Solving Spotify Multiclass Genre Classification Problem

Tarak Ram — Published On March 4, 2023 and Last Modified On March 30th, 2023

Beginner Classification Deep Learning Guide Machine Learning Pandas Recommendation Statistics

Introduction

The music industry has become more popular, and how people listen to music is changing like wildfire. The development of music streaming services has increased the demand for automatic music categorization and recommendation systems. Spotify, one of the world's leading music streaming sites, has millions of subscribers and a massive song catalog. Yet, for customers to have a personalized music experience, Spotify must recommend tracks that fit their preferences. Spotify uses machine learning algorithms to guide and categorizes music based on the Genre.

Source: www.analyticsvidhya.com

This project will focus on the Spotify Multiclass Genre Classification problem, where we download the Dataset from Kaggle.

Goal: This project aims to develop a model that classifies the Genre that can accurately predict the Genre of a music track on spotify.

Learning Objectives

- To investigate the link between music genres on Spotify and their acoustic characteristics.
- To create a classification model based on auditory characteristics to predict the genre of a given song.
- To investigate the distribution of various spotify music genres in the dataset.
- To clean and preprocess data in order to prepare it for modeling.
- To assess the categorization model's performance and improve its accuracy.

This article was published as a part of the Data Science Blogathon.

Solving Spotify Multiclass Genre Classification Problem

Prerequisites

Before we begin implementation, we must install and import some of the libraries. The libraries listed below are required:

Pandas: A library for data manipulation and analysis.

NumPy: A scientific computing package used for matrix computations.

Matplotlib: A plotting library for the Python programming language.

Seaborn: A data visualization library based on matplotlib.

Sklearn: A machine learning library for building models for classification

TensorFlow: A popular open-source library for building and training deep learning models.

To install these, we run this command.

```
!pip install pandas
!pip install numpy
!pip install matplotlib
!pip install seaborn
!pip install sklearn
!pip install tensorflow
```

Project Pipeline

Data Preprocessing: Clean and preprocess the “genres_v2” dataset to prepare it for machine learning.

Feature Engineering: Extract meaningful features from the audio files that will help us train our model.

Model Selection: Evaluate several machine learning algorithms to find the best-performing model.

Model Training: Train the selected model on the preprocessed Dataset and evaluate its performance.

Model Deployment: Deploy the trained model in an online application that can recommend music tracks on Spotify based on the user’s preferences

So, let’s get started doing some code.

Project

First, we need to download the data set. You can download the Dataset from Kaggle. We need to import the necessary libraries to perform our tasks.

Solving Spotify Multiclass Genre Classification Problem

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import preprocessing
from sklearn import metrics
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.decomposition import PCA, KernelPCA, TruncatedSVD
from sklearn.manifold import Isomap, TSNE, MDS
import random
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
import warnings
warnings.simplefilter("ignore")
```

Load the Dataset

We load the Dataset using pandas read_csv, and the data set contains 42305 rows and 22 columns and consists of 18000+ tracks.

```
data = pd.read_csv("Desktop/genres_v2.csv")
data
```

```
In [11]: data = pd.read_csv("Desktop/genres_v2.csv")
         data
```

Out[11]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	...	id	
0	0.831	0.814	2	-7.364	1	0.4200	0.059800	0.013400	0.0556	0.3890	...	2Vc6NJ9PW9gD9q343XFRKx	spotify:track:2
1	0.719	0.493	8	-7.230	1	0.0794	0.401000	0.000000	0.1180	0.1240	...	7pgJBLVz5VmnL7uGHmRj6p	spotify:track:3
2	0.850	0.893	5	-4.783	1	0.0623	0.013800	0.000004	0.3720	0.0391	...	0vSWgAlfpye0WCGeNmuNhy	spotify:track:0
3	0.476	0.781	0	-4.710	1	0.1030	0.023700	0.000000	0.1140	0.1750	...	0VSXnJqQkwuH2ei1nOQ1nu	spotify:track:1
4	0.798	0.624	2	-7.668	1	0.2930	0.217000	0.000000	0.1660	0.5910	...	4jCeguq9rMTlbMmPHuO7S3	spotify:track:4
...	...	...	...	...	...	...	...	...	...	...	...	...	
42300	0.528	0.693	4	-5.148	1	0.0304	0.031500	0.000345	0.1210	0.3940	...	46bXU7SgjT104ZoXxzz9tM	spotify:track:4
42301	0.517	0.768	0	-7.922	0	0.0479	0.022500	0.000018	0.2050	0.3830	...	0he2VlGMUO3ajKTxLOfWVT	spotify:track:0
42302	0.361	0.821	8	-3.102	1	0.0505	0.026000	0.000242	0.3850	0.1240	...	72DAt9Lbpy9EUS29OzQLob	spotify:track:7
42303	0.477	0.921	6	-4.777	0	0.0392	0.000551	0.029600	0.0575	0.4680	...	6H0gExFVuE1c3cq9QjFCcU	spotify:track:6
42304	0.529	0.945	9	-5.862	1	0.0615	0.001890	0.000055	0.4140	0.1340	...	6MAAMZlmxcvYhRnxDLTufD	spotify:track:1

42305 rows × 22 columns

Exploring the Data

I use the 'iloc' method to select the rows and columns that form a data frame by their integer index positions. I am choosing the first 20 columns of the df.

```
data.iloc[:,:20]
```

this is for the first 20 columns

Solving Spotify Multiclass Genre Classification Problem

0	0.831	0.814	2	-7.364	1	0.4200	0.059800	0.013400	0.0556	0.3890	156.985	audio_features	2Vc6NJ9PW9gD9q343XFRKx
1	0.719	0.493	8	-7.230	1	0.0794	0.401000	0.000000	0.1180	0.1240	115.080	audio_features	7pgJBLVz5VmnL7uGHmRj6p
2	0.850	0.893	5	-4.783	1	0.0623	0.013800	0.000004	0.3720	0.0391	218.050	audio_features	0vSWgAlfpye0WCGeNmuNhy
3	0.476	0.781	0	-4.710	1	0.1030	0.023700	0.000000	0.1140	0.1750	186.948	audio_features	0VSXnJqQkwuH2ei1nOQ1nu
4	0.798	0.624	2	-7.668	1	0.2930	0.217000	0.000000	0.1660	0.5910	147.988	audio_features	4jCeguq9rMTlbMmPHuO7S3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
42300	0.528	0.693	4	-5.148	1	0.0304	0.031500	0.000345	0.1210	0.3940	150.013	audio_features	46bXU7Sgj7104ZoXxzz9tM
42301	0.517	0.768	0	-7.922	0	0.0479	0.022500	0.000018	0.2050	0.3830	149.928	audio_features	0he2ViGMUO3ajKTxLOfWVT
42302	0.361	0.821	8	-3.102	1	0.0505	0.026000	0.000242	0.3850	0.1240	154.935	audio_features	72DAt9Lbpy9EUS29OzQLob
42303	0.477	0.921	6	-4.777	0	0.0392	0.000551	0.029600	0.0575	0.4880	150.042	audio_features	6HXgExFVuE1c3cq9QjFCcU
42304	0.529	0.945	9	-5.862	1	0.0615	0.001890	0.000055	0.4140	0.1340	155.047	audio_features	6MAAMZImxcvYhRnxDLTufD

42305 rows × 20 columns

```
data.iloc[:,20:]#this is for the 21st column
```

	Unnamed: 0	title
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN
...	...	...
42300	20995.0	Euphoric Hardstyle
42301	20996.0	Greatest Hardstyle Playlist
42302	20997.0	Best of Hardstyle 2020
42303	20998.0	Euphoric Hardstyle
42304	20999.0	Best of Hardstyle 2020

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42305 entries, 0 to 42304
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   danceability      42305 non-null  float64
 1   energy            42305 non-null  float64
 2   key               42305 non-null  int64  
 3   loudness          42305 non-null  float64
 4   mode              42305 non-null  int64  
 5   speechiness       42305 non-null  float64
 6   acousticness      42305 non-null  float64
 7   instrumentalness  42305 non-null  float64
 8   liveness          42305 non-null  float64
 9   valence           42305 non-null  float64
 10  tempo             42305 non-null  float64
 11  type              42305 non-null  object 
 12  id                42305 non-null  object 
 13  uri               42305 non-null  object 
 14  track_href        42305 non-null  object 
 15  analysis_url      42305 non-null  object 
 16  duration_ms       42305 non-null  int64  
 17  time_signature    42305 non-null  int64  
 18  genre             42305 non-null  object 
 19  song_name         21519 non-null  object 
 20  Unnamed: 0        20780 non-null  float64
 21  title             20780 non-null  object 
dtypes: float64(10), int64(4), object(8)
memory usage: 7.1+ MB
```

When you call data.info(), it will print the following information:

- The number of rows and columns in the data frame.
- The name of each column, its data type, and the number of non-null values in that column.
- The total number of non-null values in the data frame.
- The memory usage of the DataFrame.

Solving Spotify Multiclass Genre Classification Problem

```
# number of unique values in our data set.
```

```
danceability          890
energy                917
key                    12
loudness            11654
mode                    2
speechiness          1447
acousticness         4602
instrumentalness     4757
liveness             1695
valence              1674
tempo               15606
type                    1
id                  35877
uri                 35877
track_href          35877
analysis_url        35877
duration_ms         26261
time_signature          4
genre                  15
song_name           15439
Unnamed: 0          20780
title                 132
dtype: int64
```

Data Cleaning

Here, we want to clean our data by removing unnecessary columns that add no value to the prediction.

```
df = data.drop(["type","type","id","uri","track_href","analysis_url","song_name",
                "Unnamed: 0","title", "duration_ms", "time_signature"], axis =1)
df
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	genre
0	0.831	0.814	2	-7.364	1	0.4200	0.059800	0.013400	0.0556	0.3890	156.985	Dark Trap
1	0.719	0.493	8	-7.230	1	0.0794	0.401000	0.000000	0.1180	0.1240	115.080	Dark Trap
2	0.850	0.893	5	-4.783	1	0.0623	0.013800	0.000004	0.3720	0.0391	218.050	Dark Trap
3	0.476	0.781	0	-4.710	1	0.1030	0.023700	0.000000	0.1140	0.1750	186.948	Dark Trap
4	0.798	0.624	2	-7.668	1	0.2930	0.217000	0.000000	0.1660	0.5910	147.988	Dark Trap
...	...	...	...	...	...	...	...	...	...	...	...	...
42300	0.528	0.693	4	-5.148	1	0.0304	0.031500	0.000345	0.1210	0.3940	150.013	hardstyle
42301	0.517	0.768	0	-7.922	0	0.0479	0.022500	0.000018	0.2050	0.3830	149.928	hardstyle
42302	0.361	0.821	8	-3.102	1	0.0505	0.026000	0.000242	0.3850	0.1240	154.935	hardstyle
42303	0.477	0.921	6	-4.777	0	0.0392	0.000551	0.029600	0.0575	0.4880	150.042	hardstyle
42304	0.529	0.945	9	-5.862	1	0.0615	0.001890	0.000055	0.4140	0.1340	155.047	hardstyle

42305 rows × 12 columns

We have removed some columns that add no value to this particular problem and put axis = 1, where it drops the columns rather than rows. We are again calling the Data Frame to see the new Data Frame with helpful information.

The df. describe( ) method generates descriptive statistics of a Pandas Data Frame. It provides a summary of the central tendency and dispersion and the shape of the distribution of a dataset.

After running this command, you can see all the descriptive statistics of the Data Frame, like std, mean, median, percentile, min, and max.

```
df.describe()
```

Solving Spotify Multiclass Genre Classification Problem

mean	0.639364	0.762516	5.370240	-6.465442	0.549462	0.136561	0.096160	0.283048	0.214079	0.357101	147.474056
std	0.156617	0.183823	3.666145	2.941165	0.497553	0.126168	0.170827	0.370791	0.175576	0.233200	23.844623
min	0.065100	0.000243	0.000000	-33.357000	0.000000	0.022700	0.000001	0.000000	0.010700	0.018700	57.967000
25%	0.524000	0.632000	1.000000	-8.161000	0.000000	0.049100	0.001730	0.000000	0.099600	0.161000	129.931000
50%	0.646000	0.803000	6.000000	-6.234000	1.000000	0.075500	0.016400	0.005940	0.135000	0.322000	144.973000
75%	0.766000	0.923000	9.000000	-4.513000	1.000000	0.193000	0.107000	0.722000	0.294000	0.522000	161.464000
max	0.988000	1.000000	11.000000	3.148000	1.000000	0.946000	0.988000	0.989000	0.988000	0.988000	220.290000

To display a summary of a Pandas DataFrame or Series, use the df.info() function. It gives Dataset information like the number of rows and columns, the data types of each column, the number of non-null values in each column, and the utilization of the dataset's memory.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42305 entries, 0 to 42304
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   danceability      42305 non-null  float64
 1   energy            42305 non-null  float64
 2   key               42305 non-null  int64
 3   loudness          42305 non-null  float64
 4   mode              42305 non-null  int64
 5   speechiness       42305 non-null  float64
 6   acousticness      42305 non-null  float64
 7   instrumentalness  42305 non-null  float64
 8   liveness          42305 non-null  float64
 9   valence           42305 non-null  float64
 10  tempo             42305 non-null  float64
 11  genre             42305 non-null  object
dtypes: float64(9), int64(2), object(1)
memory usage: 3.9+ MB
```

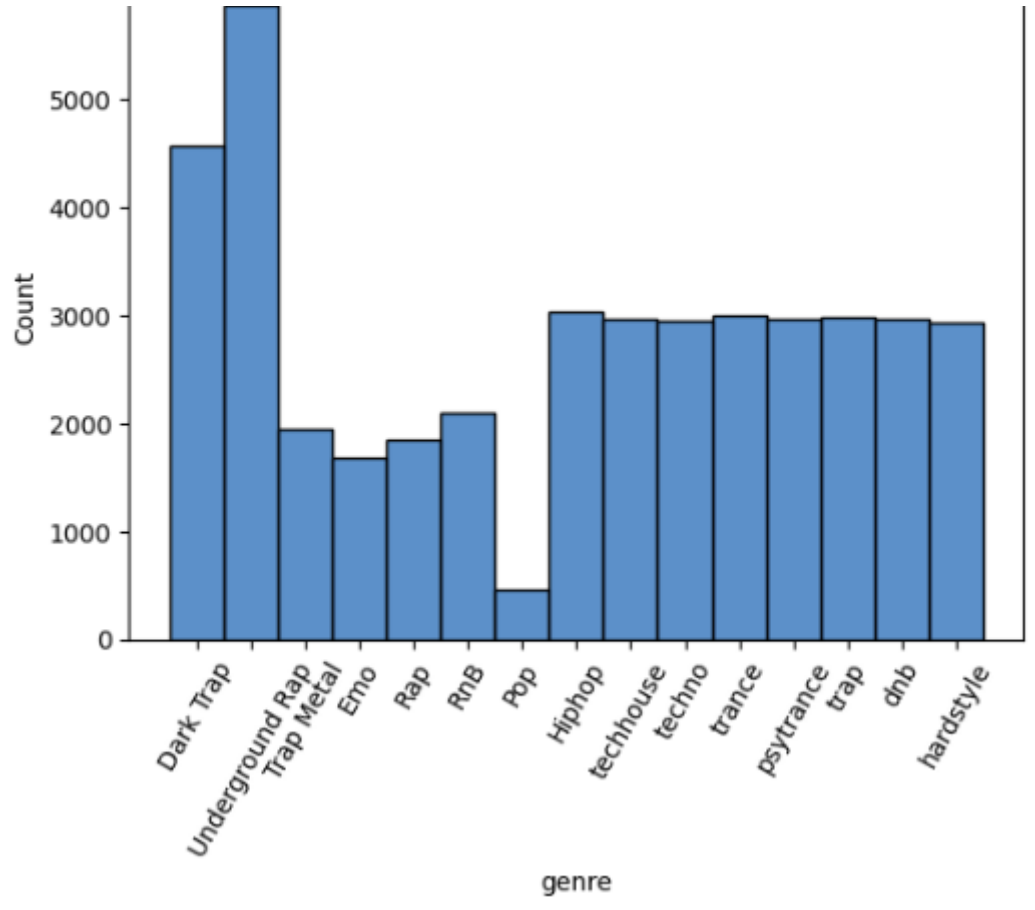
```
df["genre"].value_counts()
```

```
Underground Rap    5875
Dark Trap          4578
Hiphop             3028
trance             2999
trap               2987
techhouse          2975
dnb                2966
psytrance          2961
techno             2956
hardstyle          2936
RnB                2099
Trap Metal         1956
Rap                1848
Emo                1680
Pop                 461
Name: genre, dtype: int64
```

axe = sns.histplot(df[“genre”]) generates a histogram of the distribution of values in a Pandas DataFrame named df’s “genre” column. This code may be used to visualize the frequency of some Spotify genres in a music dataset.

```
ax = sns.histplot(df["genre"])
_ = plt.xticks(rotation=60)
_ = plt.title("Genres")
```

Solving Spotify Multiclass Genre Classification Problem



The following code eliminates or deletes all rows in a Pandas DataFrame where the value in the “genre” column is equal to “Pop”. The DataFrame’s index is then reset to the range where it starts with 0. Lastly, it computes the correlation matrix of the DataFrame’s remaining columns.

This code helps study a dataset by deleting unnecessary rows and finding correlations between the remaining variables.

```
df.drop(df.loc[df['genre']=="Pop"].index, inplace=True)
df = df.reset_index(drop = True)
df.corr()
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
danceability	1.000000	-0.324608	-0.013133	-0.218870	0.085789	0.184211	0.070473	-0.065724	-0.196866	0.369169	-0.167563
energy	-0.324608	1.000000	0.045013	0.603140	-0.031562	-0.150872	-0.495804	0.303650	0.231560	-0.013813	-0.023276
key	-0.013133	0.045013	1.000000	-0.006947	-0.250188	-0.031088	-0.004237	0.068760	0.003343	0.029533	-0.010681
loudness	-0.218870	0.603140	-0.006947	1.000000	-0.004879	0.050098	-0.284931	-0.186337	0.167971	0.076700	0.153185
mode	0.085789	-0.031562	-0.250188	-0.004879	1.000000	0.050455	-0.015634	-0.016375	0.008253	0.023070	-0.011819
speechiness	0.184211	-0.150872	-0.031088	0.050098	0.050455	1.000000	0.163345	-0.392013	0.056733	0.224940	0.166511
acousticness	0.070473	-0.495804	-0.004237	-0.284931	-0.015634	0.163345	1.000000	-0.262410	-0.106497	0.101200	0.054707
instrumentalness	-0.065724	0.303650	0.068760	-0.186337	-0.016375	-0.392013	-0.262410	1.000000	-0.017781	-0.252966	-0.208680
liveness	-0.196866	0.231560	0.003343	0.167971	0.008253	0.056733	-0.106497	-0.017781	1.000000	-0.023217	0.029513
valence	0.369169	-0.013813	0.029533	0.076700	0.023070	0.224940	0.101200	-0.252966	-0.023217	1.000000	0.058152
tempo	-0.167563	-0.023276	-0.010681	0.153185	-0.011819	0.166511	0.054707	-0.208680	0.029513	0.058152	1.000000

The following code sns. heatmap (df, cmap=’coolwarm, annot=True) plt. show() generates a heatmap depicting a Pandas DataFrame df’s correlation matrix.

This code helps to find and display the strength and direction of correlations between variables in a dataset. The heatmap color coding makes it easy to see which pairs of variables are highly correlated and which are not.

```
plt.subplots(figsize=(10,9))
sns.heatmap(df.corr(), cmap='coolwarm', annot=True)
plt.show()
```

The following code picks a subset of columns in a Pandas DataFrame df named x, which contains all columns from the DataFrame’s beginning, including the “tempo” column. Then it chooses the DataFrame’s “genre” as the target variable and assigns it to y.

Solving Spotify Multiclass Genre Classification Problem

Series with the “genre” column values.

The methods x.unique() and y.unique() retrieve the unique values in the x and y variables, respectively. These routines can be helpful for determining the number of unique values in the variables of a dataset.

```
x = df.loc[:,:"tempo"]
y = df["genre"]
```

x

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
0	0.831	0.814	2	-7.364	1	0.4200	0.059800	0.013400	0.0556	0.3890	156.985
1	0.719	0.493	8	-7.230	1	0.0794	0.401000	0.000000	0.1180	0.1240	115.080
2	0.850	0.893	5	-4.783	1	0.0623	0.013800	0.000004	0.3720	0.0391	218.050
3	0.476	0.781	0	-4.710	1	0.1030	0.023700	0.000000	0.1140	0.1750	186.948
4	0.798	0.624	2	-7.668	1	0.2930	0.217000	0.000000	0.1660	0.5910	147.988
...	...	...	...	...	...	...	...	...	...	...	...
41839	0.528	0.693	4	-5.148	1	0.0304	0.031500	0.000345	0.1210	0.3940	150.013
41840	0.517	0.768	0	-7.922	0	0.0479	0.022500	0.000018	0.2050	0.3830	149.928
41841	0.361	0.821	8	-3.102	1	0.0505	0.026000	0.000242	0.3850	0.1240	154.935
41842	0.477	0.921	6	-4.777	0	0.0392	0.000551	0.029600	0.0575	0.4880	150.042
41843	0.529	0.945	9	-5.862	1	0.0615	0.001890	0.000055	0.4140	0.1340	155.047

41844 rows × 11 columns

```
y
```

```
0        Dark Trap
1        Dark Trap
2        Dark Trap
3        Dark Trap
4        Dark Trap
           ...
41839    hardstyle
41840    hardstyle
41841    hardstyle
41842    hardstyle
41843    hardstyle
Name: genre, Length: 41844, dtype: object
```

```
y.unique()
```

```
array(['Dark Trap', 'Underground Rap', 'Trap Metal', 'Emo', 'Rap', 'RnB',
       'Hiphop', 'techhouse', 'techno', 'trance', 'psytrance', 'trap',
       'dnb', 'hardstyle'], dtype=object)
```

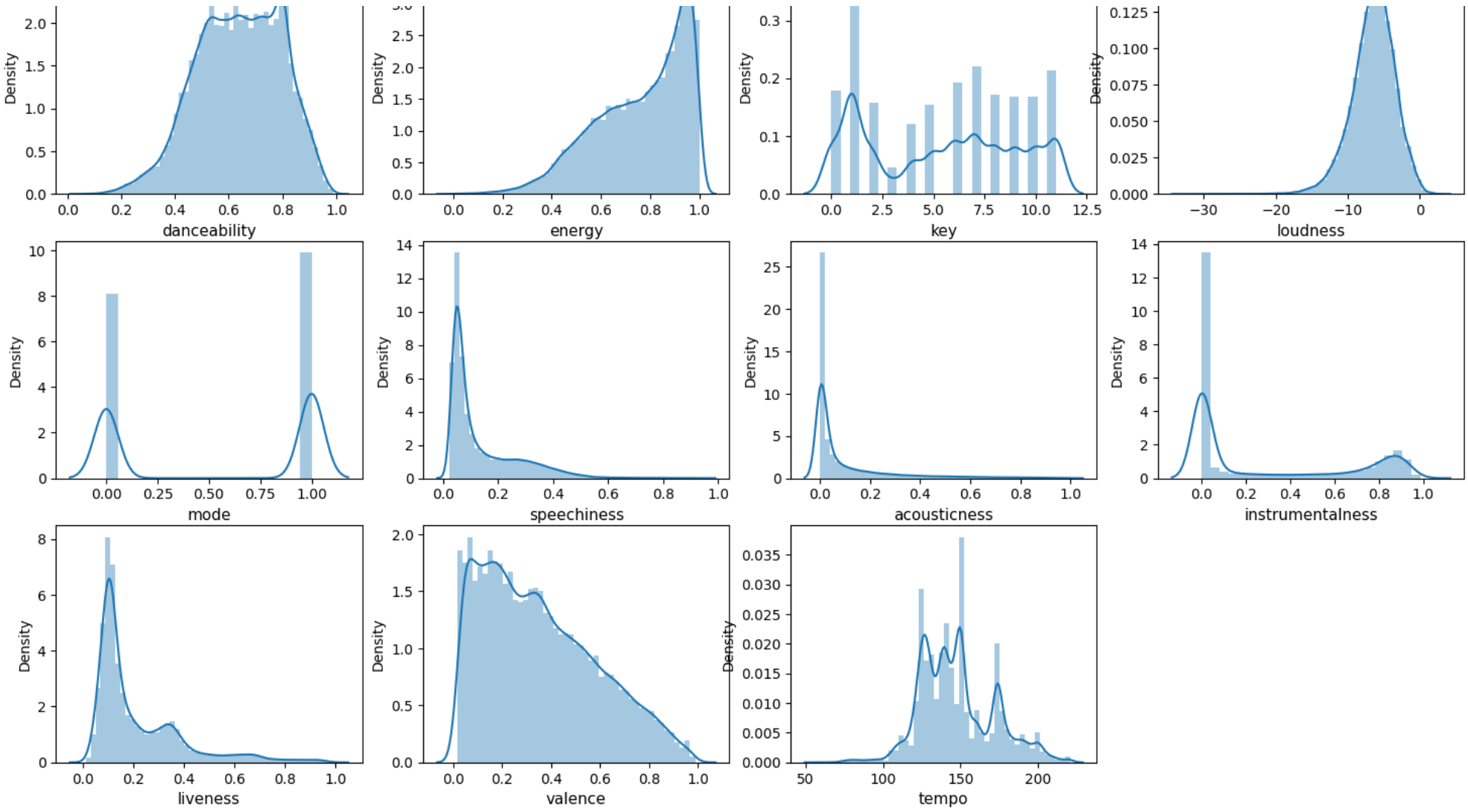
I am not giving all the images. You can check the notebook down below.

The given code generates a grid of distribution plots that allow users to view the distribution of values over several columns in a dataset. Discovering patterns, trends, and outliers in the data by showing the distribution of values in each column. These are helpful and beneficial for exploratory data analysis and finding valuable and potential faults or inaccuracies in a dataset.

```
k=0
plt.figure(figsize = (18,14))
for i in x.columns:
    plt.subplot(4,4, k + 1)
    sns.distplot(x[i])
    plt.xlabel(i, fontsize=11)
    k +=1
```

Here, we are plotting for each x_columns, by using the for loop.

Solving Spotify Multiclass Genre Classification Problem



Model Training

The following code divides a dataset into training and testing subsets. It divides the input variables and target variables into 80% training and 20% testing groups at random. The descriptive statistics of the training data are then outputted to aid in data exploration and the identification of possible problems.

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y,

test_size= 0.2, random_state=42, shuffle = True)


xtrain.columns
```

```
Index(['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
       'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo'],
      dtype='object')
```

```
xtrain.describe()
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
count	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000
mean	0.638809	0.763796	5.382972	-6.467526	0.548559	0.137127	0.095507	0.287004	0.214505	0.355343	147.398774
std	0.156886	0.183906	3.666728	2.943751	0.497644	0.126734	0.170971	0.371588	0.176067	0.232911	23.689341
min	0.065100	0.000243	0.000000	-33.357000	0.000000	0.023200	0.000001	0.000000	0.010700	0.018700	61.309000
25%	0.523000	0.633000	1.000000	-8.171000	0.000000	0.049250	0.001680	0.000000	0.099600	0.160000	129.962000
50%	0.645000	0.806000	6.000000	-6.251000	1.000000	0.075700	0.016000	0.007240	0.135000	0.319000	144.976000
75%	0.766000	0.923000	9.000000	-4.510000	1.000000	0.195000	0.105000	0.729000	0.295000	0.520000	160.968500
max	0.988000	1.000000	11.000000	3.108000	1.000000	0.944000	0.986000	0.989000	0.981000	0.988000	220.290000

Here we are splitting the data into training and testing (size = 20%), and we are using the describe function to see the descriptive statistics.

The MinMaxScaler() function from the sklearn.preprocessing module is used to do feature scaling. It stores the training data's column names in the variable ol. The scaler object is then used to fit and convert the xtrain data while changing xtest data.

Solving Spotify Multiclass Genre Classification Problem

a critical step in the preprocessing and standardizing of data for machine learning models.

```
col = xtrain.columns
scalerx = MinMaxScaler()
xtrain = scalerx.fit_transform(xtrain)
xtest = scalerx.transform(xtest)
xtrain = pd.DataFrame(xtrain, columns = col)
xtest = pd.DataFrame(xtest, columns = col)
```

Here we use the MinMaxScaler, mainly for scaling and normalizing the data.

The following allows us to see the descriptive statistics of the xtrain and xtest.

```
xtrain.describe()
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
count	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000	33475.000000
mean	0.621638	0.763738	0.489361	0.737405	0.548559	0.123726	0.096862	0.290196	0.210043	0.347305	0.541510
std	0.169993	0.183950	0.333339	0.080728	0.497644	0.137635	0.173399	0.375721	0.181456	0.240288	0.149007
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.496153	0.632911	0.090909	0.690690	0.000000	0.028291	0.001703	0.000000	0.091621	0.145775	0.431831
50%	0.628345	0.805953	0.545455	0.743343	1.000000	0.057016	0.016226	0.007321	0.128105	0.309811	0.526270
75%	0.759454	0.922981	0.818182	0.791087	1.000000	0.186577	0.106490	0.737108	0.293002	0.517177	0.626864
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

```
xtest.describe()
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
count	8369.000000	8369.000000	8369.000000	8369.000000	8369.000000	8369.000000	8.369000e+03	8369.000000	8369.000000	8369.000000	8369.000000
mean	0.622409	0.760911	0.483429	0.736432	0.553113	0.123533	9.721151e-02	0.285188	0.209900	0.345398	0.542275
std	0.170245	0.184347	0.333178	0.081439	0.497201	0.136224	1.705395e-01	0.375950	0.180522	0.238152	0.150267
min	0.000000	0.000000	0.000000	0.198656	0.000000	-0.000543	7.505079e-07	0.000000	0.001443	0.000000	-0.021021
25%	0.497237	0.628910	0.090909	0.690251	0.000000	0.028345	1.783907e-03	0.000000	0.092033	0.144744	0.431303
50%	0.628345	0.799951	0.545455	0.743754	1.000000	0.057450	1.612469e-02	0.005561	0.128105	0.309811	0.526201
75%	0.761621	0.922981	0.727273	0.789908	1.000000	0.186577	1.095325e-01	0.725986	0.295063	0.513051	0.626685
max	0.993499	1.000000	1.000000	1.001097	1.000000	1.002172	1.002028e+00	0.995956	1.007214	0.990715	0.999535

The LabelEncoder() function from the sklearn.preprocessing package is used to encode labels. It uses the fit transform() and transform() routines to encode the category target variables (ytrain and ytest) into numerical values.

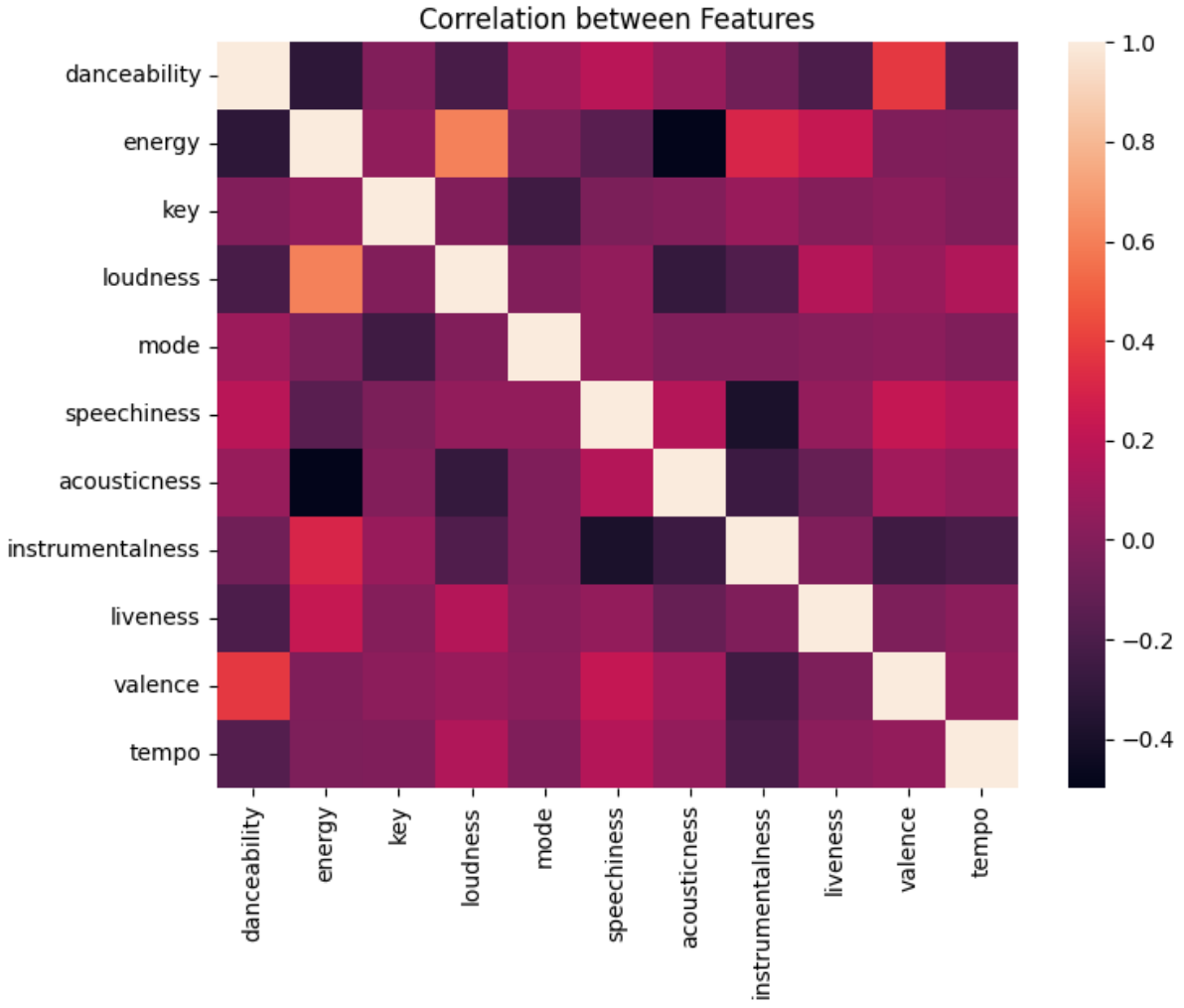
The training and testing data for input (x) and target (y) variables are then concatenated. The numerical labels are then inversely transformed into their original category values (y train, y test, and y org).

Next, we use the np.unique() method, which returns the individual categories in the training data.

Lastly, using the seaborn library generates a heatmap graphic to illustrate the relationship between the input characteristics. This is a critical stage when we examine and prepare data for machine-learning models.

Solving Spotify Multiclass Genre Classification Problem

```
ytrain = le.fit_transform(ytrain)

ytest = le.transform(ytest)

x = pd.concat([xtrain, xtest], axis = 0)

y = pd.concat([pd.DataFrame(ytrain), pd.DataFrame(ytest)], axis = 0)

y_train = le.inverse_transform(ytrain)

y_test = le.inverse_transform(ytest)

y_org = pd.concat([pd.DataFrame(y_train), pd.DataFrame(y_test)], axis = 0)

np.unique(y_train)

plt.subplots(figsize=(8,6))

ax = sns.heatmap(xtrain.corr()).set(title = "Correlation between Features")
```

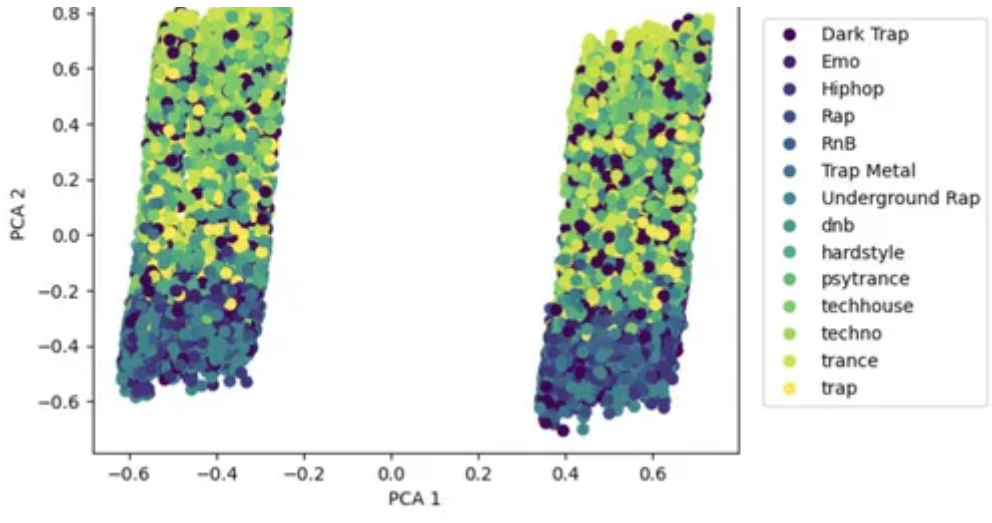


PCA is a popular dimensionality reduction approach that may assist in decreasing the complexity of large datasets and increasing the performance of machine learning models.

With input data x, the algorithm uses PCA to minimize the number of features to two parts that explain the variation. The reduced Dataset is shown on a 2D scatter plot, with dots colored by class labels in y. This aids in visualizing the dividing of some classes in the reduced feature space.

```
pca = PCA(n_components=2)
x_pca = pca.fit_transform(x, y)
plot_pca = plt.scatter(x_pca[:,0], x_pca[:,1], c=y)
handles, labels = plot_pca.legend_elements()
lg = plt.legend(handles, list(np.unique(y_org)), loc = 'center right', bbox_to_anchor=(1.4, 0.5))
plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
_ = plt.title("PCA")
```

Solving Spotify Multiclass Genre Classification Problem

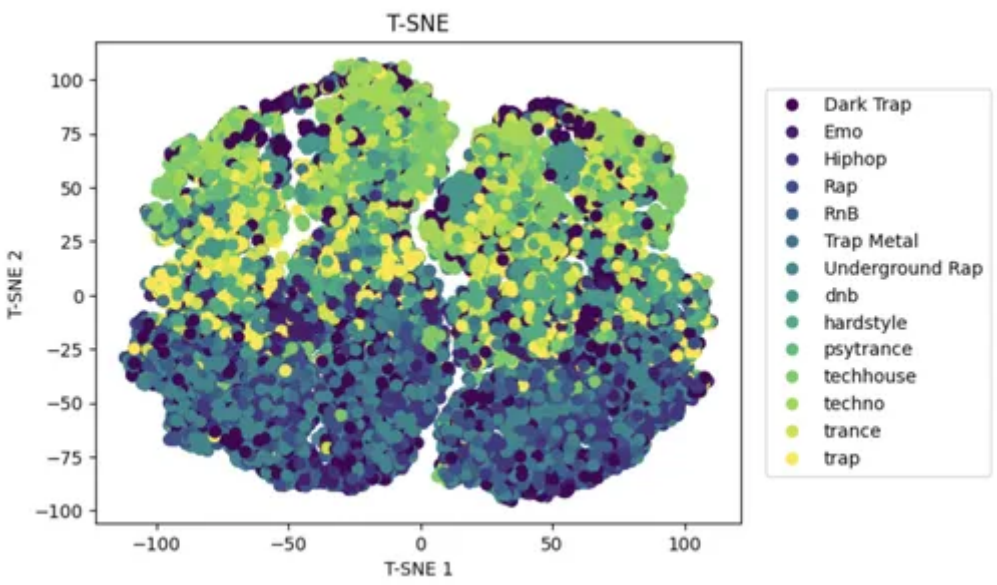


t-SNE is a popular nonlinear dimensionality reduction approach that may assist in decreasing the complexity of large datasets and improve the performance of machine learning models.

Using t-Distributed Stochastic Neighbor Embedding (t-SNE) on the input data x reduces the number of features in the high-dimensional space to 2D while maintaining similarity between Data points.

A 2D scatter plot shows the reduced Dataset, with dots colored according to their y-class labels. It helps visualize the division of some classes in the reduced feature space.

```
tsne = TSNE(n_components=2)
x_tsne = tsne.fit_transform(x, y)
plot_tsne = plt.scatter(x_tsne[:,0], x_tsne[:,1], c=y)
handles, labels = plot_tsne.legend_elements()
lg = plt.legend(handles, list(np.unique(y_org)), loc = 'center right', bbox_to_anchor=(1.4, 0.5))
plt.xlabel("T-SNE 1")
plt.ylabel("T-SNE 2")
_ = plt.title("T-SNE")
```



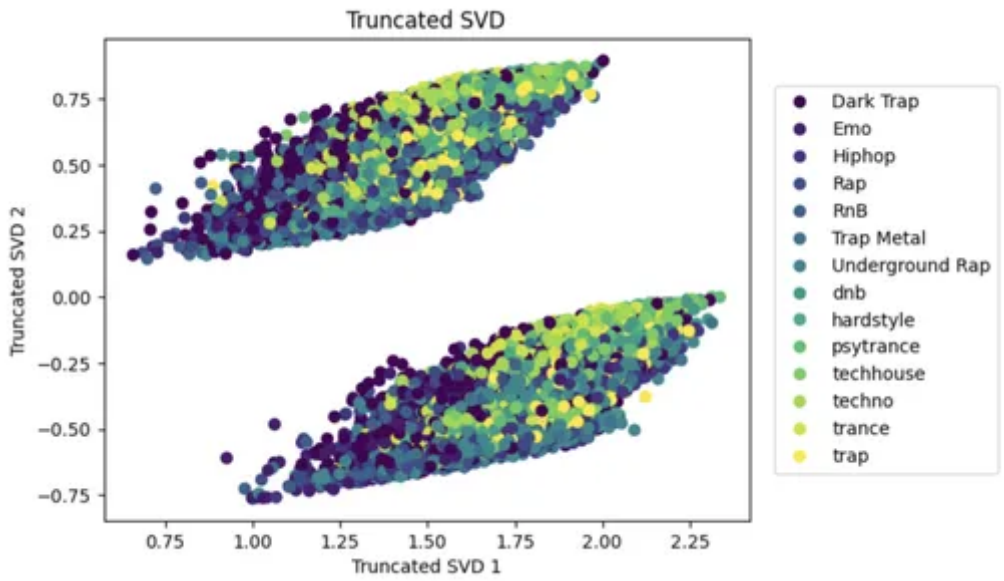
SVD is a popular dimensionality reduction approach that may assist in decreasing the complexity of large datasets and increasing the performance of machine learning models.

The following code applies Singular Value Decomposition (SVD) on the input data x with n components=2, reducing the number of input features to two that explain the most variance in the data. The reduced Dataset is then shown on a 2D scatter plot, with the dots colored based on their y-class labels.

This facilitates visualizing the division of multiple classes in the reduced feature space, and the scatter plot is made with the matplotlib tool.

Solving Spotify Multiclass Genre Classification Problem

```
x_svd = svd.fit_transform(x, y)
plot_svd = plt.scatter(x_svd[:,0], x_svd[:,1], c=y)
handles, labels = plot_svd.legend_elements()
lg = plt.legend(handles, list(np.unique(y_org)), loc = 'center right', bbox_to_anchor=(1.4, 0.5))
plt.xlabel("Truncated SVD 1")
plt.ylabel("Truncated SVD 2")
_ = plt.title("Truncated SVD")
```

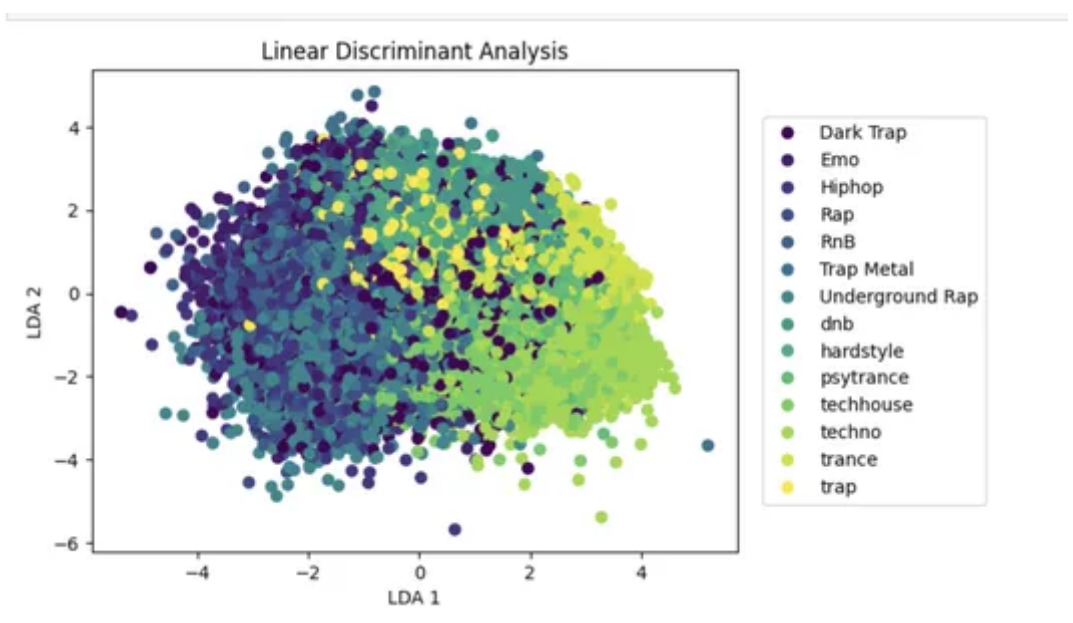


LDA is a popular dimensionality reduction approach that can increase machine learning model performance by decreasing the influence of irrelevant information.

The following code does Linear Discriminant Analysis (LDA) on the input data x with n components=2, which reduces the number of input features to two linear discriminants that maximize the division between the different classes in the data.

The reduced Dataset is then shown on a 2D scatter plot, with the dots colored based on their y-class labels. This aids in visualizing the division of some classes in the reduced feature space.

```
lda = LinearDiscriminantAnalysis(n_components=2)
x_lda = lda.fit_transform(x, y.values.ravel())
plot_lda = plt.scatter(x_lda[:,0], x_lda[:,1], c=y)
handles, labels = plot_lda.legend_elements()
lg = plt.legend(handles, list(np.unique(y_org)), loc = 'center right', bbox_to_anchor=(1.4, 0.5))
plt.xlabel("LDA 1")
plt.ylabel("LDA 2")
_ = plt.title("Linear Discriminant Analysis")
```



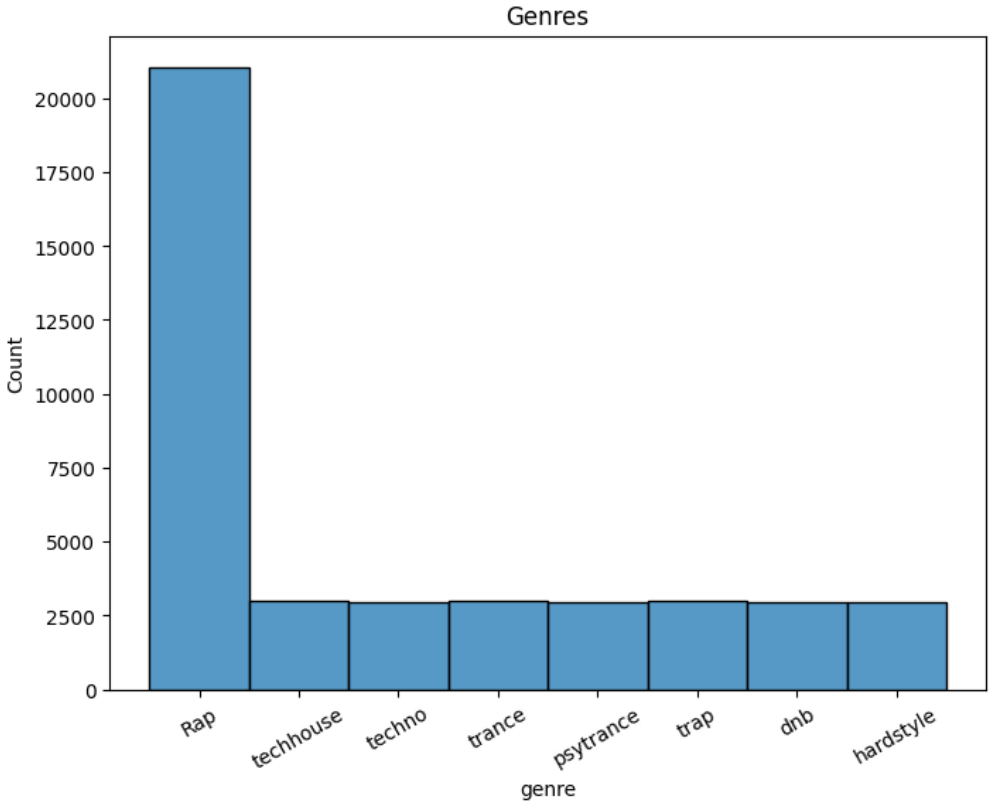
The following code substitutes some values in a Data Frame column called ‘genre’ with the new deal ‘Rap.’ Specifically, it replaces the values “Trap Metal,” “Underground Rap,” “Emo,” “RnB,” etc., with “Rap.” This is useful for grouping genres under a single name for analysis or modeling.

Solving Spotify Multiclass Genre Classification Problem

```
df = df.replace("Underground Rap", "Rap")
df = df.replace("Emo", "Rap")
df = df.replace("RnB", "Rap")
df = df.replace("Hiphop", "Rap")
df = df.replace("Dark Trap", "Rap")
```

The code below generates a histogram using the seaborn library to illustrate the variable “genre” distribution in the input dataset df. The figure has been rotated by 30 degrees to improve the visibility of the x-axis labels. “Genres” is a title.

```
plt.subplots(figsize=(8,6))
ax = sns.histplot(df["genre"])
_ = plt.xticks(rotation=30)
_ = plt.title('Genres')
```



The provided code removes the rows from the Data Frame. Specifically, it eliminates rows with a frequency of 0.85 where the genre column value is “Rap,” using a random number generator.

The rows to be discarded are saved in a list of rows dropped before being removed from the Data Frame using the drop function. The code then prints a histogram of the remaining genre values with the seaborn plot function and changes the title and rotation of the x-axis labels with matplotlib’s title and xticks methods.

```
rows_drop = []

for i in range(len(df)):
  if df.iloc[i]['genre'] == 'Rap':
    if random.random()<0.85:
      rows_drop.append(i)
df.drop(index = rows_drop, inplace=True)

ax = sns.histplot(df["genre"])
_ = plt.xticks(rotation=30)
_ = plt.title("Genres")
```

The code provided preprocesses the data. The first step is to divide the input data into training and testing sets using the Sklearn library’s train test split function.

Solving Spotify Multiclass Genre Classification Problem

code encodes the category target variable using the preprocessing module's LabelEncoder function.

As a result, the training and testing sets are preprocessed previously are merged into a single dataset that the machine learning algorithm can process.

```
x = df.loc[:,:"tempo"]
y = df["genre"]

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size= 0.2,
                                        random_state=42, shuffle = True)

col = xtrain.columns
scalerx = MinMaxScaler()

xtrain = scalerx.fit_transform(xtrain)
xtest = scalerx.transform(xtest)

xtrain = pd.DataFrame(xtrain, columns = col)
xtest = pd.DataFrame(xtest, columns = col)
le = preprocessing.LabelEncoder()
ytrain = le.fit_transform(ytrain)
ytest = le.transform(ytest)

x = pd.concat([xtrain, xtest], axis = 0)
y = pd.concat([pd.DataFrame(ytrain), pd.DataFrame(ytest)], axis = 0)

y_train = le.inverse_transform(ytrain)
y_test = le.inverse_transform(ytest)
y_org = pd.concat([pd.DataFrame(y_train), pd.DataFrame(y_test)], axis = 0)
```

This code creates two early stopping callbacks for model training, one based on validation loss and the other on validation accuracy. Keras' Sequential API makes a NN model with various connected layers using the ReLU activation function, batch normalization, and dropout regularisation. The summary of the model is printed on the console.

The final output layer outputs class probabilities using the softmax activation function. The summary of the model is printed on the console.

```
early_stopping1 = keras.callbacks.EarlyStopping(monitor = "val_loss",
                                                patience = 10, restore_best_weights = True)
early_stopping2 = keras.callbacks.EarlyStopping(monitor = "val_accuracy",
                                                patience = 10, restore_best_weights = True)

model = keras.Sequential([
    keras.layers.Input(name = "input", shape = (xtrain.shape[1])),
    keras.layers.Dense(256, activation = "relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(128, activation = "relu"),
    keras.layers.Dense(128, activation = "relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation = "relu"),
    keras.layers.Dense(max(ytrain)+1, activation = "softmax")
])

model.summary()
```

Solving Spotify Multiclass Genre Classification Problem

```
 dense (Dense)               (None, 256)               3072

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 128)               16512

 batch_normalization_1 (Batc  (None, 128)              512
 hNormalization)

 dropout_1 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 64)                8256

 dense_4 (Dense)             (None, 8)                 520

=================================================================
Total params: 62,792
Trainable params: 62,024
Non-trainable params: 768
```

The following code block uses Keras to compile and train a neural network model. The model is a sequential model with multiple dense layers with relu activation function, batch normalization, and dropout regularisation. “sparse categorical cross entropy” is the loss function utilized. At the same time, “Adam” is the optimizer. The model is trained for 100 epochs, with callbacks that end early based on validation loss and accuracy.

```
model.compile(optimizer = keras.optimizers.Adam(),
            loss = "sparse_categorical_crossentropy",
            metrics = ["accuracy"])

model_history = model.fit(xtrain, ytrain,
                epochs = 100,
                verbose = 1, batch_size = 128,
                validation_data = (xtest, ytest),
                callbacks = [early_stopping1, early_stopping2])
```

```
Epoch 1/100
150/150 [==============================] - 10s 20ms/step - loss: 0.9957 - accuracy: 0.6468 - val_loss: 1.8709 - val_accuracy: 0.3881
Epoch 2/100
150/150 [==============================] - 1s 9ms/step - loss: 0.5867 - accuracy: 0.7884 - val_loss: 2.1501 - val_accuracy: 0.3218
Epoch 3/100
150/150 [==============================] - 1s 10ms/step - loss: 0.5279 - accuracy: 0.8086 - val_loss: 1.0925 - val_accuracy: 0.5787
Epoch 4/100
150/150 [==============================] - 2s 16ms/step - loss: 0.4920 - accuracy: 0.8202 - val_loss: 0.5769 - val_accuracy: 0.7917
Epoch 5/100
150/150 [==============================] - 2s 14ms/step - loss: 0.4718 - accuracy: 0.8279 - val_loss: 0.4505 - val_accuracy: 0.8407
Epoch 6/100
150/150 [==============================] - 1s 9ms/step - loss: 0.4649 - accuracy: 0.8317 - val_loss: 0.5100 - val_accuracy: 0.8231
Epoch 7/100
150/150 [==============================] - 1s 9ms/step - loss: 0.4432 - accuracy: 0.8345 - val_loss: 0.4329 - val_accuracy: 0.8438
Epoch 8/100
150/150 [==============================] - 1s 9ms/step - loss: 0.4450 - accuracy: 0.8371 - val_loss: 0.4574 - val_accuracy: 0.8342
Epoch 9/100
150/150 [==============================] - 1s 9ms/step - loss: 0.4317 - accuracy: 0.8426 - val_loss: 0.4788 - val_accuracy: 0.8321
Epoch 10/100
150/150 [==============================] - 1s 9ms/step - loss: 0.4335 - accuracy: 0.8416 - val_loss: 0.4036 - val_accuracy: 0.8521
Epoch 11/100
150/150 [==============================] - 1s 9ms/step - loss: 0.4273 - accuracy: 0.8443 - val_loss: 0.4399 - val_accuracy: 0.8430
Epoch 12/100
150/150 [==============================] - 1s 9ms/step - loss: 0.4277 - accuracy: 0.8438 - val_loss: 0.4112 - val_accuracy: 0.8526
Epoch 13/100
150/150 [==============================] - 2s 14ms/step - loss: 0.4133 - accuracy: 0.8489 - val_loss: 0.4325 - val_accuracy: 0.8450
Epoch 14/100
150/150 [==============================] - 2s 14ms/step - loss: 0.4125 - accuracy: 0.8484 - val_loss: 0.4279 - val_accuracy: 0.8507
```

The training data is sent as xtrain and ytrain, whereas the validation data is sent as xtest and ytest. The training history of the model is saved in the model history variable.

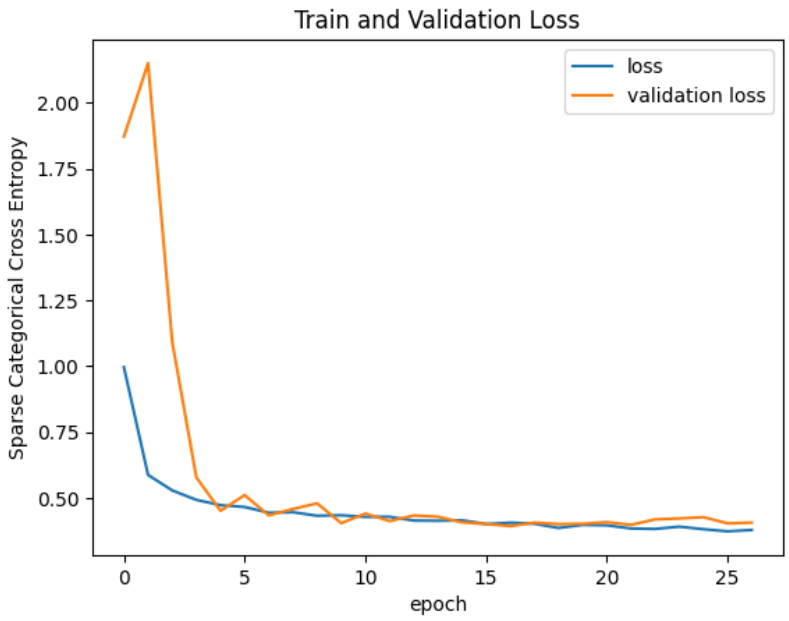
```
print(model.evaluate(xtrain, ytrain))
 print(model.evaluate(xtest, ytest))
```

```
600/600 [==============================] - 2s 4ms/step - loss: 0.3311 - accuracy: 0.8786
[0.3310985267162323, 0.8786109089851379]
150/150 [==============================] - 1s 3ms/step - loss: 0.3923 - accuracy: 0.8569
[0.3922579288482666, 0.8569343090057373]
```

The following code generates a plot using matplotlib; on the x_axis, we have the epoch, and on the y_axis, we have the sparse Categorical Cross Entropy.

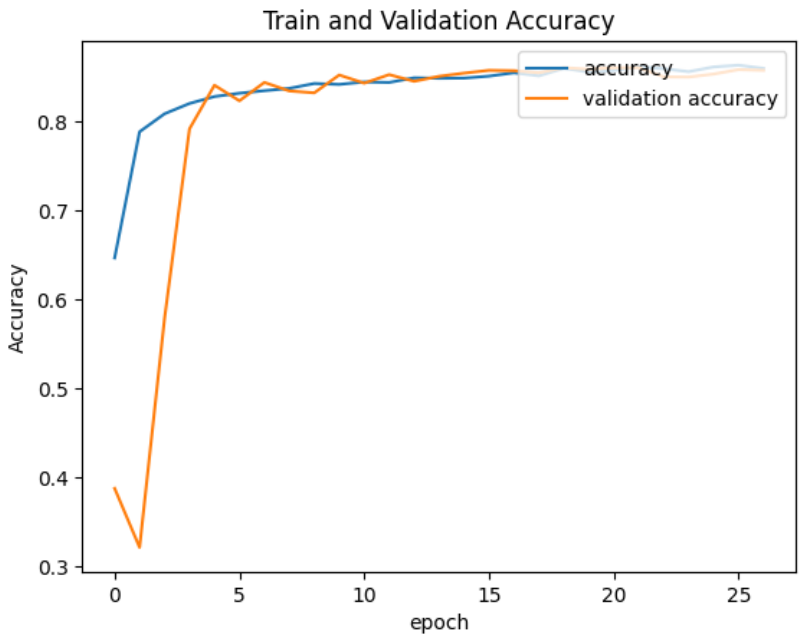
Solving Spotify Multiclass Genre Classification Problem

```
plt.plot(model_history.history["val_loss"])
plt.legend(["loss", "validation loss"], loc ="upper right")
plt.title("Train and Validation Loss")
plt.xlabel("epoch")
plt.ylabel("Sparse Categorical Cross Entropy")
plt.show()
```



Same as above, but here we are plotting between the epoch and the accuracy.

```
plt.plot(model_history.history["accuracy"])
plt.plot(model_history.history["val_accuracy"])
plt.legend(["accuracy", "validation accuracy"], loc ="upper right")
plt.title("Train and Validation Accuracy")
plt.xlabel("epoch")
plt.ylabel("Accuracy")
plt.show()
```



The following code ypred, which predicts the xtest.

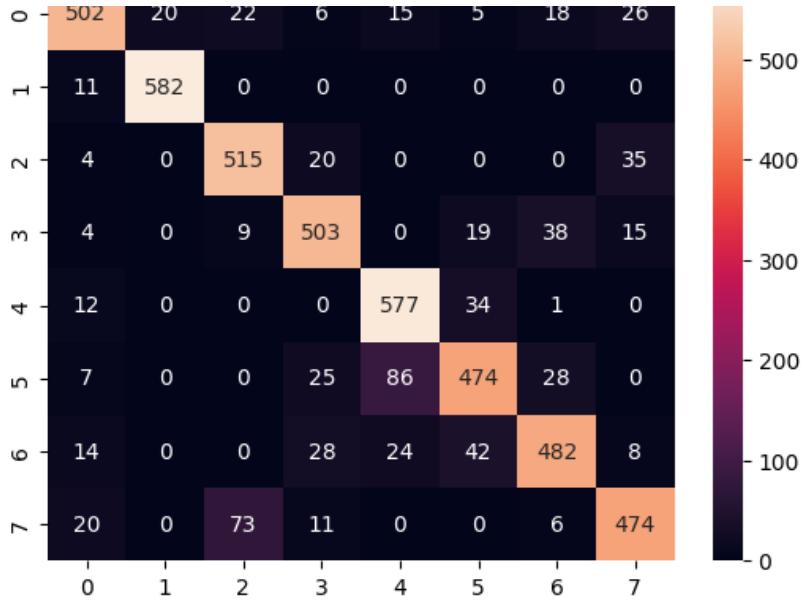
```
ypred = model.predict(xtest).argmax(axis=1)
```

```
150/150 [==============================] - 1s 4ms/step
```

The following code evaluates the classification metrics on the test and ypred, where we can see the precision, recall, and F1score. Based on the values, we can proceed with our model.

```
cf_matrix = metrics.confusion_matrix(ytest, ypred)
_ = sns.heatmap(cf_matrix, fmt=".0f", annot=True)
_ = plt.title("Confusion Matrix")
```

Solving Spotify Multiclass Genre Classification Problem



Finally, we will do the model Evaluation.

Model Evaluation

The following code evaluates the classification metrics on the test and ypred, where we can the precision, recall, F1score. Based on the values we can proceed with our model.

```
print(metrics.classification_report(ytest, ypred))
```

	precision	recall	f1-score	support
0	0.92	0.80	0.86	594
1	0.96	0.99	0.98	583
2	0.85	0.90	0.88	598
3	0.87	0.88	0.88	601
4	0.91	0.85	0.88	618
5	0.78	0.89	0.83	603
6	0.86	0.88	0.87	583
7	0.86	0.79	0.83	595
accuracy			0.87	4775
macro avg	0.88	0.87	0.87	4775
weighted avg	0.88	0.87	0.87	4775

Conclusion

In conclusion, we could categorize Spotify music genres with an accuracy of 88% using the analysis and modeling done in this study. Given the complexity and subjectivity in defining music genres, this is a reasonable level of accuracy. Yet, there is always an opportunity for improvement, and our analysis has a few limitations.

One disadvantage is the need for more diversity in our Dataset, primarily rap and hip-hop music on Spotify. This influenced our research and modeling in favor of specific genres. We must incorporate a wider variety of music genres into the Dataset to improve our model.

Another restriction is the likelihood of human mistakes in classifying the data, which might have resulted in genre categorization discrepancies. We may utilize more sophisticated approaches, such as deep learning models, to automatically label music based on auditory attributes to address this.

Our analysis and modeling give a solid foundation for categorizing Spotify music genres, but more study and improvements are required to increase the model's accuracy and resilience.

Key Takeaways

Solving Spotify Multiclass Genre Classification Problem

- Data cleaning and preprocessing are critical processes in preparing data for modeling and can significantly influence model performance.
- Early stopping approaches, such as monitoring validation loss and accuracy, can help to prevent model overfitting.
- Increase the dataset size, add features, and experiment with alternative methods and hyperparameters to enhance the classification model's performance.

The media shown in this article is not owned by Analytics Vidhya and is used at the Author's discretion.

blogathon categorization Dataframe dataset deep learning distribution library machine learning Models music pandas training

Previous Post

How to Reverse a String in Python in 5 Ways?

Next Post

Campus Recruitment: A Classification Problem with Logistic Regression

Solving Spotify Multiclass Genre Classification Problem

5 Free Data Science Projects With Solutions

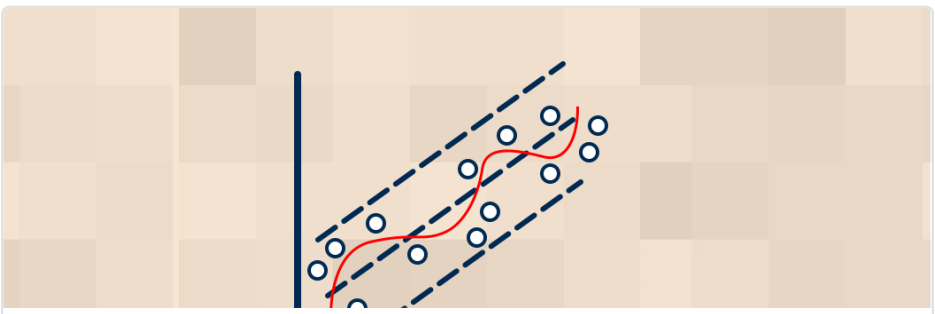
Nitika Sharma - SEP 23, 2023

10 Best AI Image Generator Tools to Use in 2023

avcontentteam - AUG 17, 2023

Top 20 Data Engineering Project Ideas [With Source Code]

Nitika Sharma - SEP 20, 2023



Everything you need to Know about Linear Regression!

KAVITA MALI - OCT 04, 2021