2306 AML 117 MAHESH BABU M ASSIGNMENT 15 OR ML8

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
data=pd.read_csv('data.csv')
data
```

Out[5]:

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness	key	liveness	loudness	mode	name	popularity
0	0.0594	1921	0.98200	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	0.878000	10	0.6650	-20.096	1	Piano Concerto No. 3 in D Minor, Op. 30: III. ...	4
1	0.9630	1921	0.73200	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	0.000000	7	0.1600	-12.441	1	Clancy Lowered the Boom	5
2	0.0394	1921	0.96100	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	0.328	500062	0.166	0	1o6I8BglA6ylDMrIELygv1	0.913000	3	0.1010	-14.850	1	Gati Bali	5
3	0.1650	1921	0.96700	['Frank Parker']	0.275	210000	0.309	0	3ftBPsC5vPBKxYSee08FDH	0.000028	5	0.3810	-9.316	1	Danny Boy	3
4	0.2530	1921	0.95700	['Phil Regan']	0.418	166693	0.193	0	4d6HGyGT8e121BsdKmw9v6	0.000002	3	0.2290	-10.096	1	When Irish Eyes Are Smiling	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
170648	0.6080	2020	0.08460	['Anuel AA', 'Daddy Yankee', 'KAROL G', 'Ozuna...	0.786	301714	0.808	0	0KkIkfsLEJbrcIhYsCL7L5	0.000289	7	0.0822	-3.702	1	China	72
170649	0.7340	2020	0.20600	['Ashnikko']	0.717	150654	0.753	0	0OStKKAuXlxA0fMH54Qs6E	0.000000	7	0.1010	-6.020	1	Halloweenie III: Seven Days	68
170650	0.6370	2020	0.10100	['MAMAMOO']	0.634	211280	0.858	0	4BZXVFYCb76Q0Klojq4piV	0.000009	4	0.2580	-2.226	0	AYA	76
170651	0.1950	2020	0.00998	['Eminem']	0.671	337147	0.623	1	5SiZJoLXp3WOl3J4C8IK0d	0.000008	2	0.6430	-7.161	1	Darkness	70
170652	0.6420	2020	0.13200	['KEVVO', 'J Balvin']	0.856	189507	0.721	1	7HmnJHfs0BkFzX4x8j0hkl	0.004710	7	0.1820	-4.928	1	Billetes Azules (with J Balvin)	74

170653 rows × 19 columns

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   valence           170653 non-null  float64
 1   year              170653 non-null  int64  
 2   acousticness      170653 non-null  float64
 3   artists           170653 non-null  object 
 4   danceability      170653 non-null  float64
 5   duration_ms       170653 non-null  int64  
 6   energy            170653 non-null  float64
 7   explicit          170653 non-null  int64  
 8   id                170653 non-null  object 
 9   instrumentalness  170653 non-null  float64
 10  key               170653 non-null  int64  
 11  liveness          170653 non-null  float64
 12  loudness          170653 non-null  float64
 13  mode              170653 non-null  int64  
 14  name              170653 non-null  object 
 15  popularity        170653 non-null  int64  
 16  release_date      170653 non-null  object 
 17  speechiness       170653 non-null  float64
 18  tempo             170653 non-null  float64
dtypes: float64(9), int64(6), object(4)
memory usage: 24.7+ MB
```

In [7]:
```
import seaborn as sns
```

In [8]:
```
sns.heatmap(data.corr(),annot=True)
```

```
C:\Users\sys\AppData\Local\Temp\ipykernel_10504\2578434383.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(data.corr(),annot=True)
```

Out[8]:
```
<Axes: >
```

In [9]:
```
artist=pd.read_csv('data_by_artist.csv')
artist
```

Out[9]:

	mode	count	acousticness	artists	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity	key
0	1	9	0.590111	"Cats" 1981 Original London Cast	0.467222	250318.555556	0.394003	0.011400	0.290833	-14.448000	0.210389	117.518111	0.389500	38.333333	5
1	1	26	0.862538	"Cats" 1983 Broadway Cast	0.441731	287280.000000	0.406808	0.081158	0.315215	-10.690000	0.176212	103.044154	0.268865	30.576923	5
2	1	7	0.856571	"Fiddler On The Roof" Motion Picture Chorus	0.348286	328920.000000	0.286571	0.024593	0.325786	-15.230714	0.118514	77.375857	0.354857	34.857143	0
3	1	27	0.884926	"Fiddler On The Roof" Motion Picture Orchestra	0.425074	262890.962963	0.245770	0.073587	0.275481	-15.639370	0.123200	88.667630	0.372030	34.851852	0
4	1	7	0.510714	"Joseph And The Amazing Technicolor Dreamcoat"...	0.467143	270436.142857	0.488286	0.009400	0.195000	-10.236714	0.098543	122.835857	0.482286	43.000000	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28675	1	2	0.512000	麥志誠	0.356000	198773.000000	0.306000	0.008970	0.108000	-10.119000	0.027700	150.049000	0.328000	35.000000	10
28676	0	2	0.541000	黃品源	0.578000	293840.000000	0.334000	0.000006	0.067500	-11.974000	0.026700	135.934000	0.243000	48.000000	9
28677	1	11	0.785455	黃國隆	0.570818	174582.727273	0.148400	0.000083	0.142191	-21.610091	0.054355	119.586273	0.741273	23.000000	5
28678	1	2	0.381000	黑豹	0.353000	316160.000000	0.686000	0.000000	0.056800	-9.103000	0.039500	200.341000	0.352000	35.000000	11
28679	1	2	0.568000	조정현	0.447000	237688.000000	0.215000	0.000001	0.064900	-16.478000	0.027200	71.979000	0.177000	31.000000	10

28680 rows × 15 columns

In [10]:
```
year=pd.read_csv('data_by_year.csv')
year
```

Out[10]:

	mode	year	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity	key
0	1	1921	0.886896	0.418597	260537.166667	0.231815	0.344878	0.205710	-17.048667	0.073662	101.531493	0.379327	0.653333	2
1	1	1922	0.938592	0.482042	165469.746479	0.237815	0.434195	0.240720	-19.275282	0.116655	100.884521	0.535549	0.140845	10
2	1	1923	0.957247	0.577341	177942.362162	0.262406	0.371733	0.227462	-14.129211	0.093949	114.010730	0.625492	5.389189	0
3	1	1924	0.940200	0.549894	191046.707627	0.344347	0.581701	0.235219	-14.231343	0.092089	120.689572	0.663725	0.661017	10
4	1	1925	0.962607	0.573863	184986.924460	0.278594	0.418297	0.237668	-14.146414	0.111918	115.521921	0.621929	2.604317	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	1	2016	0.284171	0.600202	221396.510295	0.592855	0.093984	0.181170	-8.061056	0.104313	118.652630	0.431532	59.647190	0
96	1	2017	0.286099	0.612217	211115.696787	0.590421	0.097091	0.191713	-8.312630	0.110536	117.202740	0.416476	63.263554	1
97	1	2018	0.267633	0.663500	206001.007133	0.602435	0.054217	0.176326	-7.168785	0.127176	121.922308	0.447921	63.296243	1
98	1	2019	0.278299	0.644814	201024.788096	0.593224	0.077640	0.172616	-7.722192	0.121043	120.235644	0.458818	65.256542	1
99	1	2020	0.219931	0.692904	193728.397537	0.631232	0.016376	0.178535	-6.595067	0.141384	124.283129	0.501048	64.301970	1

100 rows × 14 columns

In [11]:
```
genres=pd.read_csv('data_w_genres.csv')
genres
```

Out[11]:

	genres	artists	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity	key	mode	count
0	['show tunes']	"Cats" 1981 Original London Cast	0.590111	0.467222	250318.555556	0.394003	0.011400	0.290833	-14.448000	0.210389	117.518111	0.389500	38.333333	5	1	9
1	[]	"Cats" 1983 Broadway Cast	0.862538	0.441731	287280.000000	0.406808	0.081158	0.315215	-10.690000	0.176212	103.044154	0.268865	30.576923	5	1	26
2	[]	"Fiddler On The Roof" Motion Picture Chorus	0.856571	0.348286	328920.000000	0.286571	0.024593	0.325786	-15.230714	0.118514	77.375857	0.354857	34.857143	0	1	7
3	[]	"Fiddler On The Roof" Motion Picture Orchestra	0.884926	0.425074	262890.962963	0.245770	0.073587	0.275481	-15.639370	0.123200	88.667630	0.372030	34.851852	0	1	27
4	[]	"Joseph And The Amazing Technicolor Dreamcoat"...	0.510714	0.467143	270436.142857	0.488286	0.009400	0.195000	-10.236714	0.098543	122.835857	0.482286	43.000000	5	1	7
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28675	[]	麥志誠	0.512000	0.356000	198773.000000	0.306000	0.008970	0.108000	-10.119000	0.027700	150.049000	0.328000	35.000000	10	1	2
28676	['c-pop', 'classic cantopop', 'classic mandopo...	黃品源	0.541000	0.578000	293840.000000	0.334000	0.000006	0.067500	-11.974000	0.026700	135.934000	0.243000	48.000000	9	0	2
28677	[]	黃國隆	0.785455	0.570818	174582.727273	0.148400	0.000083	0.142191	-21.610091	0.054355	119.586273	0.741273	23.000000	5	1	11
28678	['chinese indie', 'chinese indie rock']	黑豹	0.381000	0.353000	316160.000000	0.686000	0.000000	0.056800	-9.103000	0.039500	200.341000	0.352000	35.000000	11	1	2
28679	['classic korean pop']	조정현	0.568000	0.447000	237688.000000	0.215000	0.000001	0.064900	-16.478000	0.027200	71.979000	0.177000	31.000000	10	1	2

28680 rows × 16 columns

In [12]:
```
print(artist.size)
```

```
430200
```

In [14]:
```
print('artist',artist.size)
print('artist',artist.shape)
print('artist',artist.columns)
print('year',year.size)
print('year',year.shape)
print('year',year.columns)
print('genres',genres.size)
print('genres',genres.shape)
print('genres',genres.columns)
```

```
artist 430200
artist (28680, 15)
artist Index(['mode', 'count', 'acousticness', 'artists', 'danceability',
       'duration_ms', 'energy', 'instrumentalness', 'liveness', 'loudness',
       'speechiness', 'tempo', 'valence', 'popularity', 'key'],
      dtype='object')
year 1400
year (100, 14)
year Index(['mode', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
       'valence', 'popularity', 'key'],
      dtype='object')
genres 458880
genres (28680, 16)
genres Index(['genres', 'artists', 'acousticness', 'danceability', 'duration_ms',
       'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness',
       'tempo', 'valence', 'popularity', 'key', 'mode', 'count'],
      dtype='object')
```

In [15]:
```
artist.info()
year.info()
genres.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28680 entries, 0 to 28679
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   mode              28680 non-null  int64  
 1   count             28680 non-null  int64  
 2   acousticness      28680 non-null  float64
 3   artists           28680 non-null  object 
 4   danceability      28680 non-null  float64
 5   duration_ms       28680 non-null  float64
 6   energy            28680 non-null  float64
 7   instrumentalness  28680 non-null  float64
 8   liveness          28680 non-null  float64
 9   loudness          28680 non-null  float64
 10  speechiness       28680 non-null  float64
 11  tempo             28680 non-null  float64
 12  valence           28680 non-null  float64
 13  popularity        28680 non-null  float64
 14  key               28680 non-null  int64  
dtypes: float64(11), int64(3), object(1)
memory usage: 3.3+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   mode              100 non-null    int64  
 1   year              100 non-null    int64  
 2   acousticness      100 non-null    float64
 3   danceability      100 non-null    float64
 4   duration_ms       100 non-null    float64
 5   energy            100 non-null    float64
 6   instrumentalness  100 non-null    float64
 7   liveness          100 non-null    float64
 8   loudness          100 non-null    float64
 9   speechiness       100 non-null    float64
 10  tempo             100 non-null    float64
 11  valence           100 non-null    float64
 12  popularity        100 non-null    float64
 13  key               100 non-null    int64  
dtypes: float64(11), int64(3)
memory usage: 11.1 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28680 entries, 0 to 28679
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   genres            28680 non-null  object 
 1   artists           28680 non-null  object 
 2   acousticness      28680 non-null  float64
 3   danceability      28680 non-null  float64
 4   duration_ms       28680 non-null  float64
 5   energy            28680 non-null  float64
 6   instrumentalness  28680 non-null  float64
 7   liveness          28680 non-null  float64
 8   loudness          28680 non-null  float64
 9   speechiness       28680 non-null  float64
 10  tempo             28680 non-null  float64
 11  valence           28680 non-null  float64
 12  popularity        28680 non-null  float64
 13  key               28680 non-null  int64  
 14  mode              28680 non-null  int64  
 15  count             28680 non-null  int64  
dtypes: float64(11), int64(3), object(2)
memory usage: 3.5+ MB
```

In [19]:
```
artist.dtypes
year.dtypes
genres.dtypes
```

Out[19]:
```
genres               object
artists              object
acousticness        float64
danceability        float64
duration_ms         float64
energy              float64
instrumentalness    float64
liveness            float64
loudness            float64
speechiness         float64
tempo               float64
valence             float64
popularity          float64
key                   int64
mode                  int64
count                 int64
dtype: object
```

In [21]:
```
artist['artists'].unique()
```

Out[21]:
```
array(['"Cats" 1981 Original London Cast', '"Cats" 1983 Broadway Cast',
       '"Fiddler On The Roof" Motion Picture Chorus', ..., '黃國隆', '黑豹',
       '조정현'], dtype=object)
```

In [22]:
```
year['year'].unique()
```

Out[22]:
```
array([1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931,
       1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942,
       1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953,
       1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964,
       1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975,
       1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986,
       1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997,
       1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008,
       2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019,
       2020], dtype=int64)
```

In [23]:
```
genres['genres'].unique()
```

Out[23]:
```
array(["['show tunes']", '[]', "['comedy rock', 'comic', 'parody']", ...,
       "['mainland chinese pop', 'zhongguo feng']",
       "['c-pop', 'classic mandopop', 'mainland chinese pop', 'mandopop']",
       "['chinese indie', 'chinese indie rock']"], dtype=object)
```

In [33]:
```
features = artist[['acousticness','danceability','duration_ms','energy','instrumentalness','liveness','loudness','speechiness',
'tempo','valence','key','mode' ]]
target = artist['popularity']

labels = artist['popularity']
```

In [34]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [35]:
```
models = []
models.append(('KNN', KNeighborsClassifier()))
models.append(('SVC', SVC()))
models.append(('LR', LogisticRegression()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('GNB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('GB', GradientBoostingClassifier()))
```

In [58]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [59]:
```
from imblearn.over_sampling import SMOTE
from sklearn.datasets import make_classification

# Create a toy imbalanced dataset for demonstration
X, y = make_classification(n_classes=2, class_sep=2,
                           weights=[0.1, 0.9], n_informative=3, n_redundant=1,
                           flip_y=0, n_features=20, n_clusters_per_class=1,
                           n_samples=1000, random_state=42)

# Instantiate the SMOTE object
smote = SMOTE(sampling_strategy='auto', random_state=42)

# Fit and transform the dataset to generate synthetic samples
X_resampled, y_resampled = smote.fit_resample(X, y)

# Check the class distribution after SMOTE
import numpy as np
unique, counts = np.unique(y_resampled, return_counts=True)
print(dict(zip(unique, counts)))
```

```
{0: 900, 1: 900}
```

In [60]:
```
X1=X_resampled
y1=y_resampled
```

In [61]:
```
X_train, X_test, y_train, y_test = train_test_split(X1, y1, stratify = y1, random_state=0)
```

In [62]:
```
names = []
scores = []
for name, model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))
    names.append(name)
tr_split = pd.DataFrame({'Name': names, 'Score': scores})
print(tr_split)
```

```
  Name     Score
0  KNN  1.000000
1  SVC  1.000000
2   LR  1.000000
3   DT  1.000000
4  GNB  0.997778
5   RF  1.000000
6   GB  1.000000
```

In [63]:
```
acc = []
model = []
```

In [64]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\anaconda\lib\site-packages (1.7.6)
Requirement already satisfied: scipy in c:\anaconda\lib\site-packages (from xgboost) (1.10.0)
Requirement already satisfied: numpy in c:\anaconda\lib\site-packages (from xgboost) (1.23.5)
```

In [65]:
```
from sklearn import metrics
```

In [68]:
```
import xgboost as xgb
XB = xgb.XGBClassifier()
XB.fit(X_train,y_train)

predicted_values = XB.predict(X_test)

x = metrics.accuracy_score(y_test, predicted_values)
acc.append(x)
model.append('XGBoost')
print("XGBoost's Accuracy is: ", x)

print(classification_report(y_test,predicted_values))
```

```
XGBoost's Accuracy is:  1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       225
           1       1.00      1.00      1.00       225

    accuracy                           1.00       450
   macro avg       1.00      1.00      1.00       450
weighted avg       1.00      1.00      1.00       450
```

In [75]:
```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=20, random_state=0)
RF.fit(X_train,y_train)

predicted_values = RF.predict(X_test)

x = metrics.accuracy_score(y_test, predicted_values)
acc.append(x)
model.append('RF')
print("RF's Accuracy is: ", x)

print(classification_report(y_test,predicted_values))
```

```
RF's Accuracy is:  1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       225
           1       1.00      1.00      1.00       225

    accuracy                           1.00       450
   macro avg       1.00      1.00      1.00       450
weighted avg       1.00      1.00      1.00       450
```

In [77]:
```
accuracy_models = dict(zip(model, acc))
for k, v in accuracy_models.items():
    print (k, '-->', v)
```

```
XGBoost --> 1.0
RF --> 1.0
```

In [80]:
```
data = np.array([[1,9,0.5901,0.46,250318.55,0.394003333,0.011399851,0.290833333,-14.448,0.210388889,117.5181111,0.3895,38.333,5,

20,3,5,12,5,5]])
prediction = RF.predict(data)
print(prediction)
```

```
[1]
```

In [ ]:

In [ ]: