```
#!/usr/bin/env python3

import numpy as np
import pandas as pd
import librosa
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn import metrics

import getopt, sys

argumentList = sys.argv[1:]
options = "scp"
long_options = ["Score", "ConfusionMatrix", "Predict"]

X_train = np.load('GTZAN Genre Classification\GTZAN Dataset\X_train.npy')
X_test = np.load('GTZAN Genre Classification\GTZAN Dataset\X_test.npy')
y_train = np.load('GTZAN Genre Classification\GTZAN Dataset\y_train.npy')
y_test = np.load('GTZAN Genre Classification\GTZAN Dataset\y_test.npy')

# KNN Classifier
clf = KNeighborsClassifier(n_neighbors=1)

# PCA for maximum accuracy with minimum components
pca = PCA(n_components=21)
pca.fit(X_train)

# Transform features
X_train_pc = pca.transform(X_train)
X_test_pc =  pca.transform(X_test)

# Fit the model
clf.fit(X_train_pc, y_train)

try:
    arguments, values = getopt.getopt(argumentList, options, long_options)

    for currentArgument, currentValue in arguments:
        if currentArgument in ("-s", "--Score"):
            # Accuracy
            score = clf.score(X_test_pc, y_test)
            print("Test Score: ", score)

        elif currentArgument in ("-c", "--ConfusionMatrix"):
            # Confusion Matrix
            y_test_pred = clf.predict(X_test_pc)
            expected = y_test
            predicted = clf.predict(X_test_pc)
            print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected,
predicted))

        elif currentArgument in ("-p", "--Predict"):

            # # # # # #
            # This is all redudant and should be eliminated in a future version
            data = pd.read_csv('GTZAN Genre Classification\GTZAN Dataset\data.csv')
            data = data.drop(['filename'], axis = 1)
            genre_list = data.iloc[:, -1]
            encoder = LabelEncoder()
            encoder.fit_transform(genre_list)
            scaler = StandardScaler()
            scaler.fit(np.array(data.iloc[:, :-1], dtype = float))
            # # # # #

            # # # # # #
            # More redudancy, condense into a single callable function
            y, sr = librosa.load(sys.argv[2], mono=True, duration=30)
            print(sys.argv[2])
            chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
            rmse = librosa.feature.rms(y=y)[0]
            spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
            spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
            rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
            zcr = librosa.feature.zero_crossing_rate(y)
            mfcc = librosa.feature.mfcc(y=y, sr=sr)
            to_append = np.array([np.mean(chroma_stft), np.mean(rmse),
np.mean(spec_cent), np.mean(spec_bw), np.mean(rolloff), np.mean(zcr)])
```

```
            for e in mfcc:
                to_append = np.append(to_append, np.mean(e))
            # # # # # # #

            y = pd.DataFrame(data=to_append)
            y = np.array(y.iloc[:, 0], dtype=float).reshape(1, -1)
            y = scaler.transform(y)
            y = pca.transform(y)
            predicted = clf.predict(y)
            print("Predicted genre:", encoder.inverse_transform(predicted)[0].upper(),
"\n\n")

except getopt.error as err:
    print(str(err))


```