IEOR142_Fiinal_Project_Notebook

May 5, 2023

1 IEOR 142 Final Project - Predicting Spotify Track Popularity

Mina Baghai, Abhigyan Biswas, Toufiq Hossain, Naya Lee, Luna Ragot

2 Data Cleaning

[ ]:
```
# loading the dataset
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[1]:
```
import numpy as np
import pandas as pd
```

[ ]:
```
#artists = pd.read_csv('/content/drive/MyDrive/IEOR_142_Project/Data_Sets/
↪artists.csv')
#tracks = pd.read_csv('/content/drive/MyDrive/IEOR_142_Project/Data_Sets/tracks.
↪csv')

artists = pd.read_csv('/content/drive/MyDrive/artists.csv')
tracks = pd.read_csv('/content/drive/MyDrive/tracks.csv')
```

[3]:
```
artists.head()
```

[3]:
```
                       id  followers genres  \
0  0DheY5irMjBUeLybbCUEZ2        0.0     []
1  0DlhY15l3wsrnlfGio2bjU        5.0     []
2  0DmRESX2JknGPQy015yxg7        0.0     []
3  0DmhnbHjm1qw6NCYPeZNgJ        0.0     []
4  0Dn11fWM7vHQ3rinvWEl4E        2.0     []

                                            name  popularity
0  Armid & Amir Zare Pashai feat. Sara Rouzbehani          0
1                                                    0
2                                          Sadaa          0
```

```
3                           Tra'gruda          0
4                 Ioannis Panoutsopoulos          0
```

[4]:

```python
tracks.head()
```

[4]:

```
                       id                                  name  popularity  \
0  35iwgR4jXetI318WEWsa1Q                                 Carve           6
1  021ht4sdgPcrDgSk7JTbKY  Capítulo 2.16 - Banquero Anarquista           0
2  07A5yehtSnoedViJAZkNnc   Vivo para Quererte - Remasterizado           0
3  08FmqUhxtyLTn6pAh6bk45       El Prisionero - Remasterizado           0
4  08y9GfoqCWfOGsKdwojr5e                  Lady of the Evening           0

   duration_ms  explicit              artists                  id_artists  \
0       126903         0              ['Uli']  ['45tIt06XoI0Iio4LBEVpls']
1        98200         0  ['Fernando Pessoa']  ['14jtPCOoNZwquk5wd9DxrY']
2       181640         0  ['Ignacio Corsini']  ['5LiOoJbxVSAMkBS2fUm3X2']
3       176907         0  ['Ignacio Corsini']  ['5LiOoJbxVSAMkBS2fUm3X2']
4       163080         0      ['Dick Haymes']  ['3BiJGZsyX9sJchTqcSA7Su']

  release_date  danceability  energy  key  loudness  mode  speechiness  \
0   1922-02-22         0.645  0.4450    0   -13.338     1       0.4510
1   1922-06-01         0.695  0.2630    0   -22.136     1       0.9570
2   1922-03-21         0.434  0.1770    1   -21.180     1       0.0512
3   1922-03-21         0.321  0.0946    7   -27.961     1       0.0504
4         1922         0.402  0.1580    3   -16.900     0       0.0390

   acousticness  instrumentalness  liveness  valence    tempo  time_signature
0         0.674            0.7440     0.151    0.127  104.851               3
1         0.797            0.0000     0.148    0.655  102.009               1
2         0.994            0.0218     0.212    0.457  130.418               5
3         0.995            0.9180     0.104    0.397  169.980               3
4         0.989            0.1300     0.311    0.196  103.220               4
```

[5]:

```python
#add length of id to see if there's more than one artist
tracks['len artist id'] = [len(i) for i in tracks['id_artists']]

#drop tracks with more than one artist
tracks = tracks[tracks['len artist id'] <= 26]

tracks.head()
```

[5]:

```
                       id                                  name  popularity  \
0  35iwgR4jXetI318WEWsa1Q                                 Carve           6
1  021ht4sdgPcrDgSk7JTbKY  Capítulo 2.16 - Banquero Anarquista           0
2  07A5yehtSnoedViJAZkNnc   Vivo para Quererte - Remasterizado           0
3  08FmqUhxtyLTn6pAh6bk45       El Prisionero - Remasterizado           0
4  08y9GfoqCWfOGsKdwojr5e                  Lady of the Evening           0
```

```
   duration_ms  explicit                artists                 id_artists  \
0       126903         0                ['Uli']  ['45tIt06XoI0Iio4LBEVpls']
1        98200         0  ['Fernando Pessoa']  ['14jtPCOoNZwquk5wd9DxrY']
2       181640         0  ['Ignacio Corsini']  ['5LiOoJbxVSAMkBS2fUm3X2']
3       176907         0  ['Ignacio Corsini']  ['5LiOoJbxVSAMkBS2fUm3X2']
4       163080         0      ['Dick Haymes']  ['3BiJGZsyX9sJchTqcSA7Su']

  release_date  danceability  energy  …  loudness  mode  speechiness  \
0   1922-02-22         0.645  0.4450  …   -13.338     1       0.4510
1   1922-06-01         0.695  0.2630  …   -22.136     1       0.9570
2   1922-03-21         0.434  0.1770  …   -21.180     1       0.0512
3   1922-03-21         0.321  0.0946  …   -27.961     1       0.0504
4         1922         0.402  0.1580  …   -16.900     0       0.0390

   acousticness  instrumentalness  liveness  valence    tempo  time_signature  \
0         0.674            0.7440     0.151    0.127  104.851               3
1         0.797            0.0000     0.148    0.655  102.009               1
2         0.994            0.0218     0.212    0.457  130.418               5
3         0.995            0.9180     0.104    0.397  169.980               3
4         0.989            0.1300     0.311    0.196  103.220               4

   len artist id
0             26
1             26
2             26
3             26
4             26

[5 rows x 21 columns]
```

[6]:
```python
#cleaning id_artists to match id of artists dataset
tracks['simplified_artist_id'] = [i.replace("['", '').replace("']", '') for i␣
 ↪in tracks['id_artists']]

#merging datasets
tracks = tracks.merge(artists, left_on='simplified_artist_id', right_on='id')
tracks.head()
```

[6]:
```
                     id_x    name_x  popularity_x  duration_ms  explicit  \
0  35iwgR4jXetI318WEWsa1Q     Carve             6       126903         0
1  0PH9AACae1f957JAavh0l2  Lazy Boi             0       157333         0
2  2SiNuAZ6jIU9xhClRKXcST    Sketch             0        87040         0
3  4vV7uBcF2AnjNT0ejBS5oL   L'enfer             0        40000         0
4  598LlBn6jpEpVbLjmZPsYV  Graphite             0       104400         0

  artists                  id_artists release_date  danceability   energy  \
```

```
0  ['Uli']  ['45tIt06XoI0Iio4LBEVpls']   1922-02-22         0.645  0.44500
1  ['Uli']  ['45tIt06XoI0Iio4LBEVpls']   1922-02-22         0.298  0.46000
2  ['Uli']  ['45tIt06XoI0Iio4LBEVpls']   1922-02-22         0.634  0.00399
3  ['Uli']  ['45tIt06XoI0Iio4LBEVpls']   1922-02-22         0.657  0.32500
4  ['Uli']  ['45tIt06XoI0Iio4LBEVpls']   1922-02-22         0.644  0.68400

   …  valence    tempo  time_signature  len artist id  \
0  …    0.127  104.851               3             26
1  …    0.402   87.921               4             26
2  …    0.396   79.895               4             26
3  …    0.105   81.944               5             26
4  …    0.138  100.031               4             26

     simplified_artist_id                    id_y  followers  genres  name_y  \
0  45tIt06XoI0Iio4LBEVpls  45tIt06XoI0Iio4LBEVpls       91.0      []     Uli
1  45tIt06XoI0Iio4LBEVpls  45tIt06XoI0Iio4LBEVpls       91.0      []     Uli
2  45tIt06XoI0Iio4LBEVpls  45tIt06XoI0Iio4LBEVpls       91.0      []     Uli
3  45tIt06XoI0Iio4LBEVpls  45tIt06XoI0Iio4LBEVpls       91.0      []     Uli
4  45tIt06XoI0Iio4LBEVpls  45tIt06XoI0Iio4LBEVpls       91.0      []     Uli

   popularity_y
0             4
1             4
2             4
3             4
4             4

[5 rows x 27 columns]
```

[7]:
```python
#cleaning columns and column names
tracks.rename(columns={'id_x': 'track_id', 'name_x': 'track_name',␣
↪'popularity_x': 'track_popularity', 'simplified_artist_id': 'artist_id',␣
↪'name_y': 'artist_name', 'popularity_y': 'artist_popularity'}, inplace=True)
tracks.drop(['artists', 'id_artists', 'id_y', 'len artist id'], axis=1,␣
↪inplace=True)
tracks.head()
```

[7]:
```
                 track_id track_name  track_popularity  duration_ms  explicit  \
0  35iwgR4jXetI318WEWsa1Q      Carve                 6       126903         0
1  0PH9AACae1f957JAavh0l2   Lazy Boi                 0       157333         0
2  2SiNuAZ6jIU9xhClRKXcST     Sketch                 0        87040         0
3  4vV7uBcF2AnjNT0ejBS5oL    L'enfer                 0        40000         0
4  598LlBn6jpEpVbLjmZPsYV   Graphite                 0       104400         0

  release_date  danceability   energy  key  loudness  …  instrumentalness  \
0   1922-02-22         0.645  0.44500    0   -13.338  …             0.744
1   1922-02-22         0.298  0.46000    1   -18.645  …             0.856
```

```
2   1922-02-22         0.634  0.00399    5   -29.973  …              0.919
3   1922-02-22         0.657  0.32500   10   -14.319  …              0.856
4   1922-02-22         0.644  0.68400    7    -8.247  …              0.802

   liveness  valence    tempo  time_signature              artist_id  \
0    0.1510    0.127  104.851               3  45tIt06XoI0Iio4LBEVpls
1    0.4360    0.402   87.921               4  45tIt06XoI0Iio4LBEVpls
2    0.1050    0.396   79.895               4  45tIt06XoI0Iio4LBEVpls
3    0.0931    0.105   81.944               5  45tIt06XoI0Iio4LBEVpls
4    0.0847    0.138  100.031               4  45tIt06XoI0Iio4LBEVpls

   followers  genres artist_name  artist_popularity
0       91.0      []         Uli                  4
1       91.0      []         Uli                  4
2       91.0      []         Uli                  4
3       91.0      []         Uli                  4
4       91.0      []         Uli                  4

[5 rows x 23 columns]
```

[8]:
```
#change date column from objects to int objects for modeling
#tracks['release_date'] = pd.to_datetime(tracks['release_date'], format =␣
↪"%Y-%m-%d")
tracks['release_date'] = pd.to_datetime(tracks["release_date"]).dt.
↪strftime("%Y%m%d")
```

Cleaning up genres column

[9]:
```
tracks.set_index('track_id', inplace=True)
```

[10]:
```
#changing from string to list of strings
clean = [i.replace("[", '').replace("]", '') for i in tracks['genres']]
clean = [k[1:-1].split("', '") for k in clean]

tracks['clean_genres'] = clean
tracks.drop('genres', inplace = True, axis=1)
```

[11]:
```
#onehotencoding genres
from sklearn.preprocessing import MultiLabelBinarizer

mlb = MultiLabelBinarizer(sparse_output=True)

df = tracks.join(
            pd.DataFrame.sparse.from_spmatrix(
                mlb.fit_transform(tracks.pop('clean_genres')),
                index=tracks.index,
                columns=mlb.classes_))
```

[12]:
```python
#only keeping genres that appear more than 8000 times in the dataset
drops = []
for i in range(4521 - 21):
  if df.iloc[::, i+21].sum() < 8000:
    drops.append(df.iloc[::, i+21].name)

df.drop(drops, axis=1, inplace=True)
tracks = df
```

[13]:
```python
#replace all spaces in column names with underscores to make modeling easier
tracks.columns = tracks.columns.str.replace(' ', '_')
tracks.head()
```

[13]:
```
                         track_name  track_popularity  duration_ms  explicit  \
track_id
35iwgR4jXetI318WEWsa1Q      Carve                  6       126903         0
0PH9AACae1f957JAavhOl2   Lazy Boi                  0       157333         0
2SiNuAZ6jIU9xhClRKXcST     Sketch                  0        87040         0
4vV7uBcF2AnjNTOejBS5oL    L'enfer                  0        40000         0
598LlBn6jpEpVbLjmZPsYV   Graphite                  0       104400         0

                        release_date  danceability   energy  key  loudness  \
track_id
35iwgR4jXetI318WEWsa1Q      19220222         0.645  0.44500    0   -13.338
0PH9AACae1f957JAavhOl2      19220222         0.298  0.46000    1   -18.645
2SiNuAZ6jIU9xhClRKXcST      19220222         0.634  0.00399    5   -29.973
4vV7uBcF2AnjNTOejBS5oL      19220222         0.657  0.32500   10   -14.319
598LlBn6jpEpVbLjmZPsYV      19220222         0.644  0.68400    7    -8.247

                         mode  …  latin  latin_pop  lounge  mellow_gold  \
track_id                       …
35iwgR4jXetI318WEWsa1Q      1  …      0          0       0            0
0PH9AACae1f957JAavhOl2      1  …      0          0       0            0
2SiNuAZ6jIU9xhClRKXcST      0  …      0          0       0            0
4vV7uBcF2AnjNTOejBS5oL      0  …      0          0       0            0
598LlBn6jpEpVbLjmZPsYV      1  …      0          0       0            0

                         psychedelic_rock  rock  rock_en_espanol  soft_rock  \
track_id
35iwgR4jXetI318WEWsa1Q                  0     0                0          0
0PH9AACae1f957JAavhOl2                  0     0                0          0
2SiNuAZ6jIU9xhClRKXcST                  0     0                0          0
4vV7uBcF2AnjNTOejBS5oL                  0     0                0          0
598LlBn6jpEpVbLjmZPsYV                  0     0                0          0

                         soul  vocal_jazz
track_id
```

```
35iwgR4jXetI318WEWsa1Q     0          0
0PH9AACae1f957JAavh0l2     0          0
2SiNuAZ6jIU9xhClRKXcST     0          0
4vV7uBcF2AnjNT0ejBS5oL     0          0
598LlBn6jpEpVbLjmZPsYV     0          0

[5 rows x 46 columns]
```

3 Regression Modeling

Splitting into training and testing set. Same training/testing datasets will be used throughout the regression and classification modeling sections.

[14]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(tracks, test_size=0.3, random_state=88)

X_train = train.drop('track_popularity', axis=1)
y_train = train['track_popularity']
X_test = test.drop('track_popularity', axis=1)
y_test = test['track_popularity']
print(X_train.shape, X_test.shape)
```

```
(329026, 45) (141012, 45)
```

3.1 OLS

[15]:
```
#calculate VIFs
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm

def VIF(df, cols):
    values = sm.add_constant(df[cols]).values
    vif = [variance_inflation_factor(values, i) for i in range(len(cols)+1)]
    return pd.Series(vif[1:], index=cols)

VIF(train,␣
 ↪['duration_ms','explicit','danceability','energy','key','loudness','mode','acousticness',
         ␣
 ↪'instrumentalness','liveness','valence','tempo','time_signature','followers','artist_popula
```

[15]:
```
duration_ms         1.055264
explicit            1.051991
danceability        1.537024
energy              4.149880
key                 1.018456
loudness            2.502978
```

```
mode                  1.027044
acousticness          2.165865
instrumentalness      1.134507
liveness              1.078624
valence               1.705608
tempo                 1.100166
time_signature        1.051884
followers             1.228751
artist_popularity     1.303838
dtype: float64
```

No need to drop any features due to high VIFs since they are all below 5

[16]:
```python
import statsmodels.formula.api as smf

ols = smf.ols(formula='track_popularity ~ duration_ms + explicit + danceability␣
↪+ energy + key + loudness + mode + acousticness + instrumentalness +␣
↪liveness + valence + tempo + time_signature + followers + artist_popularity',
                 data=train).fit()

print(ols.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       track_popularity   R-squared:                       0.423
Model:                            OLS   Adj. R-squared:                  0.423
Method:                 Least Squares   F-statistic:                 1.610e+04
Date:                Fri, 05 May 2023   Prob (F-statistic):               0.00
Time:                        14:13:57   Log-Likelihood:            -1.3180e+06
No. Observations:              329026   AIC:                         2.636e+06
Df Residuals:                  329010   BIC:                         2.636e+06
Df Model:                          15
Covariance Type:            nonrobust
================================================================================
=====
                        coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----
Intercept             8.9785      0.319     28.130      0.000       8.353
9.604
duration_ms        6.127e-06   2.06e-07     29.723      0.000    5.72e-06
6.53e-06
explicit              6.6870      0.129     51.799      0.000       6.434
6.940
danceability         12.9013      0.180     71.481      0.000      12.548
13.255
energy               -0.7028      0.193     -3.635      0.000      -1.082
```

```
-0.324
key                    0.0252      0.007      3.800      0.000       0.012
0.038
loudness               0.5317      0.008     67.067      0.000       0.516
0.547
mode                  -0.1392      0.050     -2.799      0.005      -0.237
-0.042
acousticness          -7.7630      0.101    -77.229      0.000      -7.960
-7.566
instrumentalness      -3.9817      0.102    -39.030      0.000      -4.182
-3.782
liveness              -6.4951      0.130    -50.113      0.000      -6.749
-6.241
valence               -8.6830      0.120    -72.491      0.000      -8.918
-8.448
tempo                  0.0167      0.001     20.401      0.000       0.015
0.018
time_signature         0.3926      0.052      7.524      0.000       0.290
0.495
followers           -1.28e-07   6.68e-09    -19.153      0.000   -1.41e-07
-1.15e-07
artist_popularity      0.4513      0.001    335.872      0.000       0.449
0.454
==============================================================================
Omnibus:                      276.956   Durbin-Watson:                   2.009
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              308.399
Skew:                           0.031   Prob(JB):                     1.08e-67
Kurtosis:                       3.137   Cond. No.                     5.98e+07
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.98e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

All of the p-values are now extremeley small, so we will not be removing any more features.

[ ]:
```python
# compute out-of-sample R-squared using the test set
def OSR2(model, df_train, df_test, dependent_var):
    y_test = df_test[dependent_var]
    y_pred = model.predict(df_test)
    SSE = np.sum((y_test - y_pred)**2)
    SST = np.sum((y_test - np.mean(df_train[dependent_var]))**2)
    return 1 - SSE/SST
```

[ ]:
```python
# compute test set RSS
def test_rss(model, df_train, df_test, dependent_var):
```

```
    y_test = df_test[dependent_var]
    y_pred = model.predict(df_test)
    return np.sum((y_test - y_pred)**2)
```

[ ]:
```
ols_osr2 = OSR2(ols, train, test, 'track_popularity')
ols_test_rss = test_rss(ols, train, test, 'track_popularity')
print('Out-of-sample R-squared for OLS: ', ols_osr2)
print('Test RSS for OLS: ', ols_test_rss)
```

```
Out-of-sample R-squared for OLS:  0.418832640795064
Test RSS for OLS:  25052477.51463911
```

3.2 CART - Regression Tree

[ ]:
```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import KFold
```

[ ]:
```
dtr_X_train = X_train.drop(['track_name', 'artist_name', 'artist_id'], axis=1)
dtr_X_test = X_test.drop(['track_name', 'artist_name', 'artist_id'], axis=1)

dtr = DecisionTreeRegressor(min_samples_split=10,
                            ccp_alpha=0.02,
                            random_state = 88)
dtr.fit(dtr_X_train, y_train)
```

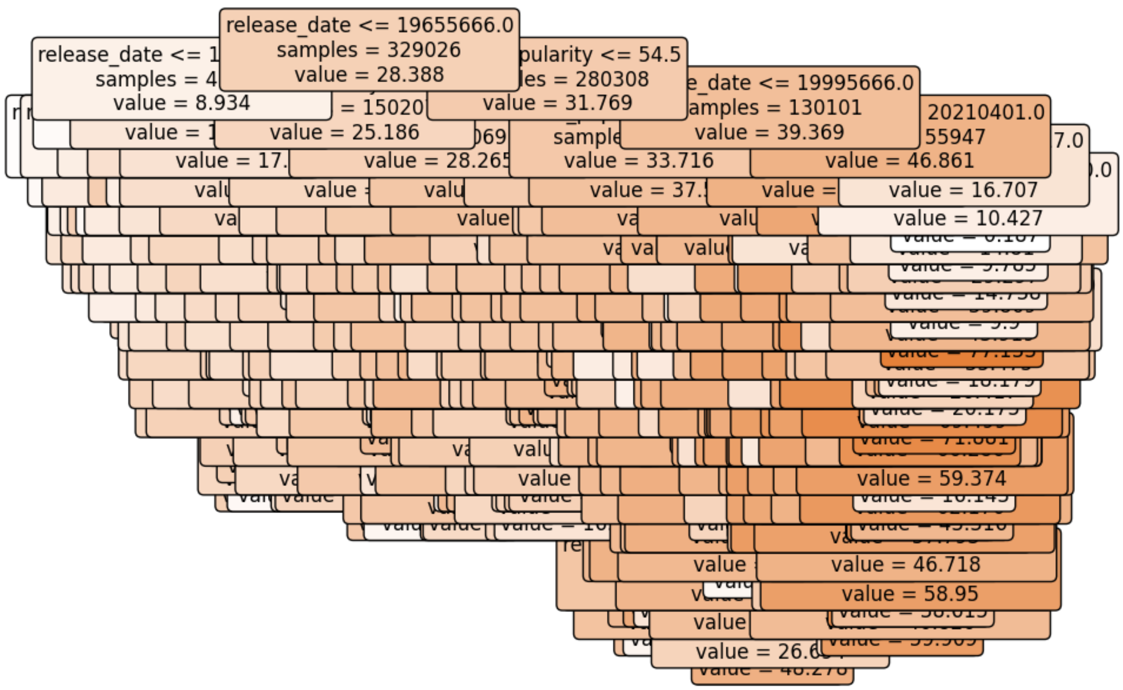
```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:768:
UserWarning: pandas.DataFrame with sparse columns found.It will be converted to
a dense numpy array.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py:185:
FutureWarning: The behavior of .astype from SparseDtype to a non-sparse dtype is
deprecated. In a future version, this will return a non-sparse array with the
requested dtype. To retain the old behavior, use
`obj.astype(SparseDtype(dtype))`
  array = numpy.asarray(array, order=order, dtype=dtype)
```

[ ]:
```
DecisionTreeRegressor(ccp_alpha=0.02, min_samples_split=10, random_state=88)
```

[ ]:
```
print('Node count =', dtr.tree_.node_count)
plt.figure(figsize=(9,7))
plot_tree(dtr,
          feature_names=dtr_X_train.columns,
          class_names=['0','1'],
          filled=True,
```

```
            impurity=False,
            rounded=True,
            fontsize=12)
plt.show()
```

```
Node count = 843
```



[ ]:

```
def OSR2(model, X_test, y_test, y_train):
    y_pred = model.predict(X_test)
    SSE = np.sum((y_test - y_pred)**2)
    SST = np.sum((y_test - np.mean(y_train))**2)
    return (1 - SSE/SST)

def test_RSS(model, X_test, y_test, y_train):
    y_pred = model.predict(X_test)
    return np.sum((y_test - y_pred)**2)
```

[ ]:

```
dtr_osr2 = OSR2(dtr, dtr_X_test, y_test, y_train)
dtr_test_rss = test_RSS(dtr, dtr_X_test, y_test, y_train)
print('Out-of-sample R-squared for Regression Tree:', dtr_osr2)
print('Test RSS for Regression Tree: ', dtr_test_rss)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:768:
UserWarning: pandas.DataFrame with sparse columns found.It will be converted to
a dense numpy array.
```

```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py:185:
FutureWarning: The behavior of .astype from SparseDtype to a non-sparse dtype is
deprecated. In a future version, this will return a non-sparse array with the
requested dtype. To retain the old behavior, use
`obj.astype(SparseDtype(dtype))`
  array = numpy.asarray(array, order=order, dtype=dtype)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:768:
UserWarning: pandas.DataFrame with sparse columns found.It will be converted to
a dense numpy array.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py:185:
FutureWarning: The behavior of .astype from SparseDtype to a non-sparse dtype is
deprecated. In a future version, this will return a non-sparse array with the
requested dtype. To retain the old behavior, use
`obj.astype(SparseDtype(dtype))`
  array = numpy.asarray(array, order=order, dtype=dtype)
```

```
Out-of-sample R-squared for Regression Tree: 0.6334125297223259
Test RSS for Regression Tree:  15802546.73773128
```

3.3 Comparison of Regression Models (OLS vs. Regression Tree)

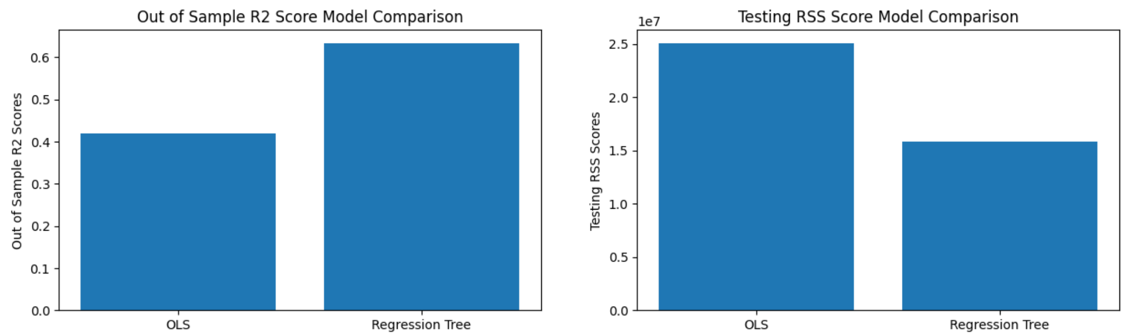
[ ]:
```
import matplotlib.pyplot as plt

models = ['OLS', 'Regression Tree']
osr2s = [ols_osr2, dtr_osr2]
test_RSSs = [ols_test_rss, dtr_test_rss]

plt.figure(figsize=(15,4))

plt.subplot(1,2,1)
plt.bar(models, osr2s)
plt.ylabel('Out of Sample R2 Scores')
plt.title('Out of Sample R2 Score Model Comparison')

plt.subplot(1,2,2)
plt.bar(models, test_RSSs)
plt.ylabel('Testing RSS Scores')
plt.title('Testing RSS Score Model Comparison');
```



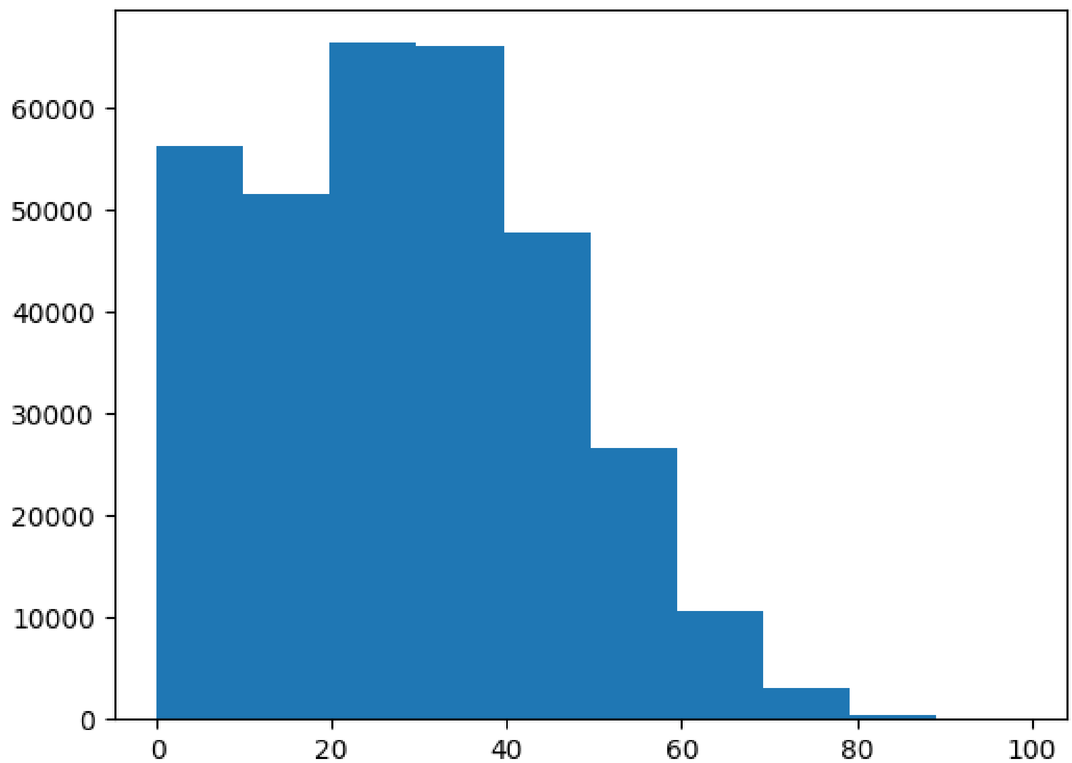
Comparing our two regression models, we can clearly see that the regression tree model performs much better compared to the OLS model. The regression tree model has a significantly higher OSR2 score at above 0.6 compared to the OLS model having an OSR2 score much lower around 0.4. Similarly, when looking at the testing RSS scores, the regression tree has a score of around 9000000 lower than that of the OLS model.

4 Classification Modeling

Considering tracks with a track popularity over 40 to be considered popular. Classification models to predict whether a track is popular or not.

4.1 Logistic Regression

[ ]:
```
plt.hist(train['track_popularity'])
#Probably will make a break along the lines of 40 track popularity for 1/0 for␣
↪logistic regression model
#since it's a 50/50 ish split
tracks['over40'] = (tracks['track_popularity'] > 40).astype(int)
```



[ ]:
```
train, test = train_test_split(tracks, test_size=0.3, random_state=88)
X_train = train.drop('track_popularity', axis=1)
y_train = train['track_popularity']
X_test = test.drop('track_popularity', axis=1)
y_test = test['track_popularity']
print(X_train.shape, X_test.shape)
```

```
(329026, 46) (141012, 46)
```

[ ]:
```
VIF(train,␣
↪['duration_ms','explicit','danceability','energy','key','loudness','mode','acousticness',
       ␣
↪'instrumentalness','liveness','valence','tempo','time_signature','followers','artist_popula
```

[ ]:
```
duration_ms         1.055264
explicit            1.051991
danceability        1.537024
energy              4.149880
key                 1.018456
loudness            2.502978
mode                1.027044
acousticness        2.165865
```

```
instrumentalness     1.134507
liveness             1.078624
valence              1.705608
tempo                1.100166
time_signature       1.051884
followers            1.228751
artist_popularity    1.303838
dtype: float64
```

[ ]:
```
VIF(train,␣
↪['duration_ms','explicit','danceability','key','loudness','mode','acousticness',
          ␣
↪'instrumentalness','liveness','valence','tempo','time_signature','followers','artist_popula
```

[ ]:
```
duration_ms          1.054485
explicit             1.051468
danceability         1.518525
key                  1.018353
loudness             1.421752
mode                 1.024924
acousticness         1.412732
instrumentalness     1.107882
liveness             1.028398
valence              1.515340
tempo                1.097532
time_signature       1.049389
followers            1.227288
artist_popularity    1.301911
dtype: float64
```

Removed energy since the VIF was close to 5. Now all VIFs are below 2. This seems reasonable to have for my model. We will cut down more columns if they are statistically not significant or otherwise don't help our model.

[ ]:
```
train.columns
```

[ ]:
```
Index(['track_name', 'track_popularity', 'duration_ms', 'explicit',
       'release_date', 'danceability', 'energy', 'key', 'loudness', 'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'time_signature', 'artist_id', 'followers',
       'artist_name', 'artist_popularity', '', 'adult_standards', 'album_rock',
       'art_rock', 'brill_building_pop', 'c-pop', 'classic_rock', 'cool_jazz',
       'country_rock', 'filmi', 'folk', 'folk_rock', 'hard_rock', 'hoerspiel',
       'jazz', 'latin', 'latin_pop', 'lounge', 'mellow_gold',
       'psychedelic_rock', 'rock', 'rock_en_espanol', 'soft_rock', 'soul',
       'vocal_jazz', 'over40'],
      dtype='object')
```

```
[ ]: logit = smf.logit(formula='over40 ~ duration_ms + explicit + danceability + key␣
     ↪+ loudness + mode + speechiness + acousticness + \
                              instrumentalness + liveness + valence + tempo +␣
     ↪time_signature + followers',
                       data=train).fit()

     print(logit.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.491185
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                 over40   No. Observations:               329026
Model:                          Logit   Df Residuals:                   329011
Method:                           MLE   Df Model:                           14
Date:                Fri, 05 May 2023   Pseudo R-squ.:                  0.1284
Time:                        17:35:22   Log-Likelihood:            -1.6161e+05
converged:                       True   LL-Null:                   -1.8542e+05
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
====
                       coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----
Intercept           -0.7284      0.059    -12.421      0.000      -0.843
-0.613
duration_ms       2.611e-07   4.09e-08      6.382      0.000    1.81e-07
3.41e-07
explicit             1.1424      0.022     52.572      0.000       1.100
1.185
danceability         2.2753      0.035     65.466      0.000       2.207
2.343
key                  0.0053      0.001      4.288      0.000       0.003
0.008
loudness             0.1028      0.001     77.821      0.000       0.100
0.105
mode                 0.0291      0.009      3.123      0.002       0.011
0.047
speechiness         -1.3848      0.041    -33.375      0.000      -1.466
-1.303
acousticness        -0.7134      0.016    -45.620      0.000      -0.744
-0.683
instrumentalness    -0.5842      0.023    -25.728      0.000      -0.629
-0.540
liveness            -0.5521      0.026    -21.246      0.000      -0.603
```

```
-0.501
valence              -1.5141      0.021    -72.054      0.000      -1.555
-1.473
tempo                 0.0024      0.000     14.758      0.000       0.002
0.003
time_signature        0.0252      0.012      2.103      0.036       0.002
0.049
followers          1.166e-07   1.68e-09     69.350      0.000    1.13e-07
1.2e-07
================================================================================
====
```

Adding genres into the model

```
[ ]: logit2 = smf.logit(formula='over40 ~ duration_ms + explicit + danceability +␣
     ↪key + loudness + mode + speechiness + acousticness \
                           + instrumentalness + liveness + valence + tempo +␣
     ↪time_signature + followers \
                           + adult_standards + album_rock + art_rock +␣
     ↪brill_building_pop + classic_rock + cool_jazz \
                           + country_rock + filmi + folk + folk_rock + hard_rock␣
     ↪+ hoerspiel + jazz + latin + latin_pop + lounge + mellow_gold \
                           + psychedelic_rock + rock + rock_en_espanol +␣
     ↪soft_rock + soul + vocal_jazz',
                  data=train).fit()

     print(logit2.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.484597
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                 over40   No. Observations:               329026
Model:                          Logit   Df Residuals:                   328988
Method:                           MLE   Df Model:                           37
Date:                Fri, 05 May 2023   Pseudo R-squ.:                  0.1401
Time:                        17:35:27   Log-Likelihood:            -1.5944e+05
converged:                       True   LL-Null:                   -1.8542e+05
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
======
                         coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------
Intercept             -0.7530      0.059    -12.661      0.000      -0.870
-0.636
```

duration_ms	2.571e-07	4.2e-08	6.119	0.000	1.75e-07	3.39e-07
explicit	1.1274	0.022	51.470	0.000	1.084	1.170
danceability	2.2922	0.036	63.956	0.000	2.222	2.362
key	0.0049	0.001	3.914	0.000	0.002	0.007
loudness	0.1047	0.001	76.990	0.000	0.102	0.107
mode	0.0054	0.009	0.572	0.567	-0.013	0.024
speechiness	-0.9385	0.047	-20.125	0.000	-1.030	-0.847
acousticness	-0.5714	0.016	-34.915	0.000	-0.604	-0.539
instrumentalness	-0.5180	0.023	-22.441	0.000	-0.563	-0.473
liveness	-0.5564	0.026	-21.291	0.000	-0.608	-0.505
valence	-1.4981	0.021	-70.175	0.000	-1.540	-1.456
tempo	0.0022	0.000	13.513	0.000	0.002	0.002
time_signature	0.0092	0.012	0.759	0.448	-0.015	0.033
followers	1.049e-07	1.74e-09	60.343	0.000	1.02e-07	1.08e-07
adult_standards	0.0386	0.036	1.085	0.278	-0.031	0.108
album_rock	0.1072	0.039	2.749	0.006	0.031	0.184
art_rock	-0.0484	0.032	-1.515	0.130	-0.111	0.014
brill_building_pop	-0.1925	0.043	-4.487	0.000	-0.277	-0.108
classic_rock	-0.9658	0.036	-26.524	0.000	-1.037	-0.894
cool_jazz	-0.5566	0.112	-4.976	0.000	-0.776	-0.337
country_rock	0.5084	0.034	14.885	0.000	0.441	0.575
filmi	-0.8134	0.046	-17.859	0.000	-0.903	-0.724
folk	0.0749	0.044	1.717	0.086	-0.011	0.160
folk_rock	-0.0675	0.041	-1.639	0.101	-0.148	0.013

```
hard_rock                -0.3215      0.037     -8.765      0.000      -0.393
-0.250
hoerspiel                -0.9276      0.072    -12.832      0.000      -1.069
-0.786
jazz                     -0.0918      0.106     -0.867      0.386      -0.299
0.116
latin                     0.1027      0.036      2.852      0.004       0.032
0.173
latin_pop                 0.8019      0.039     20.686      0.000       0.726
0.878
lounge                   -0.2878      0.052     -5.527      0.000      -0.390
-0.186
mellow_gold               0.0598      0.043      1.376      0.169      -0.025
0.145
psychedelic_rock         -0.1998      0.042     -4.786      0.000      -0.282
-0.118
rock                      0.8905      0.026     33.826      0.000       0.839
0.942
rock_en_espanol           0.4370      0.028     15.806      0.000       0.383
0.491
soft_rock                 0.4052      0.038     10.739      0.000       0.331
0.479
soul                      0.2393      0.031      7.636      0.000       0.178
0.301
vocal_jazz               -0.2184      0.053     -4.110      0.000      -0.323
-0.114
================================================================================
======
```

Remove mode, time_signature, adult_standards, art_rock, and jazz because they all have very high p-values.

[ ]:

```python
logit3 = smf.logit(formula='over40 ~ duration_ms + explicit + danceability +␣
↪key + loudness + speechiness + acousticness \
                         + instrumentalness + liveness + valence + tempo +␣
↪followers \
                         + album_rock + brill_building_pop + classic_rock +␣
↪cool_jazz \
                         + country_rock + filmi + folk + folk_rock + hard_rock␣
↪+ hoerspiel + latin + latin_pop + lounge + mellow_gold \
                         + psychedelic_rock + rock + rock_en_espanol +␣
↪soft_rock + soul + vocal_jazz',
                  data=train).fit()

print(logit3.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.484605
```

```
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                 over40   No. Observations:               329026
Model:                          Logit   Df Residuals:                   328993
Method:                           MLE   Df Model:                           32
Date:                Fri, 05 May 2023   Pseudo R-squ.:                  0.1401
Time:                        17:35:34   Log-Likelihood:            -1.5945e+05
converged:                       True   LL-Null:                   -1.8542e+05
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
======
                         coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------
Intercept             -0.7120      0.036    -19.793      0.000      -0.782
-0.641
duration_ms         2.534e-07    4.2e-08      6.038      0.000    1.71e-07
3.36e-07
explicit               1.1273      0.022     51.491      0.000       1.084
1.170
danceability           2.2923      0.036     64.394      0.000       2.223
2.362
key                    0.0048      0.001      3.874      0.000       0.002
0.007
loudness               0.1048      0.001     77.277      0.000       0.102
0.107
speechiness           -0.9417      0.047    -20.228      0.000      -1.033
-0.850
acousticness          -0.5712      0.016    -35.162      0.000      -0.603
-0.539
instrumentalness      -0.5196      0.023    -22.525      0.000      -0.565
-0.474
liveness              -0.5557      0.026    -21.270      0.000      -0.607
-0.505
valence               -1.4969      0.021    -70.176      0.000      -1.539
-1.455
tempo                  0.0022      0.000     13.497      0.000       0.002
0.002
followers           1.052e-07   1.73e-09     60.726      0.000    1.02e-07
1.09e-07
album_rock             0.0935      0.038      2.448      0.014       0.019
0.168
brill_building_pop    -0.1753      0.041     -4.316      0.000      -0.255
-0.096
classic_rock          -0.9680      0.036    -26.621      0.000      -1.039
-0.897
```

```
cool_jazz            -0.6416      0.062    -10.365      0.000      -0.763
-0.520
country_rock          0.5101      0.034     15.013      0.000       0.444
0.577
filmi                -0.8153      0.046    -17.906      0.000      -0.905
-0.726
folk                  0.0771      0.044      1.768      0.077      -0.008
0.163
folk_rock            -0.0711      0.041     -1.728      0.084      -0.152
0.010
hard_rock            -0.3126      0.036     -8.604      0.000      -0.384
-0.241
hoerspiel            -0.9293      0.072    -12.861      0.000      -1.071
-0.788
latin                 0.1022      0.036      2.836      0.005       0.032
0.173
latin_pop             0.8018      0.039     20.684      0.000       0.726
0.878
lounge               -0.2735      0.049     -5.629      0.000      -0.369
-0.178
mellow_gold           0.0661      0.043      1.530      0.126      -0.019
0.151
psychedelic_rock     -0.2144      0.041     -5.247      0.000      -0.294
-0.134
rock                  0.8824      0.026     34.048      0.000       0.832
0.933
rock_en_espanol       0.4372      0.028     15.812      0.000       0.383
0.491
soft_rock             0.4077      0.037     10.886      0.000       0.334
0.481
soul                  0.2418      0.031      7.817      0.000       0.181
0.302
vocal_jazz           -0.2066      0.049     -4.214      0.000      -0.303
-0.111
================================================================================
======
```

Now remove folk, folk_rock, and mellow_gold as they are the remaining features with high p-values.

```
[ ]: logit4 = smf.logit(formula='over40 ~ duration_ms + explicit + danceability +␣
     ↪key + loudness + speechiness + acousticness \
                           + instrumentalness + liveness + valence + tempo +␣
     ↪followers \
                           + album_rock + brill_building_pop + classic_rock +␣
     ↪cool_jazz \
                           + country_rock + filmi + hard_rock + hoerspiel +␣
     ↪latin + latin_pop + lounge \
```

```
                              + psychedelic_rock + rock + rock_en_espanol +␣
↪soft_rock + soul + vocal_jazz',
                  data=train).fit()

print(logit4.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.484615
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                 over40   No. Observations:               329026
Model:                          Logit   Df Residuals:                   328996
Method:                           MLE   Df Model:                           29
Date:                Fri, 05 May 2023   Pseudo R-squ.:                  0.1401
Time:                        17:35:38   Log-Likelihood:            -1.5945e+05
converged:                       True   LL-Null:                   -1.8542e+05
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
======
                        coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------
Intercept            -0.7127      0.036    -19.815      0.000      -0.783
-0.642
duration_ms        2.538e-07    4.2e-08      6.048      0.000    1.72e-07
3.36e-07
explicit              1.1273      0.022     51.493      0.000       1.084
1.170
danceability          2.2918      0.036     64.382      0.000       2.222
2.362
key                   0.0048      0.001      3.874      0.000       0.002
0.007
loudness              0.1046      0.001     77.254      0.000       0.102
0.107
speechiness          -0.9420      0.047    -20.237      0.000      -1.033
-0.851
acousticness         -0.5703      0.016    -35.142      0.000      -0.602
-0.538
instrumentalness     -0.5205      0.023    -22.575      0.000      -0.566
-0.475
liveness             -0.5558      0.026    -21.271      0.000      -0.607
-0.505
valence              -1.4974      0.021    -70.204      0.000      -1.539
-1.456
tempo                 0.0022      0.000     13.511      0.000       0.002
```

```
0.002
followers           1.054e-07   1.73e-09     61.031      0.000    1.02e-07
1.09e-07
album_rock             0.1010      0.038      2.689      0.007       0.027
0.175
brill_building_pop    -0.1779      0.039     -4.560      0.000      -0.254
-0.101
classic_rock          -0.9690      0.036    -27.224      0.000      -1.039
-0.899
cool_jazz             -0.6435      0.062    -10.397      0.000      -0.765
-0.522
country_rock           0.5182      0.031     16.650      0.000       0.457
0.579
filmi                 -0.8161      0.046    -17.925      0.000      -0.905
-0.727
hard_rock             -0.3237      0.036     -9.014      0.000      -0.394
-0.253
hoerspiel             -0.9298      0.072    -12.868      0.000      -1.071
-0.788
latin                  0.1019      0.036      2.828      0.005       0.031
0.173
latin_pop              0.8010      0.039     20.665      0.000       0.725
0.877
lounge                -0.2744      0.049     -5.654      0.000      -0.369
-0.179
psychedelic_rock      -0.2341      0.040     -5.893      0.000      -0.312
-0.156
rock                   0.8865      0.026     34.366      0.000       0.836
0.937
rock_en_espanol        0.4372      0.028     15.813      0.000       0.383
0.491
soft_rock              0.4483      0.026     17.238      0.000       0.397
0.499
soul                   0.2432      0.031      7.872      0.000       0.183
0.304
vocal_jazz            -0.2040      0.049     -4.162      0.000      -0.300
-0.108
================================================================================
======
```

[ ]:
```
import numpy as np
np.exp(2.2918) - 1
```

[ ]:
```
8.892728576001097
```

We can see that the odds of a track being popular goes up by almost 900% when the danceability score goes up by 1.

Now all of our p-values are low, indicating that all of our features are siginifcant and thus we will keep all the remaining features and evaluate our 4th logistic regression model on the testing data.

[ ]:
```
from sklearn.metrics import confusion_matrix
```

To remind you of what each element of the confusion matrix represents:

TN FP FN TP

[ ]:
```
y_test = test['over40']
y_prob = logit4.predict(X_test)
y_pred = pd.Series([1 if x > 0.35 else 0 for x in y_prob], index=y_prob.index)
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix : \n", cm)
def get_accuracy(cm):
    return (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
def get_tpr(cm):
    return (cm.ravel()[3]/(cm.ravel()[3] + cm.ravel()[2]))
def get_fpr(cm):
    return (cm.ravel()[1]/(cm.ravel()[1] + cm.ravel()[0]))

logit_accuracy = get_accuracy(cm)
logit_tpr = get_tpr(cm)
logit_fpr = get_fpr(cm)

print('Accuracy: ' + str(logit_accuracy))
print('TPR: ' + str(logit_tpr))
print('FPR: ' + str(logit_fpr))
```

```
Confusion Matrix :
 [[90799 14841]
 [19088 16284]]
Accuracy: 0.7593892718350211
TPR: 0.46036412982019675
FPR: 0.1404865581219235
```

4.2 CART - Classification Tree

[ ]:
```
dtc_X_train = X_train.drop(['track_name', 'artist_name', 'artist_id',␣
↪'over40'], axis=1)
dtc_X_test = X_test.drop(['track_name', 'artist_name', 'artist_id', 'over40'],␣
↪axis=1)
```

[ ]:
```
dtc = DecisionTreeClassifier(min_samples_split=20,
                             ccp_alpha=0.00,
                             random_state = 88)
dtc.fit(dtc_X_train, y_train)
```

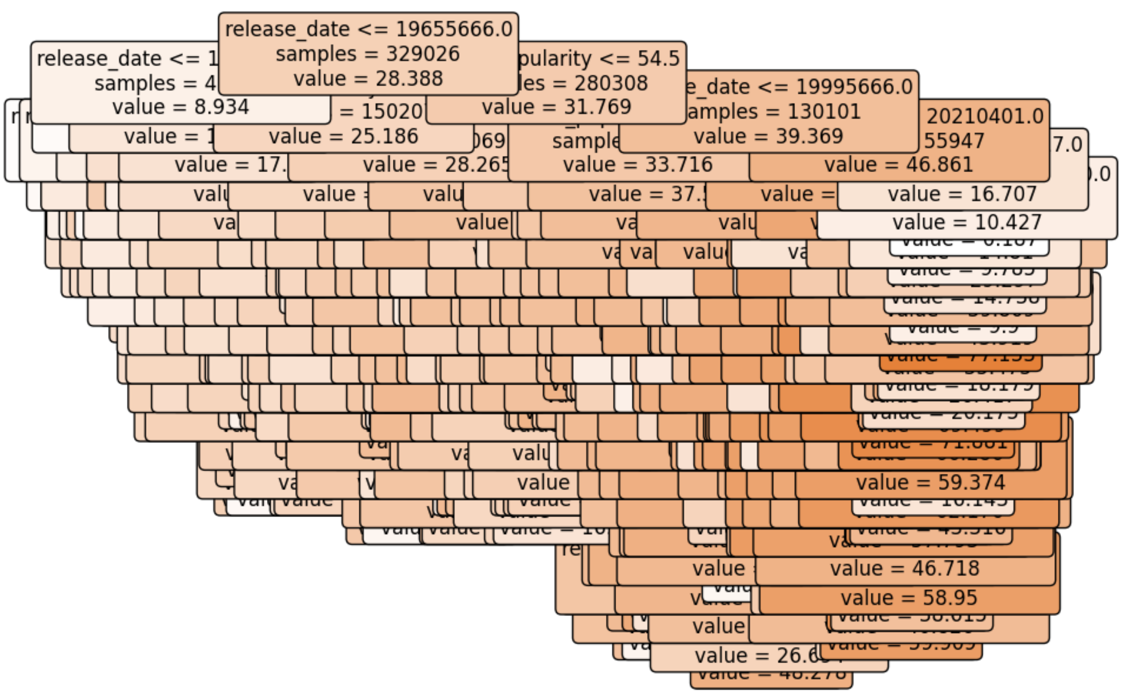
```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:768:
```

```
UserWarning: pandas.DataFrame with sparse columns found.It will be converted to
a dense numpy array.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py:185:
FutureWarning: The behavior of .astype from SparseDtype to a non-sparse dtype is
deprecated. In a future version, this will return a non-sparse array with the
requested dtype. To retain the old behavior, use
`obj.astype(SparseDtype(dtype))`
  array = numpy.asarray(array, order=order, dtype=dtype)
```

[ ]: DecisionTreeClassifier(min_samples_split=20, random_state=88)

[ ]:
```python
print('Node count =', dtr.tree_.node_count)
plt.figure(figsize=(9,7))
plot_tree(dtr,
          feature_names=dtr_X_train.columns,
          class_names=['0','1'],
          filled=True,
          impurity=False,
          rounded=True,
          fontsize=12)
plt.show()
```

```
Node count = 843
```



[ ]:
```
y_pred_ct = dtc.predict(dtc_X_test)
y_pred_ct_thresh = [1 if x >= 38 else 0 for x in y_pred_ct]
cm_ct = confusion_matrix(y_test, y_pred_ct_thresh)
ct_accuracy = get_accuracy(cm_ct)
ct_tpr = get_tpr(cm_ct)
ct_fpr = get_fpr(cm_ct)

print('Accuracy: ', ct_accuracy)
print('TPR: ', ct_tpr)
print('FPR: ', ct_fpr)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:768:
UserWarning: pandas.DataFrame with sparse columns found.It will be converted to
a dense numpy array.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py:185:
FutureWarning: The behavior of .astype from SparseDtype to a non-sparse dtype is
deprecated. In a future version, this will return a non-sparse array with the
requested dtype. To retain the old behavior, use
`obj.astype(SparseDtype(dtype))`
  array = numpy.asarray(array, order=order, dtype=dtype)
```

```
Accuracy:  0.8091935438118741
TPR:  0.6143559877869501
FPR:  0.12556796667928816
```

4.3 Comparison of Classification Models (Baseline vs. Logistic vs. Classification)

[ ]:
```
# Baseline model
# most frequent outcome for over40 is 0 ==> baseline model predicts everything␣
↪as 0
below_40 = np.sum(test['over40'] == 0)
above_40 = np.sum(test['over40'] == 1)

baseline_accuracy = below_40 / (below_40 + above_40)
baseline_tpr = 0 / (0 + above_40) # TPR = TP/P = TP/(TP+FN)
baseline_fpr = 0 / (0 + below_40) # FPR = FP/N = FP/(FP+TN)

# create dataframe comparing
df = pd.DataFrame(np.array([[baseline_accuracy, logit_accuracy, ct_accuracy],
                            [baseline_tpr, logit_tpr,  ct_tpr],
                            [baseline_fpr, logit_fpr, ct_fpr]]),
                  columns=['Baseline', 'Logistic Reg', 'Classification Tree'],
                 index=['Accuracy', 'TPR', 'FPR'])
df = df.round(decimals = 3)
df
```

[ ]:

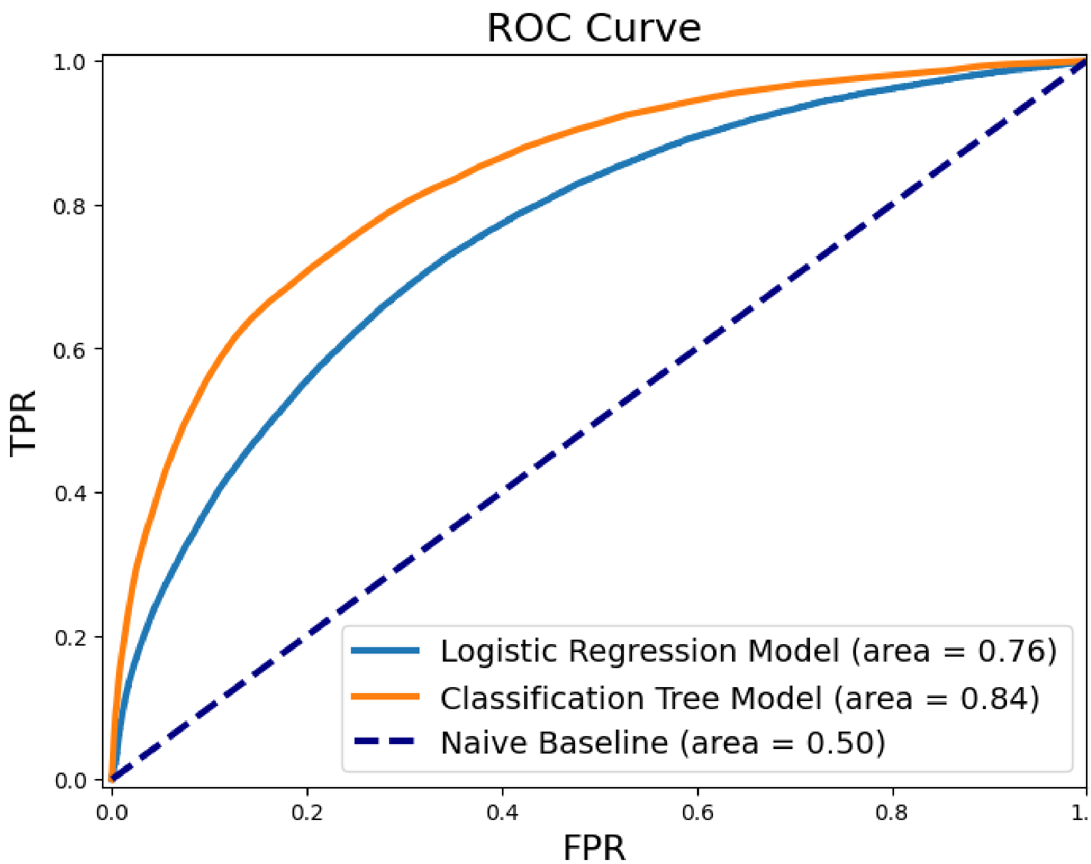
	Baseline	Logistic Reg	Classification Tree
Accuracy	0.749	0.759	0.809
TPR	0.000	0.460	0.614
FPR	0.000	0.140	0.126

[ ]:

```
# ROC/AUC Curve

from sklearn.metrics import roc_curve, auc
fpr, tpr, _mod = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
fpr_ct, tpr_ct, _mod = roc_curve(y_test, y_pred_ct)
roc_auc_ct = auc(fpr_ct, tpr_ct)

plt.figure(figsize=(8, 6))
plt.title('ROC Curve', fontsize=18)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr, tpr, lw=3, label='Logistic Regression Model (area = {:0.2f})'.
 ↪format(roc_auc))
plt.plot(fpr_ct, tpr_ct, lw=3, label='Classification Tree Model (area = {:0.
 ↪2f})'.format(roc_auc_ct))
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--', label='Naive␣
 ↪Baseline (area = 0.50)')
plt.legend(loc='lower right', fontsize=14)
plt.show()
```



When comparing our three classification models, we can clearly see that the classification tree model is the best performing model. While all three models have similar accuracies, the classification tree model has the highest accuracy at 0.809. While the accuracies were similar for all three models, the true positive rate of the classification tree model was significantly higher than the baseline and the logistic regression model at 0.614. Finally, the decision tree model has the lowest false positive rate at 0.126. Next when looking at the ROC curve, we can observe that the classification tree model has the largest area under the curve at 0.84 compared to the logistic regression model at an AUC of 0.76. Therefore since the classification tree model performed the best in every evaluation metric, we are confident that this model is the best peforming model in classifying whether a track is popular or not.