KLASIFIKASI GENRE MUSIK

KELOMPOK 1

ANGGOTA KELOMPOK:
KEANE CLEMENT (2602084981)
KEVIN MAXWELL (2602083524)
DERREN MALAKA (2602081582)
JOSEL LINUS (2602076216)
RAFEL SUPRATMAN (2602073675)
HENDRA PUTRA (2602085095)
KELSTEN WUISAN (2602080932)

Introduksi

Kami akan membuat model machine learning yang dapat mengklasifikasikan sebuah lagu berdasarkan genre, mulai dari pop hingga rock dengan menggunakan machine learning.

Problem & Solution

Problem

- Genre: a style, especially in the arts, that involves a particular set of characteristics.
- Orang-orang menggunakan genre untuk mengkategorikan lagu.
- Tidak ada metode matematis yang bisa secara langsung mengkategorikan sebuah lagu ke sebuah genre.

Solution

- Menggunakan machine learning untuk secara otomatis mengkategorikan sebuah lagu ke sebuah genre.
- Menggunakan dataset berisikan instances lagu dalam bentuk waveform dan labelnya.
- Proses waveform lagu menggunakan librosa.

Dataset

Dataset yang akan kami gunakan untuk melaksanakan model machine learning ini adalah GTZAN Dataset - Music Genre Classification yang kami dapatkan dari kaggle.

Link Dataset:

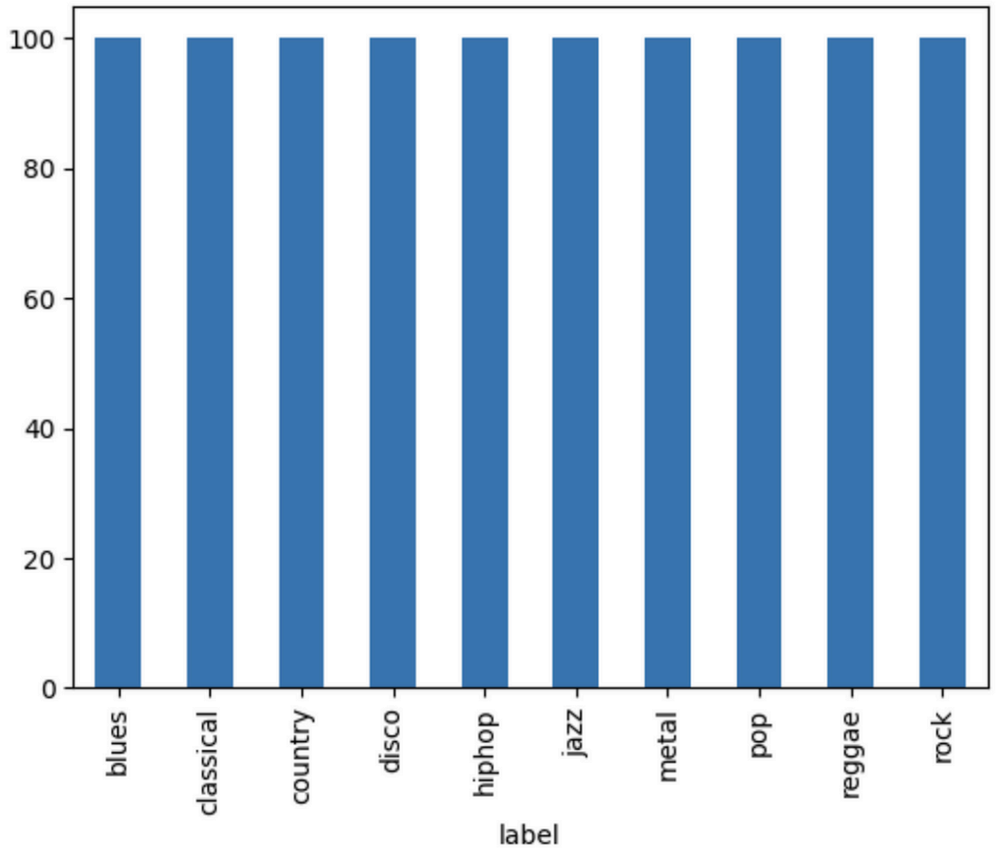
https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification/code

Data Flow

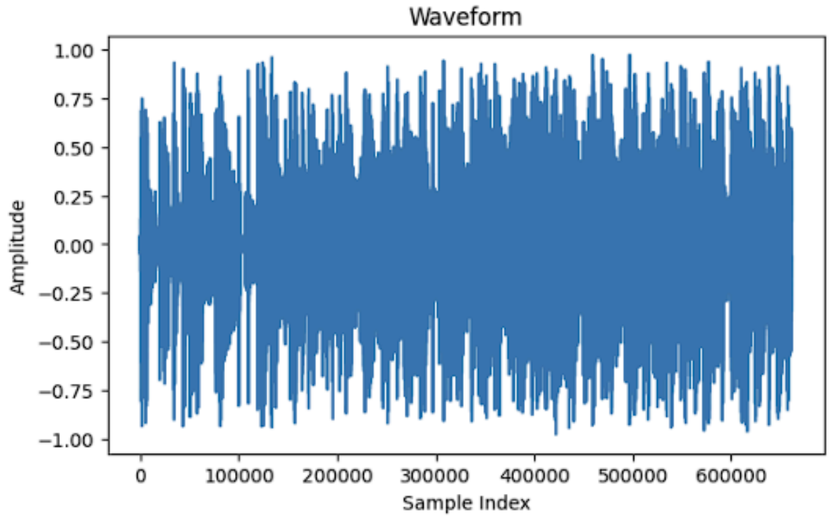
Waveform -> Librosa

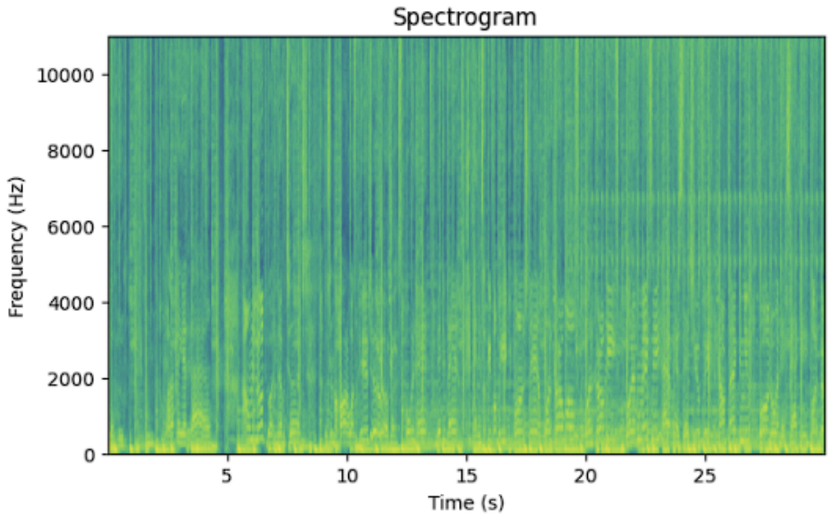
Classifier <- Audio Feature

Data Exploration



Data Exploration





Feature Extraction

Zero Crossing Rate
Harmonic & Perceptual
Spectral Centroid
Spectral Bandwidth
Spectral Rolloff
MFCC
Chroma
Root Mean Square
Tempo

Data Preprocessing

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split
```

```
df = df_3sec
df = df.drop(columns = 'filename')
le = LabelEncoder()
df['label_encoded'] = le.fit_transform(df['label'])

for i, label in enumerate(df['label'].unique()):
    print(i, label, end="| ")
df.pop('label')

scaler = MinMaxScaler()
X = df.drop(columns = ['label_encoded'])
X = scaler.fit_transform(X)

y = df['label_encoded']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_
size=0.3, random_state=42)
```

Correlation Matrix

label_encoded	1.000000
spectral_bandwidth_mean	0.376621
rolloff_mean	0.369515
spectral_centroid_mean	0.360175
chroma_stft_mean	0.330370
mfcc1_mean	0.326771
spectral_centroid_var	0.281352
rolloff_var	0.260298
zero_crossing_rate_mean	0.243590
zero_crossing_rate_var	0.215464
spectral_bandwidth_var	0.210972
mfcc9_mean	0.207059
rms_mean	0.205471
mfcc7_mean	0.198996
perceptr_var	0.170666
mfcc11_mean	0.152093
mfcc4_var	0.144013
mfcc6_var	0.131331
mfcc2_var	0.127176
harmony_var	0.115452
mfcc12_mean	0.111270
mfcc3_var	0.110916
mfcc7_var	0.107591
rms_var	0.101467
mfcc5_mean	0.100811
mfcc3_mean	0.089573
mfcc20_mean	0.083224
mfcc18_mean	0.082595
mfcc10_mean	0.074355
mfcc8_mean	0.071334
mfcc8_var	0.070706
mfcc13_mean	0.066110
mfcc15_mean	0.065622
mfcc1_var	0.059184
mfcc5_var	0.051668
mfcc14_mean	0.051521
mfcc9_var	0.041014
mfcc6_mean	0.039947
mfcc16_mean	0.026503
mfcc10_var	0.021395
mfcc19_mean	0.013143
tempo	0.012369
mfcc17_mean	0.004921
mfcc18_var	-0.039150
mfcc17_var	-0.040047
mfcc16_var	-0.044354
mfcc4_mean	-0.048384
mfcc11_var	-0.049489
harmony_mean	-0.050513
perceptr_mean	-0.053023
mfcc19_var	-0.059141
mfcc15_var	-0.067919
mfcc14_var	-0.068083
chroma_stft_var	-0.079683
mfcc13_var	-0.089063
mfcc12_var	-0.090699
mfcc20_var	-0.099627
mfcc2_mean	-0.348035
length	NaN

Name: label_encoded, dtype: float64

Method

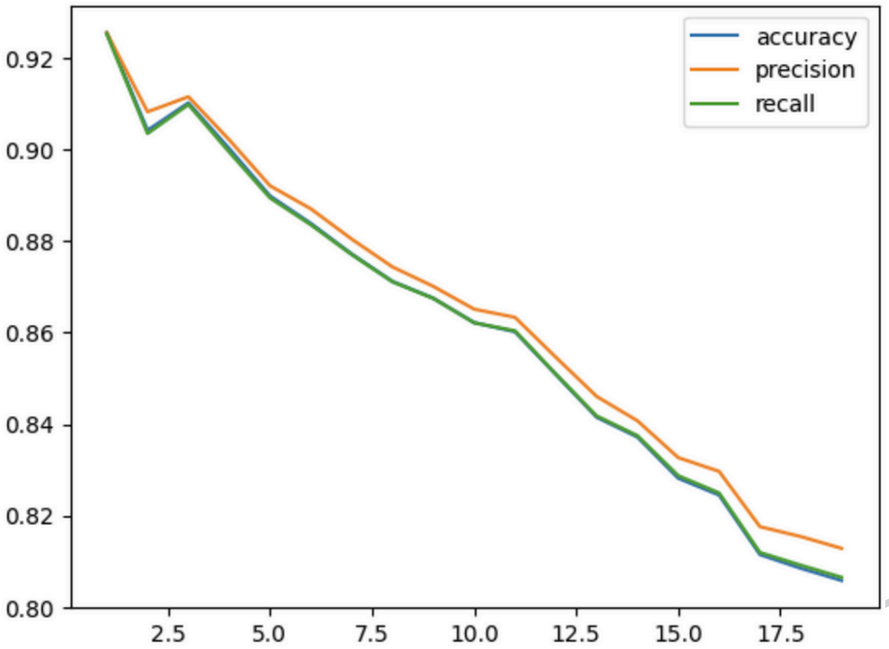
```
def modelling(model, title = "Default"):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    pre = precision_score(y_test, y_pred, average='macro')
    rec = recall_score(y_test, y_pred, average='macro')

    if title is not None: print('Accuracy', title, ':', acc)

    return acc, pre, rec
```

Method

```
# Naive Bayes
nb = GaussianNB()
modelling(nb, "Naive Bayes")

# KNN
knn = KNeighborsClassifier(n_neighbors=19)
modelling(knn, "KNN")

# Random Forest
rf = RandomForestClassifier(n_estimators=1000, max_depth=10, ra
ndom_state=0)
modelling(rf, "Random Forest")

# Cross Gradient Booster
xgb = XGBClassifier(n_estimators=1000, learning_rate=0.05)
modelling(xgb, "Cross Gradient Booster")
```

Model Evaluation KNN

```
acc = []
pre = []
rec = []

for i in range(1, 20):
    ac, pr, re = modelling(KNeighborsClassifier(n_neighbors=i),
None)
    acc.append(ac)
    pre.append(pr)
    rec.append(re)

x = range(1,20)
plt.plot(x, acc, label='accuracy')
plt.plot(x, pre, label='precision')
plt.plot(x, rec, label='recall')
plt.legend()
plt.show()
```



Evaluation

Accuracy Naive Bayes : 0.5195195195195195

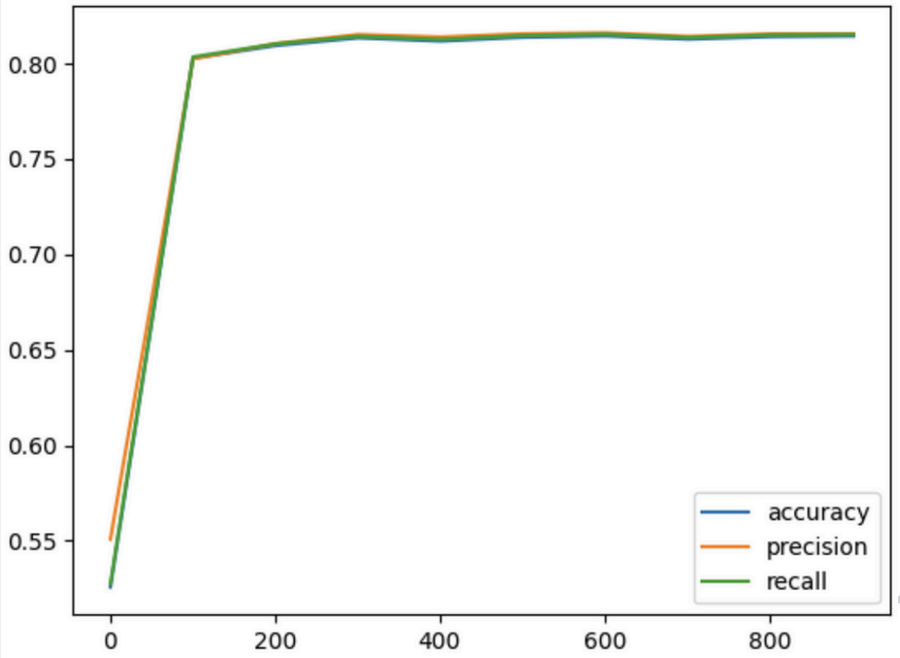
Accuracy KNN : 0.8058058058058059

Accuracy Random Forest : 0.8141474808141475

Accuracy Cross Gradient Booster : 0.9009009009009009

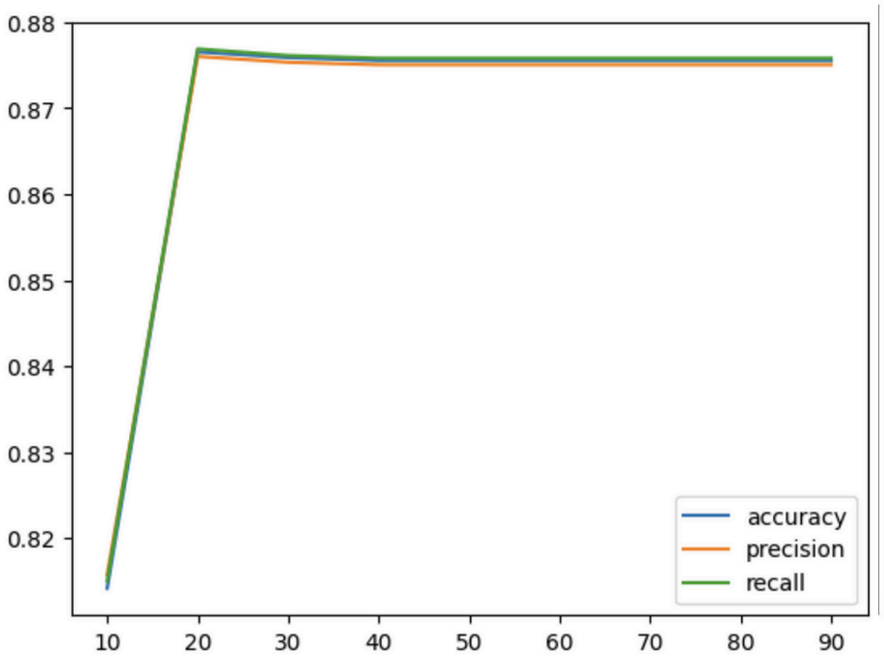
Model Evaluation Random Forest

```
acc = []
pre = []
rec = []

for i in range(1, 1000, 100):
    ac, pr, re = modelling(RandomForestClassifier(n_estimators=
i, max_depth=10, random_state=0), None)
    acc.append(ac)
    pre.append(pr)
    rec.append(re)

x = range(1, 1000, 100)
plt.plot(x, acc, label='accuracy')
plt.plot(x, pre, label='precision')
plt.plot(x, rec, label='recall')
plt.legend()
plt.show()
```



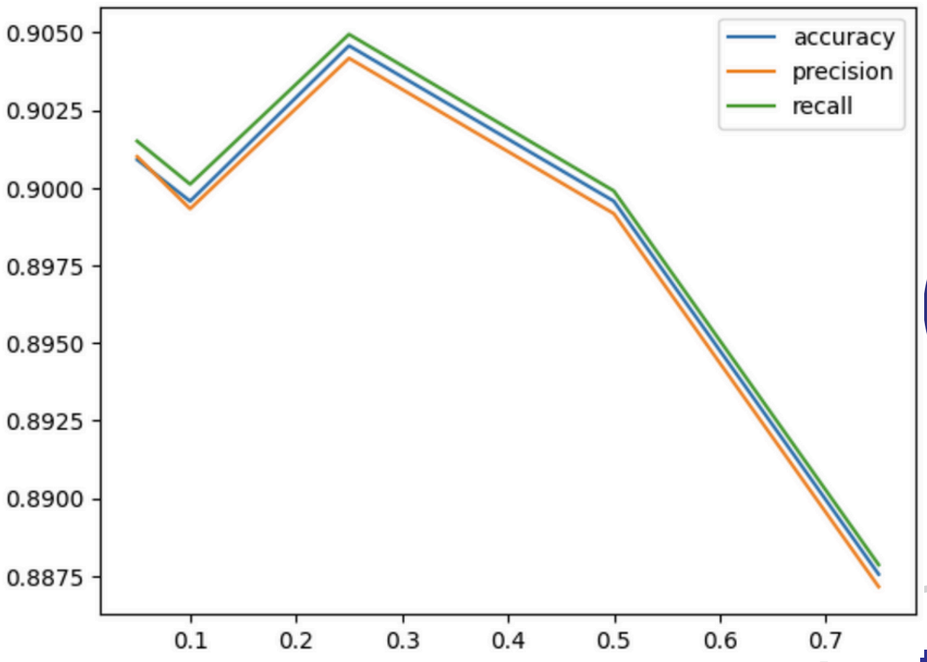
Model Evaluation Random Forest

```
acc = []
pre = []
rec = []

for i in range(10, 100, 10):
    ac, pr, re = modelling(RandomForestClassifier(n_estimators=
600, max_depth=i, random_state=0), None)
    acc.append(ac)
    pre.append(pr)
    rec.append(re)

x = range(10, 100, 10)
plt.plot(x, acc, label='accuracy')
plt.plot(x, pre, label='precision')
plt.plot(x, rec, label='recall')
plt.legend()
plt.show()
```



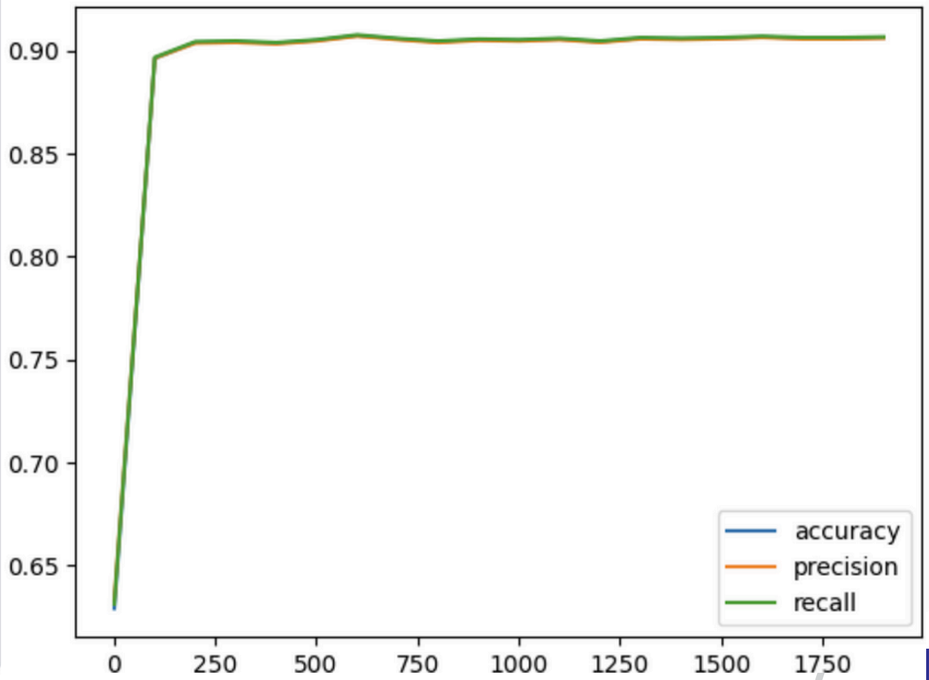
Model Evaluation GB Classifier

```
acc = []
pre = []
rec = []

for i in [0.05, 0.1, 0.25, 0.5, 0.75]:
    ac, pr, re = modelling(XGBClassifier(n_estimators=1000, lea
rning_rate=i), None)
    acc.append(ac)
    pre.append(pr)
    rec.append(re)

x = [0.05, 0.1, 0.25, 0.5, 0.75]
plt.plot(x, acc, label='accuracy')
plt.plot(x, pre, label='precision')
plt.plot(x, rec, label='recall')
plt.legend()
plt.show()
```



Model Evaluation GB Classifier

```
acc = []
pre = []
rec = []

for i in range(1, 2000, 100):
    ac, pr, re = modelling(XGBClassifier(n_estimators=i, learni
ng_rate=0.25), None)
    acc.append(ac)
    pre.append(pr)
    rec.append(re)

x = range(1, 2000, 100)
plt.plot(x, acc, label='accuracy')
plt.plot(x, pre, label='precision')
plt.plot(x, rec, label='recall')
plt.legend()
plt.show()
```



Kesimpulan

- Setelah di-tuning peforma model menjadi
  - KNN (k = 1) -> 92.5%
  - Random Forest (n_estimator = 600, max_depth = 20) -> 87.8%
  - XGBClassifier (n_estimator = 1000, learning_rate = 0.25) -> 90.5%

Saran:

Agar mendapatkan hasil model yang lebih baik lagi, kami menyarankan untuk menambah jumlah dari dataset, dan juga mengkombinasi durasi lagu baik yang memiliki durasi yang panjang maupun yang pendek saat ingin melakukan feature extraction pada masa training, agar model dapat memprediksi lagu dengan durasi yang cukup panjang maupun pendek

Demo

```
audio, sample_rate = librosa.load('/kaggle/input/gtzan-dataset-music-genre-classification/Data/genres_original/pop/pop.00050.wav')
print('Audio: ', audio)
print('Sample Rate: ', sample_rate)
print('Audio shape: ', audio.shape)

audio
ipd.display(ipd.Audio(audio, rate = sample_rate))
```

```
Audio:  [0.03704834 0.03735352 0.037323   ... 0.05224609 0.05944824 0.12362671]
Sample Rate:  22050
Audio shape:  (661504,)
```

Demo

```
ipd.display(ipd.Audio(audio, rate = sample_rate))
```

Demo

```
# Harmonics and Perceptrual
y_harmonic, y_perceptrual = librosa.effects.hpss(audio)
print('Harmonics: ', y_harmonic)
print('Harmonics shape: ', y_harmonic.shape)
print('Harmonics mean: ', y_harmonic.mean())
print('Harmonics var: ', y_harmonic.var())
print('Perceptrual: ', y_perceptrual)
print('Perceptrual shape: ', y_perceptrual.shape)
print('Perceptrual mean: ', y_perceptrual.mean())
print('Perceptrual var: ', y_perceptrual.var())
```

```
Harmonics:  [0.03326376 0.03181361 0.03337663 ... 0.01731614 0.00584044 0.02411156]
Harmonics shape:  (661504,)
Harmonics mean:  -3.5859378e-07
Harmonics var:  0.02375462
Perceptrual:  [0.00378458 0.00553991 0.00394636 ... 0.03492994 0.0536078  0.09951515]
Perceptrual shape:  (661504,)
Perceptrual mean:  -3.8225327e-05
Perceptrual var:  0.0125090685
```

Demo

```
# MFCC
mfcc = librosa.feature.mfcc(y = audio, sr = sample_rate)
print('MFCC: ', mfcc)
print('MFCC Shape: ', mfcc.shape)
print('MFCC Mean: ', mfcc.mean())
print('MFCC Var: ', mfcc.var())
plt.figure(figsize = (16, 6))
librosa.display.specshow(mfcc, sr=sample_rate, x_axis='time');
```

```
MFCC:  [[-232.74675   -125.48463    -56.231228  ...  -46.20056    -33.402805
   -31.713644 ]
 [  48.240204    44.47583     53.990505  ...   79.02171     56.27352
    54.7958   ]
 [  13.325691    22.516548    41.636967  ...  -30.524948   -13.2747345
    -1.5740416]
 ...
 [ -12.231359    -6.256031     3.6848311 ...   -8.87232     -5.096953
    -3.8704803]
 [  -9.302217     2.3334775    6.0767984 ...   -1.0246818   -4.763948
    -7.672885 ]
 [   4.2557316    6.3851395    1.1116682 ...  -12.019724    -5.4266586
    -3.7294238]]
MFCC Shape:  (20, 1293)
MFCC Mean:  4.024983
MFCC Var:  1010.4004
```

Demo

[84]:

```python
def train_model(model):
    model.fit(X_train, y_train)
    return model

knn = train_model(KNeighborsClassifier(n_neighbors=1))
rf = train_model(RandomForestClassifier(n_estimators=600, max_depth=20, random_state=0))
xgbc = train_model(XGBClassifier(n_estimators=600, learning_rate=0.25))
```

+ Code + Markdown

Demo

[92]:

```python
def extract_audio_features(file_path):
    y, sr = librosa.load(file_path, sr=None)

    features = {
        'length': y.shape[0] / sr,
        'chroma_stft_mean': np.mean(librosa.feature.chroma_stft(y=y, sr=sr)),
        'chroma_stft_var': np.var(librosa.feature.chroma_stft(y=y, sr=sr)),
        'rms_mean': np.mean(librosa.feature.rms(y=y)),
        'rms_var': np.var(librosa.feature.rms(y=y)),
        'spectral_centroid_mean': np.mean(librosa.feature.spectral_centroid(y=y, sr=sr)),
        'spectral_centroid_var': np.var(librosa.feature.spectral_centroid(y=y, sr=sr)),
        'spectral_bandwidth_mean': np.mean(librosa.feature.spectral_bandwidth(y=y, sr=sr)),
        'spectral_bandwidth_var': np.var(librosa.feature.spectral_bandwidth(y=y, sr=sr)),
        'rolloff_mean': np.mean(librosa.feature.spectral_rolloff(y=y, sr=sr)),
        'rolloff_var': np.var(librosa.feature.spectral_rolloff(y=y, sr=sr)),
        'zero_crossing_rate_mean': np.mean(librosa.feature.zero_crossing_rate(y)),
        'zero_crossing_rate_var': np.var(librosa.feature.zero_crossing_rate(y)),
        'harmony_mean': np.mean(librosa.effects.harmonic(y)),
        'harmony_var': np.var(librosa.effects.harmonic(y)),
        'perceptr_mean': np.mean(librosa.effects.percussive(y)),
        'perceptr_var': np.var(librosa.effects.percussive(y)),
        'tempo': librosa.beat.tempo(y=y, sr=sr)[0],
    }

    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20)
    for i in range(1, 21):
        features[f'mfcc{i}_mean'] = np.mean(mfcc[i-1])
        features[f'mfcc{i}_var'] = np.var(mfcc[i-1])

    return pd.DataFrame(features, index=[0])
```

Demo

[93]:
```
file_path = '/kaggle/input/tes-lagu/Baby-Justin Bieber.mp3'
test_audio = extract_audio_features(file_path)
```

Expand

[99]:
```
print(dict(zip(le.classes_, le.transform(le.classes_))))
```

```
{'blues': 0, 'classical': 1, 'country': 2, 'disco': 3, 'hiphop': 4, 'jazz': 5, 'metal': 6, 'pop': 7, 'reggae': 8, 'rock': 9}
```

[98]:
```
col = test_audio.columns
test_audio = pd.DataFrame(scaler.transform(test_audio), columns=col)
print(knn.predict(test_audio))
print(rf.predict(test_audio))
print(xgbc.predict(test_audio))
```

```
[4]
[4]
[8]
```

KNN: Hiphop, Random Forest: Hiphop, Xgbc: Reggae

Terimakasih