```
# Importing Data (Same in all notebooks)
```

In [1]:

```
from sklearn.naive_bayes import MultinomialNB
import pandas as pd
```

In [2]:

```
import  numpy  as  np
import pandas as pd
import string
```

In [3]:

```
data = pd.read_csv('/users/rohanchitte/downloads/Dataset_lyrics.csv_lyrics.csv')
```

```
# Data Preprocessing (Same in all notebooks)
```

In [4]:

```
filtered = data[data['lyrics'].notnull()]
filtered
```

Out[4]:

	index	song	year	artist	genre	lyrics
0	0	ego-remix	2009	beyonce-knowles	Pop	Oh baby, how you doing?\nYou know I'm gonna cu...
1	1	then-tell-me	2009	beyonce-knowles	Pop	playin' everything so easy,\nit's like you see...
2	2	honesty	2009	beyonce-knowles	Pop	If you search\nFor tenderness\nIt isn't hard t...
3	3	you-are-my-rock	2009	beyonce-knowles	Pop	Oh oh oh I, oh oh oh I\n[Verse 1:]\nIf I wrote...
4	4	black-culture	2009	beyonce-knowles	Pop	Party the people, the people the party it's po...
...	...	...	...	...	...	...
362232	362232	who-am-i-drinking-tonight	2012	edens-edge	Country	I gotta say\nBoy, after only just a couple of ...
362233	362233	liar	2012	edens-edge	Country	I helped you find her diamond ring\nYou made m...
362234	362234	last-supper	2012	edens-edge	Country	Look at the couple in the corner booth\nLooks ...
362235	362235	christ-alone-live-in-studio	2012	edens-edge	Country	When I fly off this mortal earth\nAnd I'm meas...
362236	362236	amen	2012	edens-edge	Country	I heard from a friend of a friend of a friend ...

266557 rows × 6 columns

In [5]:

```
import  nltk
from  nltk . corpus  import  stopwords
```

In [6]:

```
cleaned = filtered.copy()

# Remove punctuation
cleaned['lyrics'] = cleaned['lyrics'].str.replace("[-\?.,\/#!$%\^&\*;:{}=\_~()]'

# Remove song-related identifiers like [Chorus] or [Verse]
cleaned['lyrics'] = cleaned['lyrics'].str.replace("\[(.*?)\]", ' ')
cleaned['lyrics'] = cleaned['lyrics'].str.replace("' | '", ' ')
cleaned['lyrics'] = cleaned['lyrics'].str.replace('x[0-9]+', ' ')

# Remove all songs without lyrics (e.g. instrumental pieces)
cleaned = cleaned[cleaned['lyrics'].str.strip().str.lower() != 'instrumental']

# Remove any songs with corrupted/non-ASCII characters, unavailable lyrics
cleaned = cleaned[~cleaned['lyrics'].str.contains(r'[^\x00-\x7F]+')]
cleaned = cleaned[cleaned['lyrics'].str.strip() != '']
cleaned = cleaned[cleaned['genre'].str.lower() != 'not available']

#Selecting Pop, Rock, Country, Jazz
cleaned = cleaned.loc[(cleaned['genre'] == 'Pop') |
            (cleaned['genre'] == 'Country') |
            (cleaned['genre'] == 'Rock') |
            (cleaned['genre'] == 'Hip-Hop') |
            (cleaned['genre'] == 'Jazz') ]
cleaned.reset_index(inplace = True)

cleaned
print(len(cleaned))
```

```
185493
```

In [7]:

```
stop = stopwords.words('english')
#removing stop words from lyrics

cleaned['lyrics'] = cleaned['lyrics'].apply(lambda x: ' '.join([word for word in

```

In [8]:

```
#lemmatizing lyrics
import  nltk

w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()

def lemmatize_text(text, flg_lemm=True):
    #Convert string to list (tokenize)
    lst_text = text.split()

    ## Lemmatisation (convert the word into root word)
    if flg_lemm == True:
        lem = nltk.stem.wordnet.WordNetLemmatizer()
        lst_text = [lem.lemmatize(word) for word in lst_text]

    ## back to string from list
    text = " ".join(lst_text)
    return text

#cleaned["lyrics"] = cleaned["lyrics"].apply(lemmatize_text)
```

In [9]:

```
cleaned["lyrics"]  = cleaned["lyrics"].apply(lambda x:  lemmatize_text(x))
```

In [10]:

```
df = cleaned.drop(labels=["level_0", "index","song","year","artist"], axis=1)
```

Splitting Data, One hot Encoding and Text Vectorization

In [11]:

```
from sklearn.model_selection import train_test_split
```

In [12]:

```
df_train, df_test = train_test_split(df, test_size=0.33, random_state=42)
df_train.reset_index()
df_test.reset_index()
```

Out[12]:

	index	genre	lyrics
0	35835	Jazz	I dance ask I dance ask I dance madame My hear...
1	2538	Hip-Hop	Sonic boom head dread cause he's tread Upon Fl...
2	63159	Rock	If I could turn page In time I'd rearrange Jus...
3	6483	Rock	record stop stop skipping equipped stor ear fu...
4	15496	Hip-Hop	Hey yeah ya know I like playersNo Diggity No d...
...	...	...	...
61208	10254	Hip-Hop	We're never done found place belong Don't stan...
61209	31630	Country	It's fake hoax nowhere road one go anywhere an...
61210	107267	Rock	I've spent much time throwing rock window That...
61211	67806	Rock	You're lookin fine long time I still remember ...
61212	23935	Pop	I I get creepin feelin' That might start belie...

61213 rows × 3 columns

In [13]:

```
#train_test split
x_tr = df_train['lyrics'].values
x_val = df_test['lyrics'].values
```

In [14]:

```
def genre_encode(genre):
    """
    return one hot encoding for Y value
    """
    if genre == 'Pop':
        return 0
    elif genre == 'Country':
        return 1
    elif genre == 'Rock':
        return 2
    elif  genre  ==  'Hip-Hop' :
        return 3
    else:
        return 4
```

In [15]:

```
genres = df_train['genre'].tolist()
y_tr  = [ genre_encode ( genre ) for  genre  in  genres ]
y_tr  =  np . array ( y_tr )

genres = df_test['genre'].tolist()
y_val  = [ genre_encode ( genre ) for  genre  in  genres ]
y_val  =  np . array ( y_val )
```

In [16]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
```

In [17]:

```
vectors = vectorizer.fit_transform(x_tr)
```

In [18]:

```
vectors_test = vectorizer.transform(x_val)
```

In [19]:

```
vectors.shape[1]
```

Out[19]:

```
204679
```

In [20]:

```
vectors_test.shape
```

Out[20]:

```
(61213, 204679)
```

```
# MultionomialNB
```

In [21]:

```
from sklearn import metrics
clf = MultinomialNB(alpha=.03)
clf.fit(vectors, y_tr)
```

Out[21]:

```
MultinomialNB(alpha=0.03)
```

In [22]:

```
pred = clf.predict(vectors_test)
```

In [23]:

```
metrics.accuracy_score(y_val, pred)
```

Out[23]:

```
0.6660023197686766
```

```
# RandomForestClassifier
```

In [24]:

```
from  sklearn . set  import  RandomForestClassifier
model=RandomForestClassifier(n_estimators=300)
model=model.fit(vectors,y_tr)
pred_rf = model.predict(vectors_test)
```

In [25]:

```
pred_rf
```

Out[25]:

```
array([4, 3, 2, ..., 2, 2, 2])
```

In [26]:

```
metrics.accuracy_score(y_val, pred_rf)
```

Out[26]:

```
0.691666802803326
```

```
# Neural Network
```

In [27]:

```
from  keras . utils . np_utils  import  to_categorical
```

In [28]:

```
y_tr
```

Out[28]:

```
array([0, 2, 2, ..., 2, 2, 0])
```

In [29]:

```
y_tr  =  to_categorical ( y_tr )
y_val = to_categorical(y_val)
```

In [30]:

```
y_val
```

Out[30]:

```
array([[0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0.],
       ...,
       [0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0.]], dtype=float32)
```

In [31]:

```
vectors.sort_indices()
vectors_test.sort_indices()
```

In [33]:

```
vectors.shape
```

Out[33]:

```
(124280, 204679)
```

In [34]:

```
vectors_test.shape
```

Out[34]:

```
(61213, 204679)
```

In [35]:

```
#deep learning library
from keras.models import *
from keras.layers import *
from keras.callbacks import *
```

In [36]:

```
# Defining the model
model1 = Sequential()
model1.add(Dense(64, input_dim=vectors.shape[1], activation='relu'))
model1.add(Dense(5, activation='softmax'))
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu
```

In [37]:

```
batch_size = 128
# fitting the model
m1 = model1.fit(vectors, y_tr, batch_size=batch_size, epochs=5, validation_data=
```

```
Epoch 1/5

/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/framewor
k/indexed_slices.py:447: UserWarning: Converting sparse IndexedSlices
(IndexedSlices(indices=Tensor("gradient_tape/sequential/dense/embeddin
g_lookup_sparse/Reshape_1:0", shape=(None,), dtype=int32), values=Tens
or("gradient_tape/sequential/dense/embedding_lookup_sparse/Reshape:0",
shape=(None, 64), dtype=float32), dense_shape=Tensor("gradient_tape/se
quential/dense/embedding_lookup_sparse/Cast:0", shape=(2,), dtype=int3
2))) to a dense Tensor of unknown shape. This may consume a large amou
nt of memory.
  warnings.warn(

971/971 [==============================] - 164s 167ms/step - loss: 0.9
155 - accuracy: 0.6637 - val_loss: 0.8088 - val_accuracy: 0.6948
Epoch 2/5
971/971 [==============================] - 162s 167ms/step - loss: 0.6
927 - accuracy: 0.7373 - val_loss: 0.7941 - val_accuracy: 0.7014
Epoch 3/5
971/971 [==============================] - 160s 165ms/step - loss: 0.5
850 - accuracy: 0.7763 - val_loss: 0.8206 - val_accuracy: 0.6983
Epoch 4/5
971/971 [==============================] - 163s 168ms/step - loss: 0.5
047 - accuracy: 0.8085 - val_loss: 0.8611 - val_accuracy: 0.6925
Epoch 5/5
971/971 [==============================] - 165s 169ms/step - loss: 0.4
391 - accuracy: 0.8337 - val_loss: 0.9161 - val_accuracy: 0.6853
```