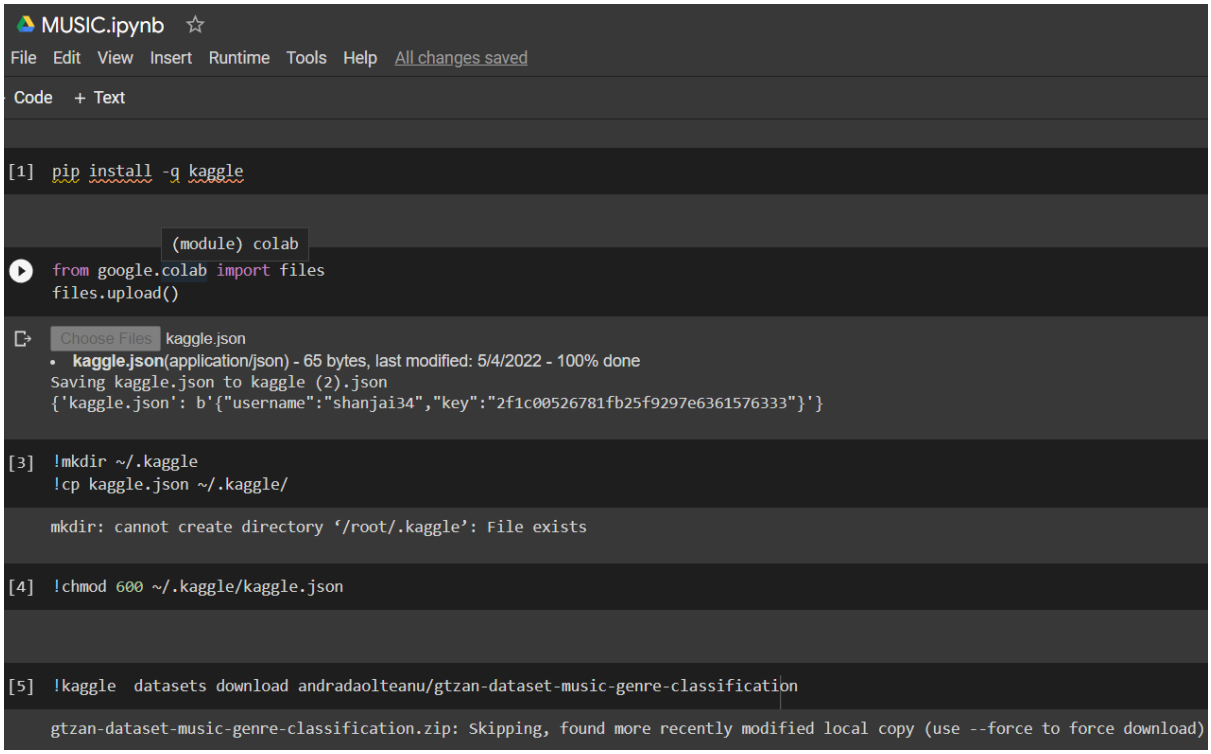
Upload Kaggle dataset in google colab:



Export data:

Importing libraries:

import libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import sklearn.metrics as skm
import sklearn.model_selection as skms
import sklearn.preprocessing as skp
import random
import librosa, IPython
import librosa.display as lplt
seed = 12
np.random.seed(seed)
```

```
df = pd.read_csv('Data/features_3_sec.csv')
df.head()
```

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spectral_bandwidth_var	...	m
0	blues.00000.0.wav	66149	0.335406	0.091048	0.130405	0.003521	1773.065032	167541.630869	1972.744388	117335.771563	...	
1	blues.00000.1.wav	66149	0.343065	0.086147	0.112699	0.001450	1816.693777	90525.690866	2010.051501	65671.875673	...	
2	blues.00000.2.wav	66149	0.346815	0.092243	0.132003	0.004620	1788.539719	111407.437613	2084.565132	75124.921716	...	
3	blues.00000.3.wav	66149	0.363639	0.086856	0.132565	0.002448	1655.289045	111952.284517	1960.039988	82913.639269	...	
4	blues.00000.4.wav	66149	0.335579	0.088129	0.143289	0.001701	1630.656199	79667.267654	1948.503884	60204.020268	...	

5 rows × 60 columns

```
print("Dataset has",df.shape)
print("Count of Positive and Negative samples")
df.label.value_counts().reset_index()
```

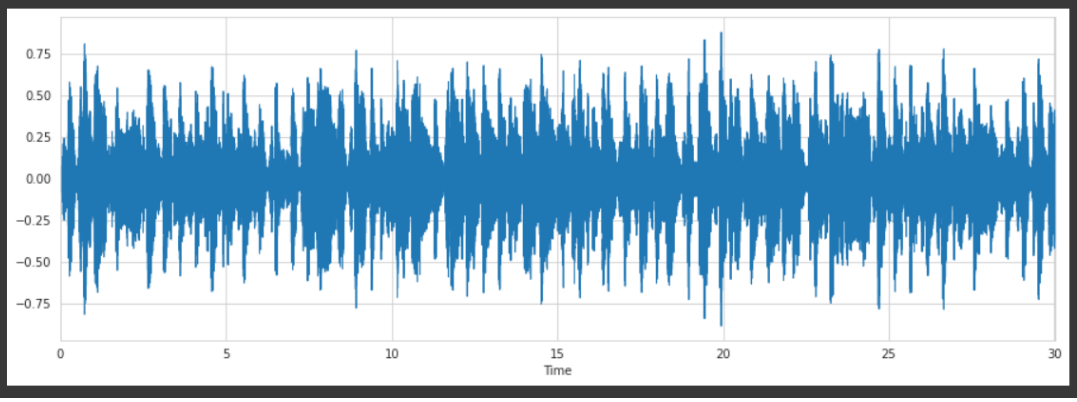
```
Dataset has (9990, 60)
Count of Positive and Negative samples
```

	index	label
0	blues	1000
1	jazz	1000
2	metal	1000
3	pop	1000
4	reggae	1000
5	disco	999
6	classical	998
7	hiphop	998
8	rock	998
9	country	997

```
[10] audio_fp = 'Data/genres_original/blues/blues.00000.wav'
     audio_data, sr = librosa.load(audio_fp)
     audio_data, _ = librosa.effects.trim(audio_data)
```

```
[11] # play sample file
     IPython.display.Audio(audio_data, rate=sr)
```

```
# plot sample file
plt.figure(figsize=(15,5))
lplt.waveplot(audio_data)
plt.show()
```

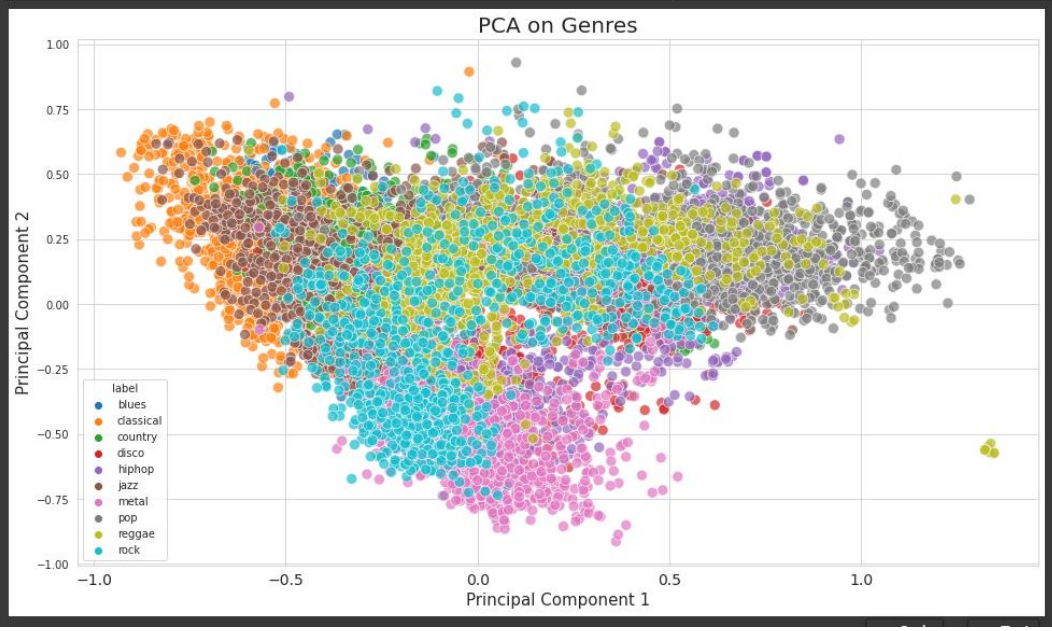


Preprocessing:

PCA

```
data = df.iloc[0:, 1:]
y = data['label']
X = data.loc[:, data.columns != 'label']

# normalize
cols = X.columns
min_max_scaler = skp.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(X)
X = pd.DataFrame(np_scaled, columns = cols)

# Top 2 pca components
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X)
principalDf = pd.DataFrame(data = principalComponents, columns = ['pc1', 'pc2'])

# concatenate with target label
finalDf = pd.concat([principalDf, y], axis = 1)

plt.figure(figsize = (16, 9))
sns.scatterplot(x = "pc1", y = "pc2", data = finalDf, hue = "label", alpha = 0.7, s = 100);
```

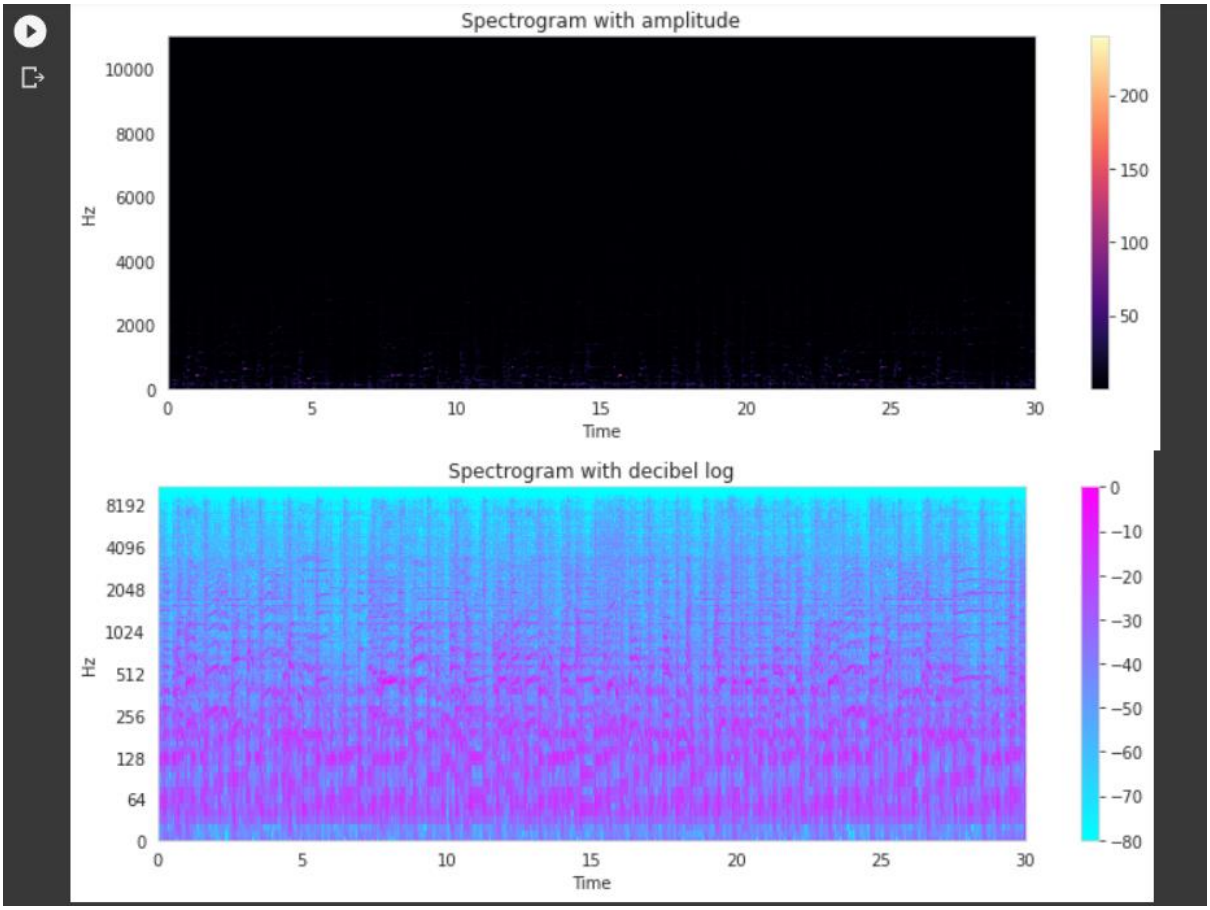
```
plt.title('PCA on Genres', fontsize = 20)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 10);
plt.xlabel("Principal Component 1", fontsize = 15)
plt.ylabel("Principal Component 2", fontsize = 15)
plt.savefig("PCA_Scattert.png")
```



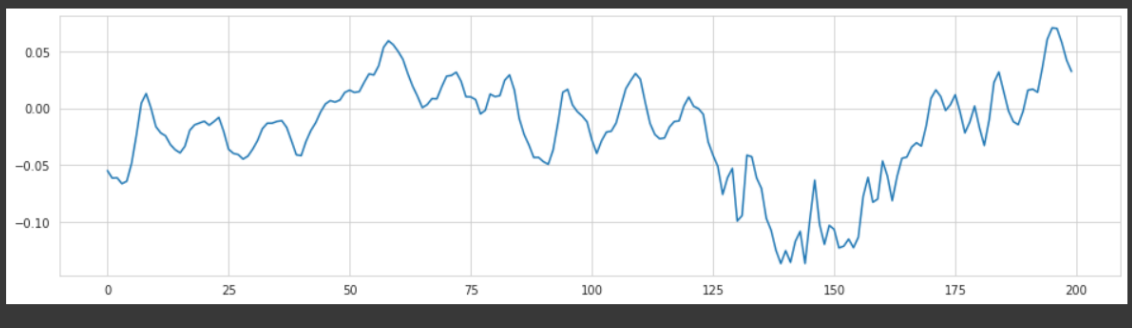
Feature Extraction:

FEATURE EXTRACTION

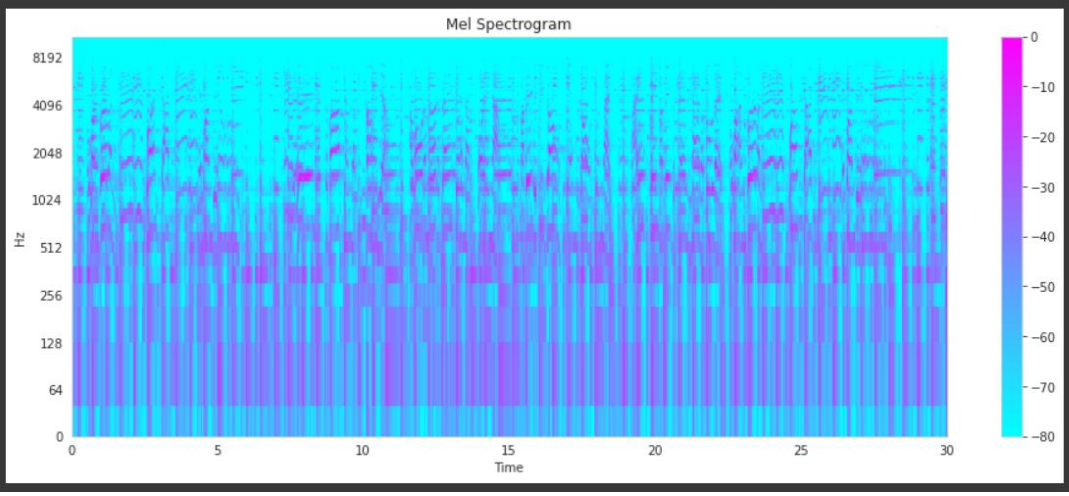
```
[14] # Default FFT window size
     n_fft = 2048 # window size
     hop_length = 512 # window hop length for STFT

     stft = librosa.stft(audio_data, n_fft=n_fft, hop_length=hop_length)
     stft_db = librosa.amplitude_to_db(stft, ref=np.max)

     plt.figure(figsize=(12,4))
     lplt.specshow(stft, sr=sr, x_axis='time', y_axis='hz')
     plt.colorbar()
     plt.title("Spectrogram with amplitude")
     plt.show()

     plt.figure(figsize=(12,4))
     lplt.specshow(stft_db, sr=sr, x_axis='time', y_axis='log', cmap='cool')
     plt.colorbar()
     plt.title("Spectrogram with decibel log")
     plt.show()
```



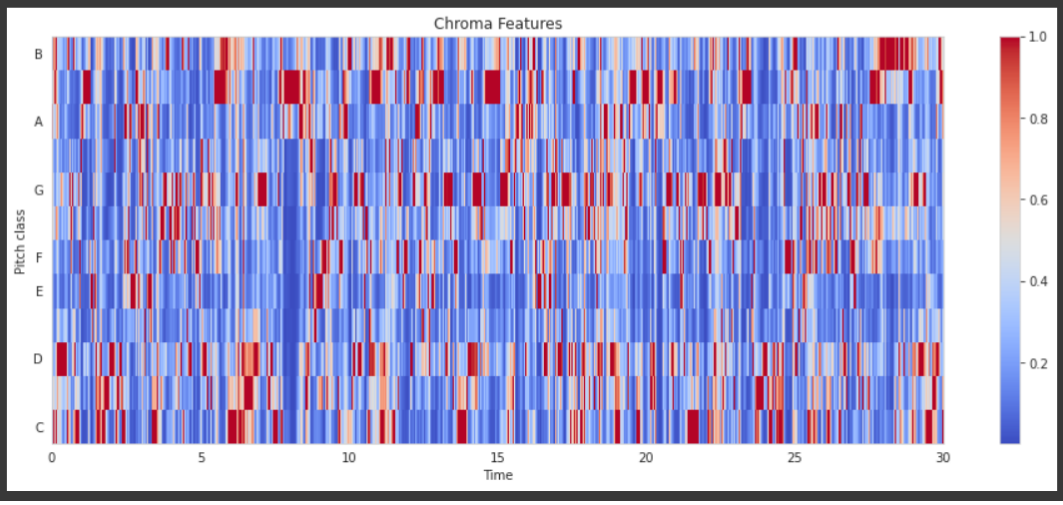

```
# plot zoomed audio wave
start = 1000
end = 1200
plt.figure(figsize=(16,4))
plt.plot(audio_data[start:end])
plt.show()
```



```
mel_spec = librosa.feature.melspectrogram(audio_data, sr=sr)
mel_spec_db = librosa.amplitude_to_db(mel_spec, ref=np.max)
plt.figure(figsize=(16,6))
lplt.specshow(mel_spec_db, sr=sr, hop_length=hop_length, x_axis='time', y_axis='log', cmap='cool')
plt.colorbar()
plt.title("Mel Spectrogram")
plt.show()
```

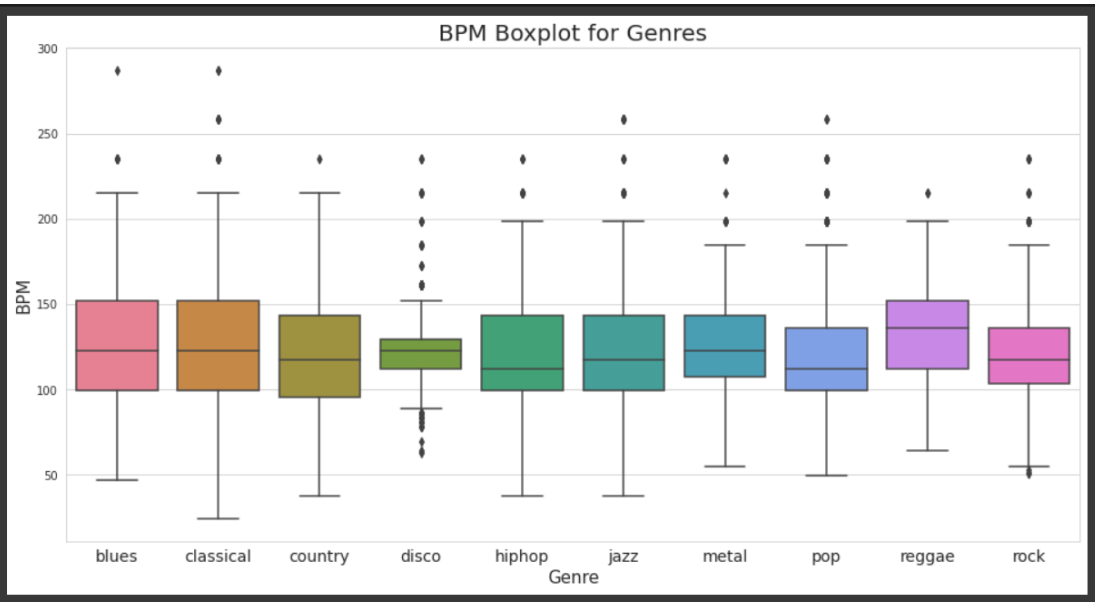



```
chroma = librosa.feature.chroma_stft(audio_data, sr=sr)
plt.figure(figsize=(16,6))
lplt.specshow(chroma, sr=sr, x_axis='time', y_axis='chroma', cmap='coolwarm')
plt.colorbar()
plt.title("Chroma Features")
plt.show()
```



```
x = df[["label", "tempo"]]

fig, ax = plt.subplots(figsize=(16, 8));
sns.boxplot(x = "label", y = "tempo", data = x, palette = 'husl');

plt.title('BPM Boxplot for Genres', fontsize = 20)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 10);
plt.xlabel("Genre", fontsize = 15)
plt.ylabel("BPM", fontsize = 15)
plt.savefig("BPM_Boxplot.png")
```

[19]



Missing value treatment

```
[20] # find all columns with any NA values
     print("Columns with NA values are",list(df.columns[df.isnull().any()]))
```

```
Columns with NA values are []
```

Encode genre label

```
[21] # map labels to index
     label_index = dict()
     index_label = dict()
     for i, x in enumerate(df.label.unique()):
         label_index[x] = i
         index_label[i] = x
     print(label_index)
     print(index_label)
```

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

Train and Test split:

```
[22] # update labels in df to index
     df.label = [label_index[l] for l in df.label]
```

Train&Test

```
[23] # shuffle samples
     df_shuffle = df.sample(frac=1, random_state=seed).reset_index(drop=True)
```

```
[24] # remove irrelevant columns
     df_shuffle.drop(['filename', 'length'], axis=1, inplace=True)
     df_y = df_shuffle.pop('label')
     df_X = df_shuffle

     # split into train dev and test
     X_train, df_test_valid_X, y_train, df_test_valid_y = skms.train_test_split(df_X, df_y, train_size=0.7, random_state=seed, stratify=df_y)
     X_dev, X_test, y_dev, y_test = skms.train_test_split(df_test_valid_X, df_test_valid_y, train_size=0.66, random_state=seed, stratify=df_test_valid_y)
```

```
print(f"Train set has {X_train.shape[0]} records out of {len(df_shuffle)} which is {round(X_train.shape[0]/len(df_shuffle)*100)}%")
print(f"Dev set has {X_dev.shape[0]} records out of {len(df_shuffle)} which is {round(X_dev.shape[0]/len(df_shuffle)*100)}%")
print(f"Test set has {X_test.shape[0]} records out of {len(df_shuffle)} which is {round(X_test.shape[0]/len(df_shuffle)*100)}%")
```

```
Train set has 6993 records out of 9990 which is 70%
Dev set has 1978 records out of 9990 which is 20%
Test set has 1019 records out of 9990 which is 10%
```

```
[27] scaler = skp.StandardScaler()
     X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)
     X_dev = pd.DataFrame(scaler.transform(X_dev), columns=X_train.columns)
     X_test = pd.DataFrame(scaler.transform(X_test), columns=X_train.columns)
```

```
[28] import tensorflow as tf
     print("TF version:-", tf.__version__)
     import keras as k
     tf.random.set_seed(seed)
```

```
TF version:- 2.8.0
```

```
ACCURACY_THRESHOLD = 0.94

class myCallback(k.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('val_accuracy') > ACCURACY_THRESHOLD):
            print("\n\nStopping training as we have reached %2.2f%% accuracy!" %(ACCURACY_THRESHOLD*100))
            self.model.stop_training = True

def trainModel(model, epochs, optimizer):
    batch_size = 128
    callback = myCallback()
    model.compile(optimizer=optimizer,
                  loss='sparse_categorical_crossentropy',
                  metrics='accuracy'
    )
    return model.fit(X_train, y_train, validation_data=(X_dev, y_dev), epochs=epochs,
                     batch_size=batch_size, callbacks=[callback])

def plotHistory(history):
    print("Max. Validation Accuracy",max(history.history["val_accuracy"]))
    pd.DataFrame(history.history).plot(figsize=(12,6))
    plt.show()
```

Training:

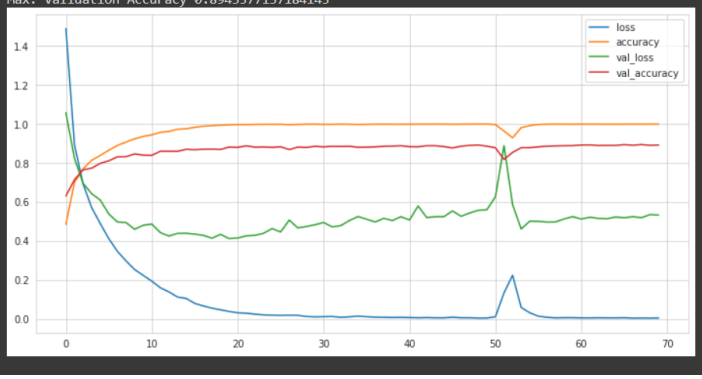
```
model_1 = k.models.Sequential([
    k.layers.Dense(256, activation='relu', input_shape=(X_train.shape[1],)),
    k.layers.Dense(128, activation='relu'),
    k.layers.Dense(64, activation='relu'),
    k.layers.Dense(10, activation='softmax'),
])
print(model_1.summary())
model_1_history = trainModel(model=model_1, epochs=70, optimizer='adam')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               14848

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 56,650
Trainable params: 56,650
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 68/70
55/55 [==============================] - 0s 5ms/step - loss: 0.0035 - accuracy: 0.9989 - val_loss: 0.5188 - val_accuracy: 0.8943
Epoch 69/70
55/55 [==============================] - 0s 5ms/step - loss: 0.0031 - accuracy: 0.9991 - val_loss: 0.5346 - val_accuracy: 0.8903
Epoch 70/70
55/55 [==============================] - 0s 4ms/step - loss: 0.0037 - accuracy: 0.9990 - val_loss: 0.5322 - val_accuracy: 0.8918
```

```
plotHistory(model_1_history)
```

```
Max. Validation Accuracy 0.8943377137184143
```



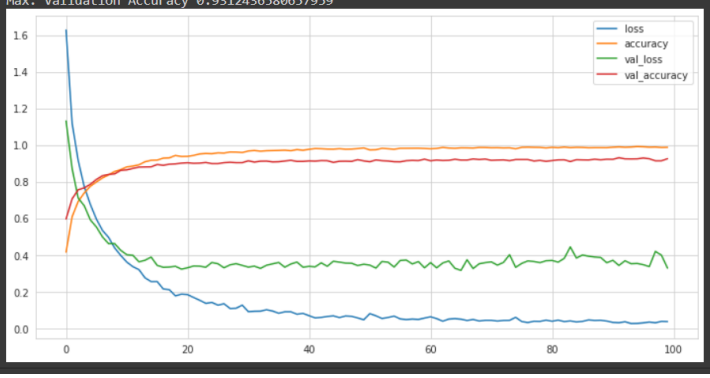
```python
model_2 = k.models.Sequential([
    k.layers.Dense(512, activation='relu', input_shape=(X_train.shape[1],)),
    k.layers.Dropout(0.2),

    k.layers.Dense(256, activation='relu'),
    k.layers.Dropout(0.2),

    k.layers.Dense(128, activation='relu'),
    k.layers.Dropout(0.2),

    k.layers.Dense(64, activation='relu'),
    k.layers.Dropout(0.2),

    k.layers.Dense(10, activation='softmax'),
])
print(model_2.summary())
model_2_history = trainModel(model=model_2, epochs=100, optimizer='adam')
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               29696

dropout (Dropout)            (None, 512)               0

dense_5 (Dense)              (None, 256)               131328

dropout_1 (Dropout)          (None, 256)               0

dense_6 (Dense)              (None, 128)               32896

dropout_2 (Dropout)          (None, 128)               0

dense_7 (Dense)              (None, 64)                8256

dropout_3 (Dropout)          (None, 64)                0

dense_8 (Dense)              (None, 10)                650
```

```
Epoch 97/100
55/55 [==============================] - 1s 16ms/step - loss: 0.0356 - accuracy: 0.9888 - val_loss: 0.3379 - val_accuracy: 0.9257
Epoch 98/100
55/55 [==============================] - 1s 13ms/step - loss: 0.0316 - accuracy: 0.9896 - val_loss: 0.4207 - val_accuracy: 0.9146
Epoch 99/100
55/55 [==============================] - 1s 18ms/step - loss: 0.0393 - accuracy: 0.9873 - val_loss: 0.4001 - val_accuracy: 0.9146
Epoch 100/100
55/55 [==============================] - 1s 16ms/step - loss: 0.0383 - accuracy: 0.9880 - val_loss: 0.3295 - val_accuracy: 0.9262
```

```
[33] plotHistory(model_2_history)
```

```
Max. Validation Accuracy 0.9312436580657959
```



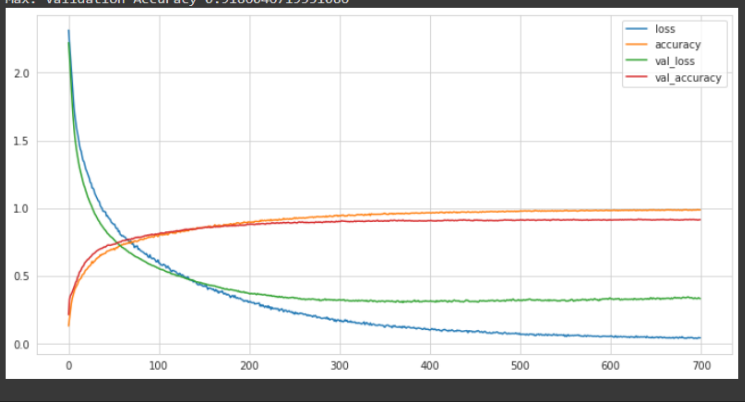

```python
model_3 = k.models.Sequential([
    k.layers.Dense(512, activation='relu', input_shape=(X_train.shape[1],)),
    k.layers.Dropout(0.2),

    k.layers.Dense(256, activation='relu'),
    k.layers.Dropout(0.2),

    k.layers.Dense(128, activation='relu'),
    k.layers.Dropout(0.2),

    k.layers.Dense(64, activation='relu'),
    k.layers.Dropout(0.2),

    k.layers.Dense(10, activation='softmax'),
])
print(model_3.summary())
model_3_history = trainModel(model=model_3, epochs=700, optimizer='sgd')
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 512)               29696

dropout_4 (Dropout)          (None, 512)               0

dense_10 (Dense)             (None, 256)               131328

dropout_5 (Dropout)          (None, 256)               0

dense_11 (Dense)             (None, 128)               32896

dropout_6 (Dropout)          (None, 128)               0

dense_12 (Dense)             (None, 64)                8256

dropout_7 (Dropout)          (None, 64)                0

dense_13 (Dense)             (None, 10)                650
```

```
55/55 [==============================] - 1s 10ms/step - loss: 0.0421 - accuracy: 0.9866 - val_loss: 0.3340 - val_accuracy: 0.9151
Epoch 696/700
55/55 [==============================] - 1s 9ms/step - loss: 0.0398 - accuracy: 0.9864 - val_loss: 0.3339 - val_accuracy: 0.9151
Epoch 697/700
55/55 [==============================] - 1s 10ms/step - loss: 0.0396 - accuracy: 0.9890 - val_loss: 0.3327 - val_accuracy: 0.9125
Epoch 698/700
55/55 [==============================] - 1s 10ms/step - loss: 0.0450 - accuracy: 0.9871 - val_loss: 0.3387 - val_accuracy: 0.9130
Epoch 699/700
55/55 [==============================] - 1s 10ms/step - loss: 0.0436 - accuracy: 0.9868 - val_loss: 0.3358 - val_accuracy: 0.9151
Epoch 700/700
55/55 [==============================] - 1s 9ms/step - loss: 0.0435 - accuracy: 0.9873 - val_loss: 0.3320 - val_accuracy: 0.9151
```

```
plotHistory(model_3_history)
```

```
Max. Validation Accuracy 0.9186046719551086
```



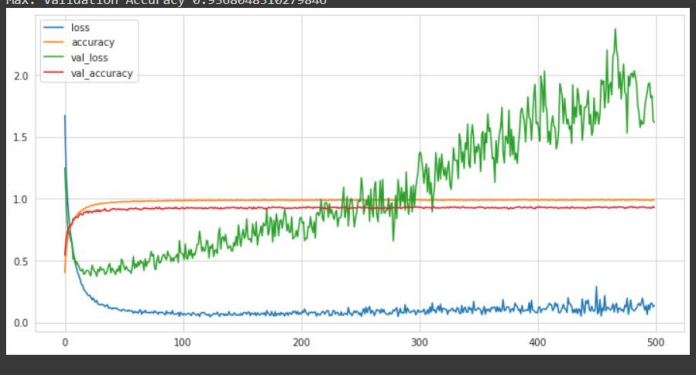
```
model_4 = k.models.Sequential([
    k.layers.Dense(1024, activation='relu', input_shape=(X_train.shape[1],)),
    k.layers.Dropout(0.3),

    k.layers.Dense(512, activation='relu'),
    k.layers.Dropout(0.3),

    k.layers.Dense(256, activation='relu'),
    k.layers.Dropout(0.3),

    k.layers.Dense(128, activation='relu'),
    k.layers.Dropout(0.3),

    k.layers.Dense(64, activation='relu'),
    k.layers.Dropout(0.3),

    k.layers.Dense(10, activation='softmax'),
])
print(model_4.summary())
model_4_history = trainModel(model=model_4, epochs=500, optimizer='rmsprop')
```

```
=================================================================
 dense_14 (Dense)            (None, 1024)              59392

 dropout_8 (Dropout)         (None, 1024)              0

 dense_15 (Dense)            (None, 512)               524800

 dropout_9 (Dropout)         (None, 512)               0

 dense_16 (Dense)            (None, 256)               131328

 dropout_10 (Dropout)        (None, 256)               0

 dense_17 (Dense)            (None, 128)               32896

 dropout_11 (Dropout)        (None, 128)               0

 dense_18 (Dense)            (None, 64)                8256

 dropout_12 (Dropout)        (None, 64)                0

 dense_19 (Dense)            (None, 10)                650
```

```
Epoch 324/500
55/55 [==============================] - 1s 24ms/step - loss: 0.1047 - accuracy: 0.9887 - val_loss: 1.1531 - val_accuracy: 0.9297
Epoch 325/500
55/55 [==============================] - 1s 25ms/step - loss: 0.0640 - accuracy: 0.9894 - val_loss: 1.2456 - val_accuracy: 0.9292
Epoch 326/500
55/55 [==============================] - 1s 24ms/step - loss: 0.0966 - accuracy: 0.9894 - val_loss: 1.3029 - val_accuracy: 0.9247
Epoch 327/500
```

```
plotHistory(model_4_history)
```

```
Max. Validation Accuracy 0.9368048310279846
```



```
[38] test_loss, test_acc  = model_4.evaluate(X_test, y_test, batch_size=128)
     print("The test Loss is :",test_loss)
     print("\nThe Best test Accuracy is :",test_acc*100)

     8/8 [==============================] - 0s 5ms/step - loss: 1.7462 - accuracy: 0.9293
     The test Loss is : 1.7461915016174316

     The Best test Accuracy is : 92.934250831604
```

Testing Prediction:

```
dict2={0: 'blues', 1: 'classical', 2: 'country', 3: 'disco', 4: 'hiphop', 5: 'jazz', 6: 'metal', 7: 'pop', 8: 'reggae', 9: 'rock'}
predicted_values=pd.DataFrame(y_test)
predicted_values=predicted_values.replace({"label":dict2})
predicted_values.reset_index(drop=True, inplace=True)
predicted_values=list(predicted_values.label)
i=0
for item in predicted_values:
    print("The prediction of song no. {} in our dataset is {}".format(i,item))
    i=i+1
```

```
The prediction of song no. 957 in our dataset is rock
The prediction of song no. 958 in our dataset is metal
The prediction of song no. 959 in our dataset is classical
The prediction of song no. 960 in our dataset is blues
The prediction of song no. 961 in our dataset is jazz
The prediction of song no. 962 in our dataset is hiphop
The prediction of song no. 963 in our dataset is disco
The prediction of song no. 964 in our dataset is reggae
The prediction of song no. 965 in our dataset is blues
The prediction of song no. 966 in our dataset is rock
The prediction of song no. 967 in our dataset is pop
The prediction of song no. 968 in our dataset is blues
The prediction of song no. 969 in our dataset is rock
The prediction of song no. 970 in our dataset is disco
The prediction of song no. 971 in our dataset is pop
The prediction of song no. 972 in our dataset is classical
The prediction of song no. 973 in our dataset is blues
The prediction of song no. 974 in our dataset is rock
The prediction of song no. 975 in our dataset is reggae
The prediction of song no. 976 in our dataset is hiphop
```