: 154.2424 - mse: 154.2
424 - r2_score: -0.1511 - val_loss: 167.7942 - val_mse: 167.7942 - val_r2_score: -0.
2708
Epoch 11/20
44/44 [==============================] - 5s 103ms/step - loss: 142.3040 - mse: 142.3
040 - r2_score: -0.0425 - val_loss: 140.6324 - val_mse: 140.6324 - val_r2_score: -0.
0668
Epoch 12/20
44/44 [==============================] - 5s 104ms/step - loss: 142.5906 - mse: 142.5
906 - r2_score: -0.0311 - val_loss: 140.4629 - val_mse: 140.4629 - val_r2_score: -0.
0726
Epoch 13/20
44/44 [==============================] - 5s 103ms/step - loss: 140.9421 - mse: 140.9
421 - r2_score: -0.0323 - val_loss: 142.8179 - val_mse: 142.8179 - val_r2_score: -0.
0819
Epoch 14/20
```

```
44/44 [==============================] - 5s 105ms/step - loss: 143.4640 - mse: 143.4
640 - r2_score: -0.0514 - val_loss: 140.1335 - val_mse: 140.1335 - val_r2_score: -0.
0712
Epoch 15/20
44/44 [==============================] - 5s 106ms/step - loss: 146.9365 - mse: 146.9
365 - r2_score: -0.0827 - val_loss: 142.8729 - val_mse: 142.8729 - val_r2_score: -0.
0828
Epoch 16/20
44/44 [==============================] - 5s 106ms/step - loss: 137.1289 - mse: 137.1
289 - r2_score: -0.0187 - val_loss: 161.9919 - val_mse: 161.9919 - val_r2_score: -0.
2675
Epoch 17/20
44/44 [==============================] - 5s 109ms/step - loss: 131.4365 - mse: 131.4
365 - r2_score: 0.0531 - val_loss: 142.3199 - val_mse: 142.3199 - val_r2_score: -0.0
940
Epoch 18/20
44/44 [==============================] - 5s 109ms/step - loss: 131.0851 - mse: 131.0
851 - r2_score: 0.0476 - val_loss: 141.8284 - val_mse: 141.8284 - val_r2_score: -0.0
801
Epoch 19/20
44/44 [==============================] - 5s 106ms/step - loss: 127.1193 - mse: 127.1
193 - r2_score: 0.0631 - val_loss: 145.3069 - val_mse: 145.3069 - val_r2_score: -0.0
968
Epoch 20/20
44/44 [==============================] - 5s 106ms/step - loss: 132.4163 - mse: 132.4
163 - r2_score: 0.0328 - val_loss: 156.8836 - val_mse: 156.8836 - val_r2_score: -0.2
073
```

In [56]:
```
rmse_5, r2_5 = plot_history(model_5, history_5, X_test, y_test)

print('''\n\nThe fifth model has an RMSE of {:.1f}, which underperfor
```

```
15/15 [==============================] - 0s 25ms/step - loss: 156.8836 - mse: 156.883
6 - r2_score: -0.2073


The fifth model has an RMSE of 12.5, which underperforms in comparison to the other m
odels even while fine tuning. It generated an R Squared of -0.2
```



FINAL MODEL EVALUATION

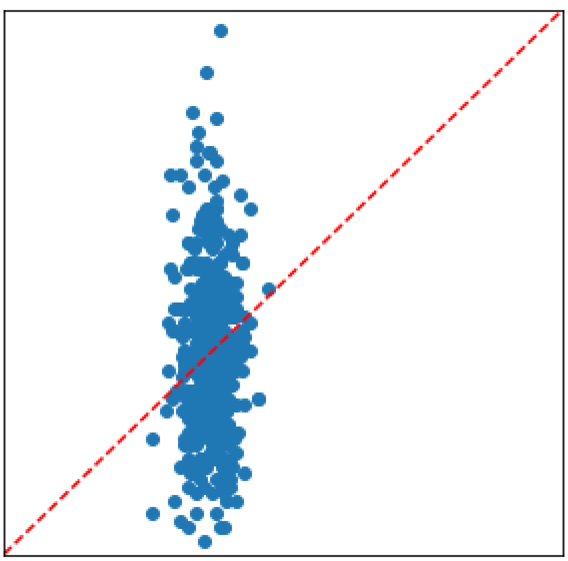
In [57]:
```
# creating loop to look at all model results side by side
for idx, i in enumerate([rmse_2, rmse_3, rmse_4, rmse_5]):
    print('model '+str(idx+2)+" RMSE: "+str(i))
```

```
model 2 RMSE: 11.804800602328791
model 3 RMSE: 11.900423161011242
model 4 RMSE: 11.852294416465122
model 5 RMSE: 12.525318597027033
```

In [58]:

```
print('''
After using the create_cnn function to test hyperparameters to furthe
linear activation, and no padding. This model was chosen because it g
others, it is not an effective predictor for Spotify's popularity ind
This essentially tells us the model is about as efficient at predicti
from the scatter plot.
'''.format(rmse_4, r2_4, y_test.mean()))

plot_scatter(model_4, X_test)
```

```
After using the create_cnn function to test hyperparameters to further fine tune, The
final model chosen was model 4, with a filter size of 3 x 3, a pooling size of 2 x 2,
linear activation, and no padding. This model was chosen because it generated the low
est Root Mean Squared Error value of 11.9. Although this model outperformed the
others, it is not an effective predictor for Spotify's popularity index. The scatterp
lot of true vs. predicted values demonstrate this, as well as it's R Squared score of
-0.1.
This essentially tells us the model is about as efficient at predicting popularity th
an a simple averaging of popularity, 30.1, would be able to predict, which is evident
from the scatter plot.
```



CONCLUSION

Our final model was unsuccessful in predicting Spotify popularity by album artwork.

Although there are other hyperparameters to tune and methods to try, I do not believe any model could have successfully predicted our target variable solely on album artwork. With a switch in streaming for music consumption, it's highly possible that users no longer focus on album artwork. With more time I would attempt to support this theory by running the same analysis on albums from the 80's up to the early 2000's. I would also look into finding a different target variable to measure "success", as Spotify's metric is based on recency, which is not relevant. Lastly, I would run a NLP analysis on track name, and see if this has any effect on popularity among Spotify listeners.

Based on the above results, I would make three recommendations to the client.

- Consider allocating resources away from album art creation, as artwork style doesn't seem to affect popularity on Spotify
- Commission a study on album and/or song name to understand naming effect on Spotify popularity
- If artwork context is crucial to telling the album story, consider other means of distribution outside of Spotify, such as physical stores, as users don't seem engaged with artwork on Spotify