Mini-Project-Classify-Song-Genres-from-Audio-Data

Preparing our dataset

In [ ]:

```
import os
```

In [8]:

```
os.getcwd()
```

Out[8]:

```
'C:\\Users\\Admin'
```

In [9]:

```
os.chdir('C:\\Users\\Admin\\Downloads')
```

In [10]:

```
os.getcwd()
```

Out[10]:

```
'C:\\Users\\Admin\\Downloads'
```

In [13]:

```
df=pd.read_csv('fma-rock-vs-hiphop.csv')
```

In [14]:

```
df
```

Out[14]:

	track_id	bit_rate	comments	composer	date_created	date_recorded	duration	favorites	genre_top	genres	...	infor
0	135	256000	1	NaN	26-11-2008 01:43	26-11-2008 00:00	837	0	Rock	[45, 58]	...	
1	136	256000	1	NaN	26-11-2008 01:43	26-11-2008 00:00	509	0	Rock	[45, 58]	...	
2	151	192000	0	NaN	26-11-2008 01:44	NaN	192	0	Rock	[25]	...	
3	152	192000	0	NaN	26-11-2008 01:44	NaN	193	0	Rock	[25]	...	
4	153	256000	0	Arc and Sender	26-11-2008 01:45	26-11-2008 00:00	405	5	Rock	[26]	...	
...	...	...	...	...	...	...	...	...	...	...	...	
[illegible]	[illegible]	[illegible]	0	NaN	24-03-2017	NaN	[illegible]	[illegible]	Hip-Hop	[21,		

	track_id	bit_rate	comments	composer	date_created	date_recorded	duration	favorites	genre_top	genres	...	infor
17729	155063	320000	0	NaN	19:40	NaN	283	3	Hip-Hop	811]	...	
17730	155064	320000	0	NaN	24-03-2017 19:40	NaN	250	2	Hip-Hop	[21, 811]	...	
17731	155065	320000	0	NaN	24-03-2017 19:40	NaN	219	3	Hip-Hop	[21, 811]	...	
17732	155066	320000	0	NaN	24-03-2017 19:40	NaN	252	6	Hip-Hop	[21, 811]	...	
17733	155247	320000	0	Fleslit	29-03-2017 01:40	NaN	211	3	Hip-Hop	[21, 539, 811]	...	

17734 rows × 21 columns

In [15]:

```
df.head()
```

Out[15]:

	track_id	bit_rate	comments	composer	date_created	date_recorded	duration	favorites	genre_top	genres	...	informati
0	135	256000	1	NaN	26-11-2008 01:43	26-11-2008 00:00	837	0	Rock	[45, 58]	...	Na
1	136	256000	1	NaN	26-11-2008 01:43	26-11-2008 00:00	509	0	Rock	[45, 58]	...	Na
2	151	192000	0	NaN	26-11-2008 01:44	NaN	192	0	Rock	[25]	...	Na
3	152	192000	0	NaN	26-11-2008 01:44	NaN	193	0	Rock	[25]	...	Na
4	153	256000	0	Arc and Sender	26-11-2008 01:45	26-11-2008 00:00	405	5	Rock	[26]	...	Na

5 rows × 21 columns

In [17]:

```
df=pd.read_csv('echonest-metrics.json')
```

In [19]:

```
os.getcwd()
```

Out[19]:

```
'C:\\Users\\Admin\\Downloads'
```

In [28]:

```
os.chdir('C:\\Users\\Admin\\Downloads')
```

In [23]:

```
os.getcwd()
```

Out[23]:

'C:\\Users\\Admin\\Downloads'

In [32]:

```
df=pd.read_json('echonest-metrics.json')
```

In [33]:

```
df
```

Out[33]:

	track_id	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	valence
0	2	0.416675	0.675894	0.634476	0.010628	0.177647	0.159310	165.922	0.576661
1	3	0.374408	0.528643	0.817461	0.001851	0.105880	0.461818	126.957	0.269240
2	5	0.043567	0.745566	0.701470	0.000697	0.373143	0.124595	100.260	0.621661
3	10	0.951670	0.658179	0.924525	0.965427	0.115474	0.032985	111.562	0.963590
4	134	0.452217	0.513238	0.560410	0.019443	0.096567	0.525519	114.290	0.894072
...	...	...	...	...	...	...	...	...	...
13124	124857	0.007592	0.790364	0.719288	0.853114	0.720715	0.082550	141.332	0.890461
13125	124862	0.041498	0.843077	0.536496	0.865151	0.547949	0.074001	101.975	0.476845
13126	124863	0.000124	0.609686	0.895136	0.846624	0.632903	0.051517	129.996	0.496667
13127	124864	0.327576	0.574426	0.548327	0.452867	0.075928	0.033388	142.009	0.569274
13128	124911	0.993606	0.499339	0.050622	0.945677	0.095965	0.065189	119.965	0.204652

13129 rows × 9 columns

In [34]:

```
df.head()
```

Out[34]:

	track_id	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	valence
0	2	0.416675	0.675894	0.634476	0.010628	0.177647	0.159310	165.922	0.576661
1	3	0.374408	0.528643	0.817461	0.001851	0.105880	0.461818	126.957	0.269240
2	5	0.043567	0.745566	0.701470	0.000697	0.373143	0.124595	100.260	0.621661
3	10	0.951670	0.658179	0.924525	0.965427	0.115474	0.032985	111.562	0.963590
4	134	0.452217	0.513238	0.560410	0.019443	0.096567	0.525519	114.290	0.894072

In [36]:

```
tracks = pd.read_csv('fma-rock-vs-hiphop.csv')
echonest_metrics = pd.read_json('echonest-metrics.json',precise_float=True)
```

In [37]:

```
df
```

Out[37]:

	track_id	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	valence

	track_id	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	valence
0	2	0.416675	0.675894	0.634476	0.010628	0.177647	0.159310	165.922	0.576661
1	3	0.374408	0.528643	0.817461	0.001851	0.105880	0.461818	126.957	0.269240
2	5	0.043567	0.745566	0.701470	0.000697	0.373143	0.124595	100.260	0.621661
3	10	0.951670	0.658179	0.924525	0.965427	0.115474	0.032985	111.562	0.963590
4	134	0.452217	0.513238	0.560410	0.019443	0.096567	0.525519	114.290	0.894072
...	...	...	...	...	...	...	...	...	...
13124	124857	0.007592	0.790364	0.719288	0.853114	0.720715	0.082550	141.332	0.890461
13125	124862	0.041498	0.843077	0.536496	0.865151	0.547949	0.074001	101.975	0.476845
13126	124863	0.000124	0.609686	0.895136	0.846624	0.632903	0.051517	129.996	0.496667
13127	124864	0.327576	0.574426	0.548327	0.452867	0.075928	0.033388	142.009	0.569274
13128	124911	0.993606	0.499339	0.050622	0.945677	0.095965	0.065189	119.965	0.204652

13129 rows × 9 columns

In [40]:

```
echonest_metrics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13129 entries, 0 to 13128
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   track_id          13129 non-null  int64  
 1   acousticness      13129 non-null  float64
 2   danceability      13129 non-null  float64
 3   energy            13129 non-null  float64
 4   instrumentalness  13129 non-null  float64
 5   liveness          13129 non-null  float64
 6   speechiness       13129 non-null  float64
 7   tempo             13129 non-null  float64
 8   valence           13129 non-null  float64
dtypes: float64(8), int64(1)
memory usage: 1.0 MB
```

Pairwise relationships between continuous variables

In [41]:

```
corr_metrics = echonest_metrics.corr()
corr_metrics.style.background_gradient()
```

Out[41]:

	track_id	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	valence
track_id	1.000000	-0.279829	0.102056	0.121991	-0.283206	-0.004059	-0.075077	0.004313	0.020201
acousticness	-0.279829	1.000000	-0.189599	-0.477273	0.110033	0.041319	0.038785	-0.110701	-0.085436
danceability	0.102056	-0.189599	1.000000	0.045345	-0.118033	-0.143339	0.171311	-0.094352	0.428515
energy	0.121991	-0.477273	0.045345	1.000000	-0.002412	0.045752	-0.008645	0.227324	0.219384
instrumentalness	-0.283206	0.110033	-0.118033	-0.002412	1.000000	-0.058593	-0.216689	0.023003	-0.145200
liveness	-0.004059	0.041319	-0.143339	0.045752	-0.058593	1.000000	0.073104	-0.007566	-0.017886
speechiness	-0.075077	0.038785	0.171311	-0.008645	-0.216689	0.073104	1.000000	0.032188	0.094794

	track_id	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	valence
tempo	0.004313	-0.110701	-0.094352	0.227324	-0.023003	-0.007566	0.032188	1.000000	0.129911
valence	0.020201	-0.085436	0.428515	0.219384	-0.145200	-0.017886	0.094794	0.129911	1.000000

In [104]:

```
# Merge the relevant columns of tracks and echonest_metrics
echo_tracks = pd.merge(echonest_metrics, tracks[['track_id', 'genre_top']], on='track_id
')

# Inspect the resultant dataframe
print(echo_tracks.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4802 entries, 0 to 4801
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   track_id          4802 non-null   int64  
 1   acousticness      4802 non-null   float64
 2   danceability      4802 non-null   float64
 3   energy            4802 non-null   float64
 4   instrumentalness  4802 non-null   float64
 5   liveness          4802 non-null   float64
 6   speechiness       4802 non-null   float64
 7   tempo             4802 non-null   float64
 8   valence           4802 non-null   float64
 9   genre_top         4802 non-null   object 
dtypes: float64(8), int64(1), object(1)
memory usage: 412.7+ KB
None
```

Normalizing the feature data

In [109]:

```
# Define the features and labels
features = echo_tracks.drop(['genre_top', 'track_id'], axis = 1)

# Define our labels
labels = echo_tracks['genre_top']

# Import the StandardScaler
from sklearn.preprocessing import StandardScaler

# Scale the features and set the values to a new variable
scaler = StandardScaler()
scaled_train_features = scaler.fit_transform(features)
```

Principal Component Analysis on our scaled data

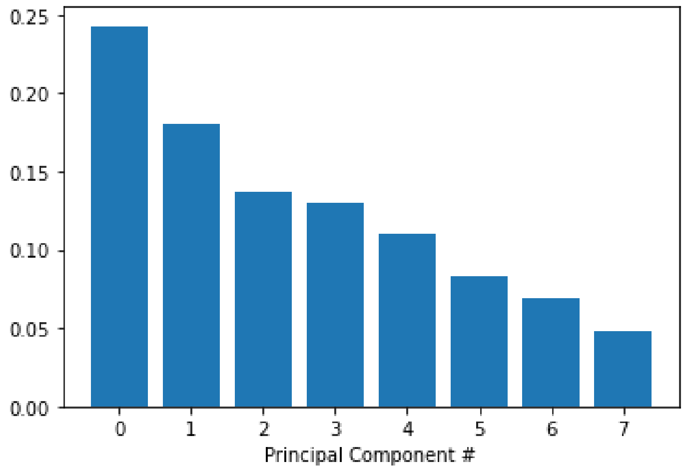
In [110]:

```
# This is just to make plots appear in the notebook
%matplotlib inline

# Import our plotting module, and PCA class
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Get our explained variance ratios from PCA using all features
pca = PCA()
pca.fit(scaled_train_features)
exp_variance = pca.explained_variance_ratio_
```

```
# plot the explained variance using a barplot
fig, ax = plt.subplots()
ax.bar(range(pca.n_components_), exp_variance)
ax.set_xlabel('Principal Component #')
```

Out[110]:

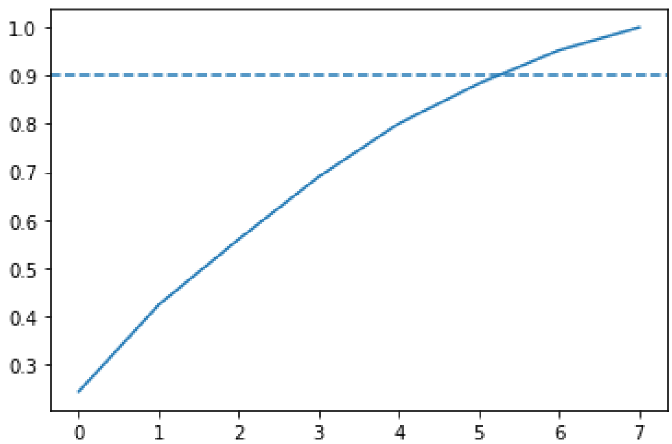
```
Text(0.5, 0, 'Principal Component #')
```



Further visualization of PCA

In [111]:

```
# Import numpy
import numpy as np

# Calculate the cumulative explained variance
cum_exp_variance = np.cumsum(exp_variance)

# Plot the cumulative explained variance and draw a dashed line at 0.90.
fig, ax = plt.subplots()
ax.plot(cum_exp_variance)
ax.axhline(y=0.9, linestyle='--')
n_components = 6

# Perform PCA with the chosen number of components and project data onto components
pca = PCA(n_components, random_state=10)
pca.fit(scaled_train_features)
pca_projection = pca.transform(scaled_train_features)
```



Train a decision tree to classify genre

In [112]:

```
# Import train_test_split function and Decision tree classifier
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeClassifier

# Split our data
train_features, test_features, train_labels, test_labels = train_test_split(pca_projectio
n,
                                                                            labels,
                                                                            random_stat
e=10)

# Train our decision tree
tree = DecisionTreeClassifier(random_state=10)
tree.fit(train_features, train_labels)

# Predict the labels for the test data
pred_labels_tree = tree.predict(test_features)
tree.score(test_features, test_labels)
```

Out[112]:

```
0.8434637801831807
```

Compare our decision tree to a logistic regression

In [114]:

```
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression

# Train our logistic regression and predict labels for the test set
logreg = LogisticRegression(random_state=10)
logreg.fit(train_features, train_labels)
pred_labels_logit = logreg.predict(test_features)

# Create the classification report for both models
from sklearn.metrics import classification_report
class_rep_tree = classification_report(test_labels, pred_labels_tree)
class_rep_log = classification_report(test_labels, pred_labels_logit)

print("Decision Tree: \n", class_rep_tree)
print("Logistic Regression: \n", class_rep_log)
```

```
Decision Tree:
               precision    recall  f1-score   support

     Hip-Hop       0.60      0.60      0.60       235
        Rock       0.90      0.90      0.90       966

    accuracy                           0.84      1201
   macro avg       0.75      0.75      0.75      1201
weighted avg       0.84      0.84      0.84      1201

Logistic Regression:
               precision    recall  f1-score   support

     Hip-Hop       0.77      0.54      0.64       235
        Rock       0.90      0.96      0.93       966

    accuracy                           0.88      1201
   macro avg       0.83      0.75      0.78      1201
weighted avg       0.87      0.88      0.87      1201
```

Balance our data for greater performance

In [115]:

```
# Subset only the hip-hop tracks, and then only the rock tracks
hop_only = echo_tracks[echo_tracks['genre_top']=='Hip-Hop']
```

```
rock_only = echo_tracks[echo_tracks['genre_top']=='Rock']

# sample the rocks songs to be the same number as there are hip-hop songs
rock_only = rock_only.sample(len(hop_only), random_state=10)

# concatenate the dataframes rock_only and hop_only
rock_hop_bal = pd.concat([rock_only, hop_only])

# The features, labels, and pca projection are created for the balanced dataframe
features = rock_hop_bal.drop(['genre_top', 'track_id'], axis=1)
labels = rock_hop_bal['genre_top']
pca_projection = pca.fit_transform(scaler.fit_transform(features))

# Redefine the train and test set with the pca_projection from the balanced data
train_features, test_features, train_labels, test_labels = train_test_split(pca_projectio
n, labels, random_state=10)
```

#To see if balancing our data improves model bias towards the "Rock" classification

In [116]:

```
# Train our decision tree on the balanced data
tree = DecisionTreeClassifier(random_state=10)
tree.fit(train_features, train_labels)
pred_labels_tree = tree.predict(test_features)

# Train our logistic regression on the balanced data
logreg = LogisticRegression(random_state=10)
logreg.fit(train_features, train_labels)
pred_labels_logit = logreg.predict(test_features)

# Compare the models
print("Decision Tree: \n", classification_report(test_labels, pred_labels_tree))
print("Logistic Regression: \n", classification_report(test_labels, pred_labels_logit))
```

```
Decision Tree: 
               precision    recall  f1-score   support

     Hip-Hop       0.74      0.73      0.74       230
        Rock       0.73      0.74      0.73       225

    accuracy                           0.74       455
   macro avg       0.74      0.74      0.74       455
weighted avg       0.74      0.74      0.74       455

Logistic Regression: 
               precision    recall  f1-score   support

     Hip-Hop       0.84      0.80      0.82       230
        Rock       0.80      0.85      0.83       225

    accuracy                           0.82       455
   macro avg       0.82      0.82      0.82       455
weighted avg       0.82      0.82      0.82       455
```

Using cross validation to evaluate our models

In [121]:

```
from sklearn.model_selection import KFold, cross_val_score

# Set up our K-fold cross-validation
kf = KFold(n_splits=10,random_state=10,shuffle=True)

tree = DecisionTreeClassifier(random_state=10)
```

```
logreg = LogisticRegression(random_state=10)

# Train our models using KFold cv
tree_score = cross_val_score(tree,pca_projection,labels,cv=kf)
logit_score = cross_val_score(logreg,pca_projection,labels,cv=kf)

# Print the mean of each array of scores
print("Decision Tree:", np.mean(tree_score),
      "Logistic Regression:", np.mean(logit_score))
```

```
Decision Tree: 0.7719780219780219 Logistic Regression: 0.823076923076923
```

 # Result : Our model will generalize 77% of the times on the future unseen data points.