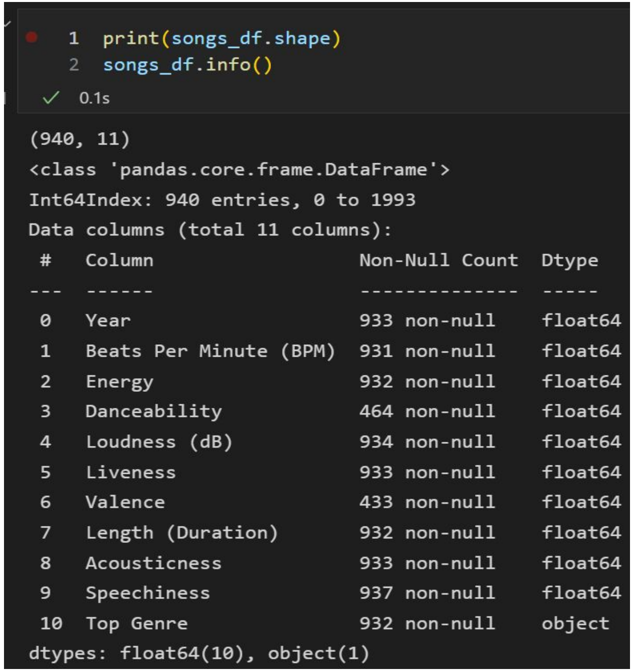
# XGBoost:
# Code Presentation

Olivier Rochaix, Erick Sanmartin,
Logan Sell, Nolan Thomas

# Data Processing - songs.csv

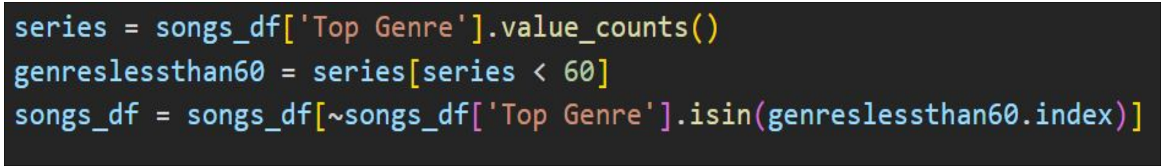
- Dropped unrelated columns
- Dropped rows for unique genres that skewed data
- Changed all data types to be a float



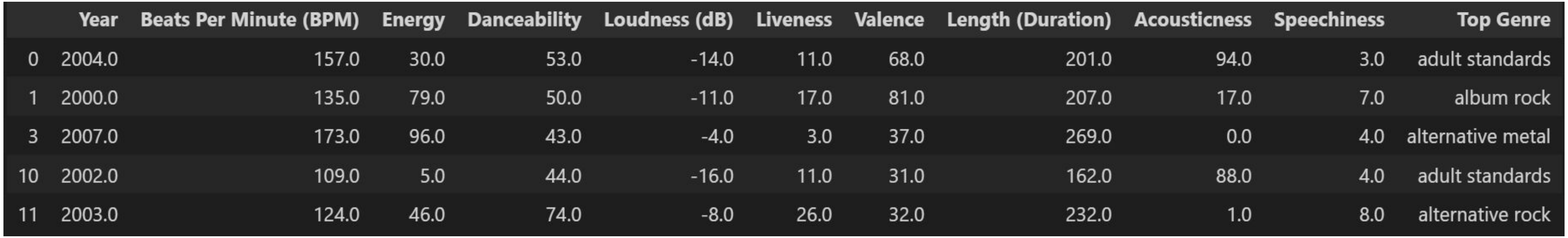
```
print(songs_df.shape)
songs_df.info()
```

✓ 0.1s

```
(940, 11)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 940 entries, 0 to 1993
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Year                    933 non-null    float64
 1   Beats Per Minute (BPM)  931 non-null    float64
 2   Energy                  932 non-null    float64
 3   Danceability            464 non-null    float64
 4   Loudness (dB)           934 non-null    float64
 5   Liveness                933 non-null    float64
 6   Valence                 433 non-null    float64
 7   Length (Duration)       932 non-null    float64
 8   Acousticness            933 non-null    float64
 9   Speechiness             937 non-null    float64
 10  Top Genre               932 non-null    object
dtypes: float64(10), object(1)
```

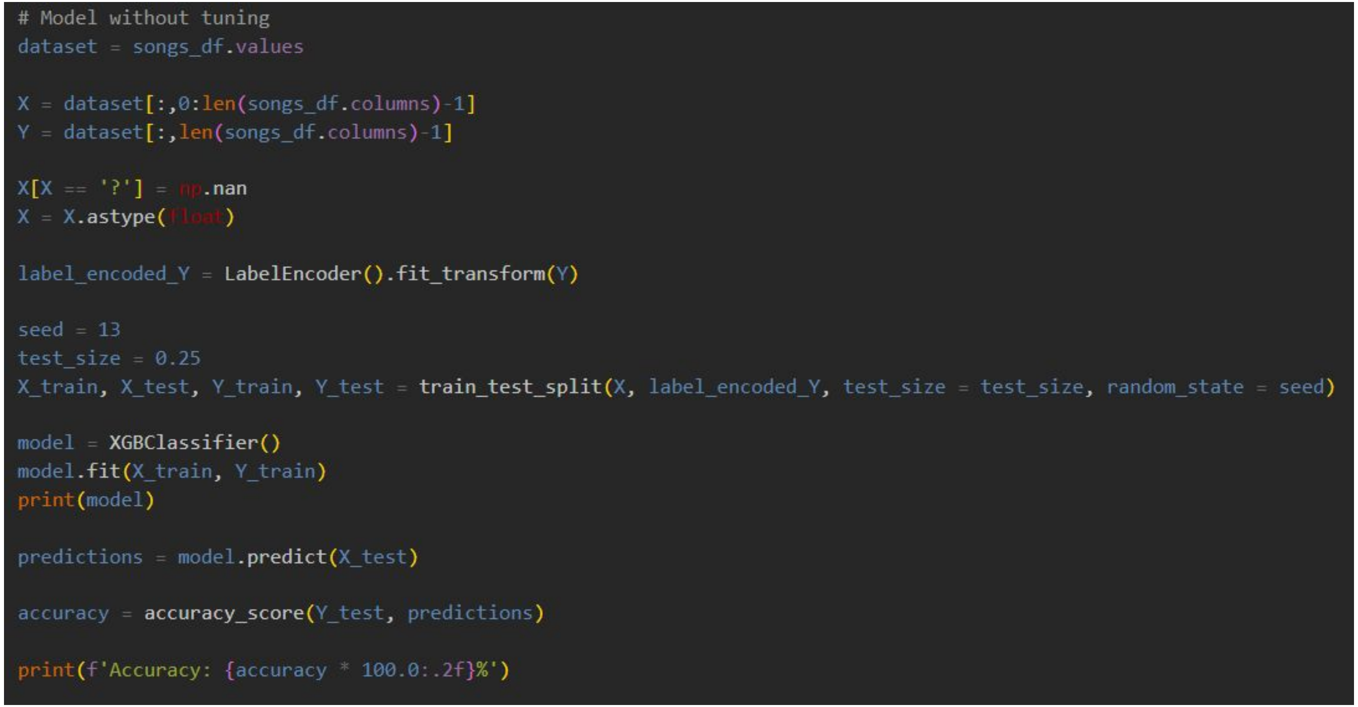


```
series = songs_df['Top Genre'].value_counts()
genreslessthan60 = series[series < 60]
songs_df = songs_df[~songs_df['Top Genre'].isin(genreslessthan60.index)]
```

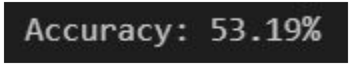


| | Year | Beats Per Minute (BPM) | Energy | Danceability | Loudness (dB) | Liveness | Valence | Length (Duration) | Acousticness | Speechiness | Top Genre |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2004.0 | 157.0 | 30.0 | 53.0 | -14.0 | 11.0 | 68.0 | 201.0 | 94.0 | 3.0 | adult standards |
| 1 | 2000.0 | 135.0 | 79.0 | 50.0 | -11.0 | 17.0 | 81.0 | 207.0 | 17.0 | 7.0 | album rock |
| 3 | 2007.0 | 173.0 | 96.0 | 43.0 | -4.0 | 3.0 | 37.0 | 269.0 | 0.0 | 4.0 | alternative metal |
| 10 | 2002.0 | 109.0 | 5.0 | 44.0 | -16.0 | 11.0 | 31.0 | 162.0 | 88.0 | 4.0 | adult standards |
| 11 | 2003.0 | 124.0 | 46.0 | 74.0 | -8.0 | 26.0 | 32.0 | 232.0 | 1.0 | 8.0 | alternative rock |

# First Implementation (No Tuning)



```
# Model without tuning
dataset = songs_df.values

X = dataset[:,0:len(songs_df.columns)-1]
Y = dataset[:,len(songs_df.columns)-1]

X[X == '?'] = np.nan
X = X.astype(float)

label_encoded_Y = LabelEncoder().fit_transform(Y)

seed = 13
test_size = 0.25
X_train, X_test, Y_train, Y_test = train_test_split(X, label_encoded_Y, test_size = test_size, random_state = seed)

model = XGBClassifier()
model.fit(X_train, Y_train)
print(model)

predictions = model.predict(X_test)

accuracy = accuracy_score(Y_test, predictions)

print(f'Accuracy: {accuracy * 100.0:.2f}%')
```

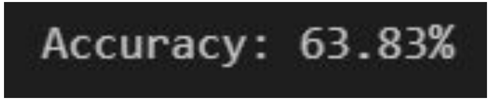


```
Accuracy: 53.19%
```

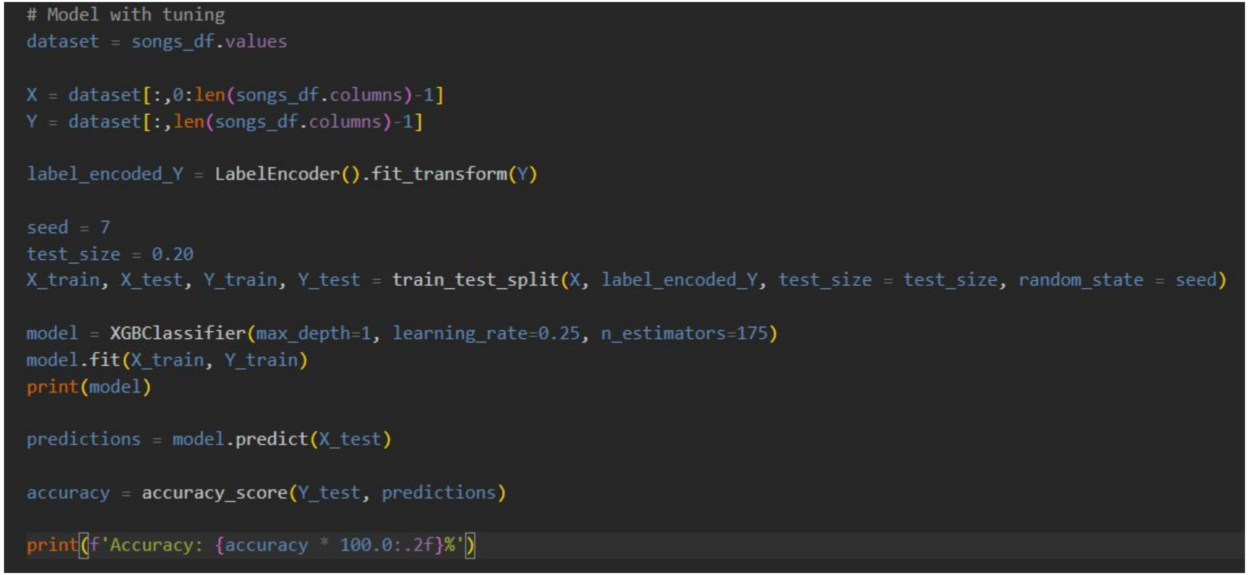
# Final Implementation (w/ Tuning)

## Hyperparameters:

- max_depth
- learning_rate
- n_estimators

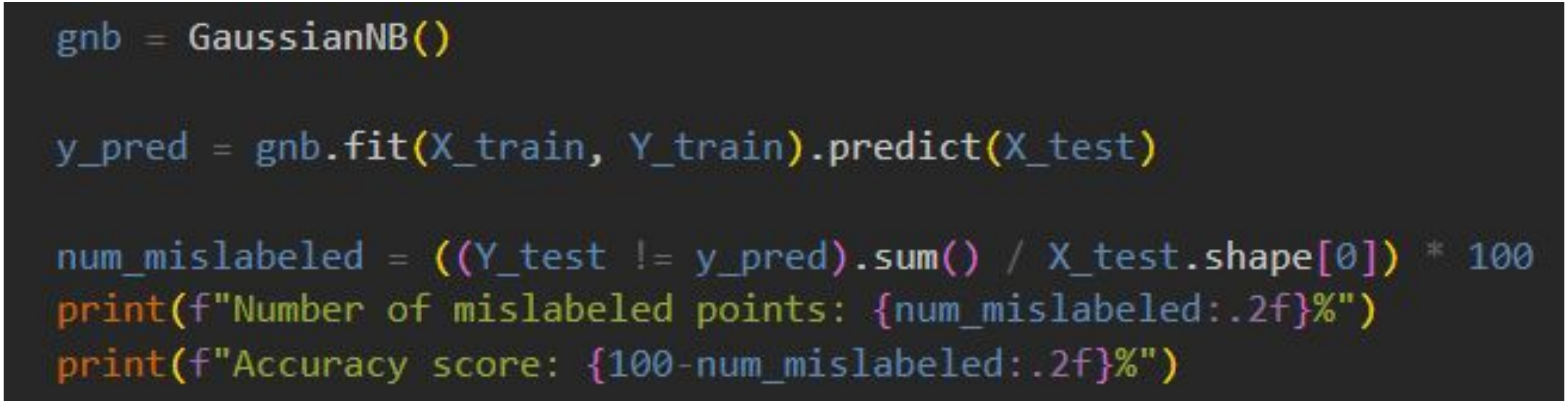


```
Accuracy: 63.83%
```

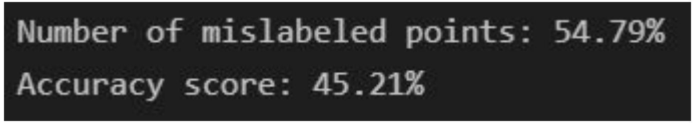


```python
# Model with tuning
dataset = songs_df.values

X = dataset[:,0:len(songs_df.columns)-1]
Y = dataset[:,len(songs_df.columns)-1]

label_encoded_Y = LabelEncoder().fit_transform(Y)

seed = 7
test_size = 0.20
X_train, X_test, Y_train, Y_test = train_test_split(X, label_encoded_Y, test_size = test_size, random_state = seed)

model = XGBClassifier(max_depth=1, learning_rate=0.25, n_estimators=175)
model.fit(X_train, Y_train)
print(model)

predictions = model.predict(X_test)

accuracy = accuracy_score(Y_test, predictions)

print(f'Accuracy: {accuracy * 100.0:.2f}%')
```

# Comparison (vs. Naive Bayes)



```
gnb = GaussianNB()

y_pred = gnb.fit(X_train, Y_train).predict(X_test)

num_mislabeled = ((Y_test != y_pred).sum() / X_test.shape[0]) * 100
print(f"Number of mislabeled points: {num_mislabeled:.2f}%")
print(f"Accuracy score: {100-num_mislabeled:.2f}%")
```



```
Number of mislabeled points: 54.79%
Accuracy score: 45.21%
```

# Suggestions for Improvement

- More **data**
- More **representative selection** of the music genres
- Potentially **standardize** the data
- Using **gridsearch**